Artificial Intelligence For Beginners

Ahmad Ali AlZubi

Abdulrhman A. Alkhanifer

Artificial Intelligence for Beginners

India | UAE | Nigeria | Uzbekistan | Montenegro | Iraq | Egypt | Thailand | Uganda | Philippines | Indonesia
www.iarapublication.com

Artificial Intelligence for Beginners

Authored By:

Ahmad Ali AlZubi
Professor, Computer Science Department, King Saud University, Riyadh, Saudi Arabia

Abdulrhman A. Alkhanifer
Assistant Professor, Computer Science Department, King Saud University, Riyadh, Saudi Arabia

First Impression: July 2024

Artificial Intelligence for Beginners

ISBN: 978-81-19481-79-8

Rs. 1000/- ($80)

Published by:
IARA Publication

Dedicated to

My Parents

Ali AlZubi and Ghazalah AlZubi

Preface

Artificial Intelligence (AI) represents a frontier of technology aimed at creating systems capable of performing tasks that traditionally require human intelligence. These tasks include learning from experience, recognizing patterns, understanding natural language, and making decisions. AI integrates various disciplines such as computer science, mathematics, psychology, and cognitive science to build systems that can simulate aspects of human cognition. The ultimate goal is to develop machines that can execute complex tasks with a high degree of autonomy, efficiency, and accuracy, enhancing capabilities across diverse fields from healthcare to finance and beyond.

Machine Learning (ML) is a subset of AI that focuses on developing algorithms capable of learning from data. Unlike traditional programming, where explicit instructions are given to the machine, ML algorithms are designed to identify patterns and make predictions based on historical data. Techniques such as supervised learning, unsupervised learning, and reinforcement learning form the backbone of modern ML, enabling systems to improve their performance over time without human intervention. This self-improving nature of ML is crucial for developing systems that can adapt to new data and environments, making them incredibly versatile and powerful.

Natural Language Processing (NLP) is another critical component of AI, enabling machines to understand, interpret, and generate human language. NLP technologies power applications like chatbots, virtual assistants, and translation services, facilitating more natural and intuitive interactions

between humans and machines. Advances in NLP have been driven by techniques such as neural networks and deep learning, which have significantly enhanced the ability of systems to understand context, sentiment, and nuances in language. This progress has made it possible for AI systems to engage in more sophisticated conversations and provide meaningful responses, bridging the gap between human and machine communication.

Robotics, when combined with AI, results in intelligent machines capable of performing a variety of tasks in dynamic environments. AI powered robots are equipped with sensors, actuators, and sophisticated algorithms that allow them to navigate, manipulate objects, and interact with humans and their surroundings. From industrial robots performing precise manufacturing tasks to service robots assisting in healthcare and hospitality, AI-driven robotics is transforming industries by increasing efficiency, safety, and productivity. The integration of AI with robotics also paves the way for innovations in areas such as space exploration, disaster response, and personalized medicine, where human presence is limited or impractical.

The ethical implications of AI are a significant area of consideration, as intelligent systems become increasingly integrated into everyday life. Issues such as privacy, security, bias, and job displacement are at the forefront of discussions surrounding AI development and deployment. Ensuring that AI systems are fair, transparent, and accountable is essential to fostering trust and acceptance among users and stakeholders. Researchers, policymakers, and industry leaders are working collaboratively to establish guidelines, standards, and regulations that promote ethical AI practices while

encouraging innovation and progress. This balance is crucial for harnessing the full potential of AI while mitigating its risks and challenges.

Artificial Intelligence for Beginners is not just a technical exploration but a journey into the transformative power of technology. As AI continues to evolve, it promises to redefine the boundaries of what machines can achieve, augmenting human capabilities and solving some of the most complex challenges facing society today. The ongoing advancements in AI technologies and their applications hold the potential to revolutionize industries, improve quality of life, and create new opportunities for growth and innovation. Embracing this future requires a commitment to continuous learning, ethical consideration, and collaboration across disciplines to ensure that AI benefits everyone and contributes to a more sustainable and equitable world.

This book delves into the fascinating world of artificial intelligence, providing insights into the mechanisms and innovations that drive intelligent systems, transforming industries and everyday life.

Acknowledgement

We would like to thank and heartfelt gratitude to King Saud University, Riyadh, Saudi Arabia for their supports.

We extend our heartfelt gratitude to the individuals who have contributed to the fruition of this endeavour, "**Artificial Intelligence for Beginners**" First and foremost, we express our deepest appreciation to the specialists in Artificial Intelligence whose expertise and insights have shaped the content of this book, enabling a comprehensive understanding of the Artificial Intelligence.

We extend our thanks to our colleagues for their invaluable support and encouragement throughout this journey.

We would like to thank our family and loved ones for their constant support, comprehension, and inspiration during the many hours that I have invested in the writing, research, and editing of this text.

Lastly, we express gratitude to the readers for their interest and trust in this work, with the hope that it serves as a meaningful resource in the field of Artificial Intelligence.

Ahmad Ali AlZubi

Abdulrhman A. Alkhanifer

Contents

1

Introduction

Artificial Intelligence System (AIS) was a distributed computing project undertaken by Intelligence Realm, Inc. with the long-term goal of simulating the human brain in real time, complete with artificial consciousness and artificial general intelligence. They claimed to have found, in research, the "mechanisms of knowledge representation in the brain which is equivalent to finding artificial intelligence", before moving into the developmental phase.

Science

The project's initial goal was recreating the largest brain simulation to date, performed by neuroscientist Eugene M. Izhikevich of The Neurosciences Institute in San Diego, California. Izhikevich simulated 1 second of activity of 100 billion neurons (the estimated number of neurons in the human brain) in 50 days using a cluster of 27 3-gigahertz processors. He extrapolated that a real-time simulation of the brain could not be achieved before 2016. The project aimed to disprove this prediction.

On July 12, 2008, AIS announced that the first phase of the project had been completed by reaching the 100 billion neuron mark. The project then continued to simulate neurons while they completed the development of the other applications.

AIS simulated the brain via an artificial neural network, and used Hodgkin–Huxley models. The project utilized the BOINC distributed computing platform. In version 1.08 of the software each work unit received by a volunteer simulated 500,000 neurons for 100 milliseconds at 5 millisecond time steps (the estimated firing rate of a human neuron).

The application had four primary modules—for creating neurons, simulating neurons, visualizing neurons, and finally, knowledge acquisition.

Intention was that the neuronal generator would eventually use genetic algorithms to generate neurons for simulation. The neuron simulator used mathematical models to simulate those neurons. Initially, Hodgkin–Huxley models were used, but more models (perhaps hundreds) were intended to be utilized in the future. The visualization software was to allow the administrators to monitor and control the neuronal simulators. The knowledge acquisition module involved feeding information to the system and training it to build its knowledge base.

The AIS project had successfully simulated over 700 billion neurons by April 2009.

The project was closed in November 2010 as the BOINC program of the project did not work.

THE IMPACT OF ARTIFICIAL INTELLIGENCE

Artificial Intelligence (AI) is becoming an important part of our daily life, in social as well as the business environment. From healthcare to the military, this technology is being introduced in all the sectors to reduce human effort and give an accurate and faster result.

We are fortunate to live in this generation, which is full of technological advancements. Now we live in a time where a lot of work is taken over by machines & software. AI has a special place in all the advancement made today. As we know that AI is the science of computers and machines developing intelligence like humans. In this technology, the machines can do some of the simples to complex tasks that we as humans need to do regularly.

The AI systems are capable enough to reduce human efforts in numerous areas. To conduct different operations in the industry, many of them are using artificial intelligence to create machines that perform various activities regularly. The artificial intelligence applications help to get the work done faster and with accurate results.

General Advantages

While AI has been very useful in many domains like healthcare, automotive etc, there are some general advantages you get in any field by applying AI. Let us have a look at some of them:

Daily operations

Computed methods for automated reasoning, learning and perception

have become a common phenomenon in our everyday lives. We have our Siri or Cortana to help us out.

The smartphone is an apt and everyday example of how we use AI. We are also hitting the road for long drives and trips with the help of GPS. In utilities, we find that they can predict what we are going to type and correct the spellings. That is machine intelligence at work.

When we take a picture, the AI algorithm identifies and detects the person's face and tags the individuals when we are posting our photographs on social media sites.

Fewer Errors

AI helps us in reducing the errors and the chance of reaching accuracy with a greater degree of precision. It is applied in various studies such as exploration of space. Intelligent robots are fed with data and are sent to explore space. Since they are more resistant and have a greater ability to endure the space and hostile atmosphere due to their metal bodies. They are built and acclimatized in such a way that they cannot be altered or get damaged or malfunction in a hostile environment.

Repetitive Tasks

Repetitive tasks are monotonous in nature can be carried out with the help of machine intelligence. Machines think faster than humans and can be put to multi-tasking. Machine intelligence can be employed to carry out dangerous tasks. Their parameters, unlike humans, can be adjusted. Their speed and time are calculation based parameters only.

When humans play a computer game or run a computer-controlled robot, we are actually interacting with artificial intelligence. In the game we are playing, the computer is our opponent. The machine intelligence plans the game movement in response to our movements. We can consider gaming to be the most common use of the benefits of artificial intelligence.

Difficult Exploration

Artificial intelligence and the science of robotics can be put to use in mining and other fuel exploration processes. Not only that, these complex machines can be used for exploring the ocean floor and hence overcome the human limitations.

Due to the programming of the robots, they can perform more laborious and hard work with greater responsibility. Moreover, they do not wear out easily.

Digital Assistants

Highly advanced organizations use 'avatars' which are replicas or digital assistants who can actually interact with the users, thus saving the need for human resources.

For artificial thinkers, emotions come in the way of rational thinking and are not a distraction at all. The complete absence of the emotional side, makes the robots think logically and take the right program decisions. Emotions are associated with moods that can cloud judgment and affect human efficiency. This is completely ruled out for machine intelligence.

Availability 24x7

Unlike humans, machines do not require frequent breaks and refreshments. They are programmed for long hours and can continuously perform without getting bored or distracted or even tired.

Domain-wise Advantages

AI for Good

'AI for Good' is a United Nations platform. It is centred around an annual Global Summit that promotes the exchange on the beneficial use of AI by building specific projects.

The purpose of organizing global summits that are action-oriented, came from an existing discussion in AI research being dominated by research streams such as the Netflix Prize (improve the movie recommendation algorithm). The AI for Good series aims to bring forward AI research topics that contribute towards more global obstacles, in particular through the Sustainable Development Goals, while at the same time avoiding typical UN-style conferences where results are usually more abstract.

Healthcare

The main purpose of healthcare AI applications is to examine relationships between prevention or treatment techniques and patient outcomes. AI programs have been built and implemented to practices such as diagnosis processes, treatment protocol development, drug development, personalized medicine, and patient monitoring and care. Many medical institutions have developed AI algorithms for their departments.

Large technology companies and even startups have also developed AI algorithms for healthcare. Additionally, hospitals are looking to AI

solutions to support operational initiatives that increase cost-saving, improve patient satisfaction, and satisfy their staffing and workforce needs.

Companies are also developing predictive analytics solutions that help healthcare managers improve business operations through increasing utilization, decreasing patient boarding, reducing the length of stay and optimizing staffing levels.

Agriculture

In agriculture, new AI developments show advances in gaining yield and to increase the research and development of growing crops. AI now predicts the time it takes for a crop like a vegetable to be ripe and ready for picking thus increasing the efficiency of farming. These advances go on including Crop and Soil Monitoring, Agricultural Robots, and Predictive Analytics. Crop and soil monitoring uses new algorithms and data collected in the field to manage and track the health of crops making it easier and more sustainable for the farmers.

More specializations of AI in agriculture is one such as greenhouse automation, simulation, modeling and optimization techniques.

Due to the rise in population and the increase in demand for food in the future, there will need to be at least a 70% boost in yield from agriculture to support this new demand. More and more of the public perceives that the adaption of these new techniques and the use of AI will help reach that goal.

Automative

The Air Operations Division (AOD) uses AI for the rule-based expert systems. The AOD has use for artificial intelligence for surrogate operators for combat and training simulators, mission management aids, support systems for tactical decision making, and post-processing of the simulator data into symbolic summaries.

The AOD also uses artificial intelligence in speech recognition software. The air traffic controllers (ATCs) are giving directions to the artificial pilots and the AOD wants the pilots to respond to the ATC's with simple responses. The programs that incorporate the speech software must be trained, which means they use neural networks. This is an early stage of the program that has plenty of room for improvement. The improvements are imperative because ATCs use very specific dialogue and the software needs to be able to communicate correctly and promptly every time.

Aviation

AI-supported Design of Aircraft is used to help designers in the process of creating conceptual designs of aircraft.

This program empowers the designers to concentrate more on the design itself and less on the design process. The software also allows the user to focus less on software tools. The AIDA uses rule-based systems to compute its data. This is a diagram of the arrangement of the AIDA modules. Although simple, the program is proving effective.

Haitham Baomar and Peter Bentley are leading a team from the University College of London to develop an artificial intelligence-based Intelligent Autopilot System (IAS) designed to teach an autopilot system to behave like a highly experienced pilot who is faced with an emergency situation such as severe weather, turbulence, or system failure.

Education

One of the more promising innovations is the idea of a personal AI tutor or assistant for each individual student. Because a single teacher can't work with every student at once, AI tutors would allow for students to get extra, one-on-one help in areas of needed growth.

There are many new possibilities due to what has been coined by The New York Times as "The Great AI Awakening." One of these possibilities mentioned by Forbes included the providing of adaptive learning programs, which assess and react to a student's emotions and learning preferences.

Many teachers fear the idea of AI replacing them in the classroom, especially with the idea of personal AI assistants for each student. The reality is, AI can create a more dystopian environment with revenge effects. It is inevitable that AI technologies will be taking over the classroom in the years to come, thus it is essential that the kinks of these new innovations are worked out before teachers decide whether or not to implement them into their daily schedules.

Arts

Artificial Intelligence has inspired numerous creative applications including its usage to produce visual art. The recent AI-based exhibitions provide a good overview of the historical applications of AI for art, architecture, and design.

These exhibitions showcasing the usage of AI to produce art include the Google-sponsored benefit and auction at the Gray Area Foundation

in San Francisco, where artists experimented with the DeepDream algorithm.

The Association of Computing Machinery dedicated a special magazine issue to the subject of computers and art highlighting the role of machine learning in the arts.

Finance & Economics

The 1980s was really when AI started to become prominent in the finance world. This is when expert systems became more of a commercial product in the financial field. Their first function was to help give financial plans for people with incomes over a threshold. That then led to the Client Profiling System that was used for a specific band of incomes. The 1990s was a lot more about fraud detection. One of the systems that were started in 1993 was able to review over lacs of transactions per week and over two years it helped identify 400 potential cases of money laundering which would have been equal to $1 billion. Although expert systems did not last in the finance world, it did help jump-start the use of AI and help make it what it is today.

These days AI is prominent in the following use cases in the financial world:

- Algorithmic Trading
- Market Analysis
- Personal Finance
- Underwriting

Government

The potential uses of AI in government are wide and varied, with recent research suggesting that 'Cognitive technologies could eventually revolutionize every facet of government operations'. Experts in government consulting suggest these type of government problems are appropriate for AI applications:

- Resource allocation
- Large public & employees datasets
- Experts shortage
- Predictable scenarios
- Procedural tasks
- Diverse data

Military

The main military applications of Artificial Intelligence and Machine Learning are to enhance C2, Communications, Sensors, Integration and Interoperability.

Artificial Intelligence technologies enable coordination of sensors and effectors, threat detection and identification, marking of enemy positions, target acquisition, coordination and deconfliction of distributed Join Fires between networked combat vehicles and tanks also inside Manned and Unmanned Teams (MUM-T).

Gaming

In video games, artificial intelligence is routinely used to generate dynamic purposeful behaviour in non-player characters (NPCs). In addition, well-understood AI techniques are routinely used for pathfinding.

Some researchers consider NPC AI in games to be a "solved problem" for most production tasks. Games with more atypical AI include the AI director of Left 4 Dead (2008) and the neuroevolutionary training of platoons in Supreme Commander 2 (2010).

News & Publishing

Companies are making computer-generated news and reports commercially available, including summarizing team sporting events based on statistical data from the game in English and also financial reports and real estate analyses.

There are software firms that help publishers increase traffic by 'intelligently' posting articles on social media platforms such as Facebook and Twitter. Another firm uses AI to turn structured data into intelligent comments and recommendations in natural language such as financial reports, executive summaries, personalized sales or marketing documents.

Yet another firm has launched an app that is designed to learn how to best engage each individual reader with the exact articles — sent through the right channel at the right time — that will be most relevant to the reader. There are firms which are helping media companies with their AI-powered video personalization and programming platform.

Advertising

It is possible to use AI to predict or generalize the behaviour of

customers from their digital footprints in order to target them with personalized promotions or build customer personas automatically. A documented case reports that online gambling companies were using AI to improve customer targeting.

Moreover, the application of Personality computing AI models can help to reduce the cost of advertising campaigns by adding psychological targeting to more traditional sociodemographic or behavioural targeting.

Disadvantages

As every bright side has a darker version in it, AI also has some disadvantages. Let's have a look at some of them:

AI Bias

One concern is that AI programs may be programmed to be biased against certain groups, such as women and minorities because most of the developers are wealthy Caucasian men. Recent researches show that support for artificial intelligence is higher among men than women.

Algorithms have a host of applications in today's legal system already, assisting officials ranging from judges to parole officers and public defenders in gauging the predicted likelihood of recidivism of defendants.

An AI-based criminal offender profiling application assigns an exceptionally elevated risk of recidivism to black defendants while, conversely, ascribing low-risk estimate to white defendants significantly more often than statistically expected.

Impact on Employment

The relationship between automation and employment has always been complicated. While automation eliminates old jobs, it also creates new jobs through micro-economic and macro-economic effects. Unlike previous waves of automation, many middle-class jobs may be eliminated by artificial intelligence.

The Economist states that 'the worry that AI could do to white-collar jobs what steam power did to blue-collar ones during the Industrial Revolution' is 'worth taking seriously'. Jobs at extreme risk range from paralegals to fast-food cooks, while job demand is likely to increase for care-related professions ranging from personal healthcare to the clergy.

Many futurists warn that these jobs may be automated in the next couple of decades and that many of the new jobs may not be 'accessible

to people with average capability', even with retraining. Economists point out that in the past technology has tended to increase rather than reduce total employment, but acknowledge that 'we're in uncharted territory' with AI.

Effect on Humanity

Some experts suggest that AI applications cannot, by definition, successfully simulate genuine human empathy and that the use of AI technology in fields such as customer service or psychotherapy was deeply misguided. Few experts are also bothered that AI researchers (and some philosophers) were willing to view the human mind as nothing more than a computer program (a position is now known as computationalism) which imply that AI research devalues human life.

Autonomous Weapons

Currently, many countries are researching battlefield robots, including the United States, China, Russia, and the United Kingdom. Many people concerned about risk from superintelligent AI also want to limit the use of artificial soldiers and drones.

Threat to Our Existence

Physicist Stephen Hawking, Microsoft founder Bill Gates, and SpaceX founder Elon Musk have expressed concerns about the possibility that AI could evolve to the point that humans could not control it, with Hawking theorizing that this could 'spell the end of the human race'.

For this danger to be realized, the hypothetical AI would have to overpower or out-think all of humanity, which a minority of experts argue is a possibility far enough in the future to not be worth researching. Other counterarguments revolve around humans being either intrinsically or convergently valuable from the perspective of AI.

Conclusion

Creating artificial intelligence is perhaps the biggest event for mankind. If used and developed constructively, we can use artificial intelligence to eradicate poverty and hunger from the human race.

The argument that will we ever achieve that supreme level of AI ever is ongoing. The creators and perpetrators of artificial intelligence insist that machine intelligence is beneficial and has been created to help the human race.

The power of artificial intelligence that inadvertently causes destruction and damage cannot be ignored. What will help us control it better is research and in-depth study of the importance of artificial intelligence. Research alone can control the potentially harmful consequences of AI and help us enjoy the fruit of this innovation.

AI will not only change the way we think or live our lives but also explores new horizons, even if its space or the ocean. Humans are getting continually better in defining their desires and quickly transforming this desire into reality. Things will happen so fast that we will not notice the minor changes and will be easily adaptable to the change it brings to us.

AI IN MYTH, FICTION AND SPECULATION

Mechanical men and artificial beings appear in Greek myths, such as the golden robots of Hephaestus and Pygmalion's Galatea. In the Middle Ages, there were rumors of secret mystical or alchemical means of placing mind into matter, such as Jâbir ibn Hayyân's *Takwin*, Paracelsus' homunculus and Rabbi Judah Loew's Golem. By the 19th century, ideas about artificial men and thinking machines were developed in fiction, as in Mary Shelley's *Frankenstein* or Karel Èapek's *R.U.R. (Rossum's Universal Robots)*, and speculation, such as Samuel Butler's "Darwin among the Machines." AI has continued to be an important element of science fiction into the present.

Automatons

Realistic humanoid automatons were built by craftsman from every civilization, including Yan Shi, Hero of Alexandria, Al-Jazari , Pierre Jaquet-Droz, and Wolfgang von Kempelen. The oldest known automatons were the sacred statues of ancient Egypt and Greece. The faithful believed that craftsman had imbued these figures with very real minds, capable of wisdom and emotion—Hermes Trismegistus wrote that "by discovering the true nature of the gods, man has been able to reproduce it."

Formal reasoning

Artificial intelligence is based on the assumption that the process of human thought can be mechanized. The study of mechanical—or "formal"—reasoning has a long history. Chinese, Indian and Greek philosophers all developed structured methods of formal deduction in the first millennium BCE. Their ideas were developed over the centuries by philosophers such as Aristotle (who gave a formal analysis of the syllogism),

Euclid (whose *Elements* was a model of formal reasoning), al-Khwârizmî (who developed algebra and gave his name to "algorithm") and European scholastic philosophers such as William of Ockham and Duns Scotus.

Spanish philosopher Ramon Llull (1232–1315) developed several *logical machines* devoted to the production of knowledge by logical means; Llull described his machines as mechanical entities that could combine basic and undeniable truths by simple logical operations, produced by the machine by mechanical meanings, in such ways as to produce all the possible knowledge. Llull's work had a great influence on Gottfried Leibniz, who redeveloped his ideas.

In the 17th century, Leibniz, Thomas Hobbes and René Descartes explored the possibility that all rational thought could be made as systematic as algebra or geometry. Hobbes famously wrote in *Leviathan*: "reason is nothing but reckoning". Leibniz envisioned a universal language of reasoning (his *characteristica universalis*) which would reduce argumentation to calculation, so that "there would be no more need of disputation between two philosophers than between two accountants. For it would suffice to take their pencils in hand, down to their slates, and to say each other (with a friend as witness, if they liked): *Let us calculate*." These philosophers had begun to articulate the physical symbol system hypothesis that would become the guiding faith of AI research.

In the 20th century, the study of mathematical logic provided the essential breakthrough that made artificial intelligence seem plausible. The foundations had been set by such works as Boole's *The Laws of Thought* and Frege's *Begriffsschrift*. Building on Frege's system, Russell and Whitehead presented a formal treatment of the foundations of mathematics in their masterpiece, the *Principia Mathematica* in 1913. Inspired by Russell's success, David Hilbert challenged mathematicians of the 1920s and 30s to answer this fundamental question: "can all of mathematical reasoning be formalized?" His question was answered by Gödel's incompleteness proof, Turing's machine and Church's Lambda calculus.

Their answer was surprising in two ways. First, they proved that there were, in fact, limits to what mathematical logic could accomplish. But second (and more important for AI) their work suggested that, within these limits, *any* form of mathematical reasoning could be mechanized. The Church-Turing thesis implied that a mechanical device, shuffling symbols as simple as 0 and 1, could imitate any conceivable process of mathematical deduction. The key insight was the Turing machine—a simple theoretical construct that captured the essence of abstract symbol

manipulation. This invention would inspire a handful of scientists to begin discussing the possibility of thinking machines.

Computer science

Calculating machines were built in antiquity and improved throughout history by many mathematicians, including (once again) philosopher Gottfried Leibniz. In the early 19th century, Charles Babbage designed a programmable computer (the Analytical Engine), although it was never built. Ada Lovelace speculated that the machine "might compose elaborate and scientific pieces of music of any degree of complexity or extent". (She is often credited as the first programmer because of a set of notes she wrote that completely detail a method for calculating Bernoulli numbers with the Engine.)

The first modern computers were the massive code breaking machines of the Second World War (such as Z3, ENIAC and Colossus). The latter two of these machines were based on the theoretical foundation laid by Alan Turing and developed by John von Neumann.

AGENT ENVIRONMENT IN AI

An environment is everything in the world which surrounds the agent, but it is not a part of an agent itself. An environment can be described as a situation in which an agent is present.

The environment is where agent lives, operate and provide the agent with something to sense and act upon it. An environment is mostly said to be non-feministic.

Features of Environment

As per Russell and Norvig, an environment can have various features from the point of view of an agent:

Fully observable vs Partially Observable

- If an agent sensor can sense or access the complete state of an environment at each point of time then it is a fully observable environment, else it is partially observable.
- A fully observable environment is easy as there is no need to maintain the internal state to keep track history of the world.
- An agent with no sensors in all environments then such an environment is called as unobservable.

Deterministic vs Stochastic

- If an agent's current state and selected action can completely determine the next state of the environment, then such environment is called a deterministic environment.
- A stochastic environment is random in nature and cannot be determined completely by an agent.
- In a deterministic, fully observable environment, agent does not need to worry about uncertainty.

Episodic vs Sequential

- In an episodic environment, there is a series of one-shot actions, and only the current percept is required for the action.
- However, in Sequential environment, an agent requires memory of past actions to determine the next best actions.

Single-agent vs Multi-agent

- If only one agent is involved in an environment, and operating by itself then such an environment is called single agent environment.
- However, if multiple agents are operating in an environment, then such an environment is called a multi-agent environment.
- The agent design problems in the multi-agent environment are different from single agent environment.

Static vs Dynamic

- If the environment can change itself while an agent is deliberating then such environment is called a dynamic environment else it is called a static environment.
- Static environments are easy to deal because an agent does not need to continue looking at the world while deciding for an action.
- However for dynamic environment, agents need to keep looking at the world at each action.
- Taxi driving is an example of a dynamic environment whereas Crossword puzzles are an example of a static environment.

Discrete vs Continuous

- If in an environment there are a finite number of percepts and actions that can be performed within it, then such an environment

is called a discrete environment else it is called continuous environment.

- A chess gamecomes under discrete environment as there is a finite number of moves that can be performed.
- A self-driving car is an example of a continuous environment.

Known vs Unknown

- Known and unknown are not actually a feature of an environment, but it is an agent's state of knowledge to perform an action.
- In a known environment, the results for all actions are known to the agent. While in unknown environment, agent needs to learn how it works in order to perform an action.
- It is quite possible that a known environment to be partially observable and an Unknown environment to be fully observable.

Accessible vs Inaccessible

- If an agent can obtain complete and accurate information about the state's environment, then such an environment is called an Accessible environment else it is called inaccessible.
- Information about an event on earth is an example of Inaccessible environment.

UNDERSTAND TYPES OF ENVIRONMENTS IN ARTIFICIAL INTELLIGENCE

The Environment is the surrounding world around the agent which is not part of the agent itself. It's important to understand the nature of the environment when solving a problem using artificial intelligence. For example, program a chess bot, the environment is a chessboard and creating a room cleaner robot, the environment is Room.

Each environment has its own properties and agents should be designed such as it can explore environment states using sensors and act accordingly using actuators. In this guide, we're going to understand all types of environments with real-life examples.

Fully Observable vs Partially-Observable

In a fully observable environment, The Agent is familiar with the complete state of the environment at a given time. There will be no portion of the environment that is hidden for the agent.

Real-life Example: While running a car on the road (Environment), The driver (Agent) is able to see road conditions, signboard and pedestrians on the road at a given time and drive accordingly. So Road is a fully observable environment for a driver while driving the car.

Real-life Example: Playing card games is a perfect example of a partially-observable environment where a player is not aware of the card in the opponent's hand. Why partially-observable? Because the other parts of the environment, e.g opponent, game name, etc are known for the player (Agent).

Deterministic vs Stochastic

Deterministic are the environments where the next state is observable at a given time. So there is no uncertainty in the environment.

Real-life Example: The traffic signal is a deterministic environment where the next signal is known for a pedestrian (Agent)

The Stochastic environment is the opposite of a deterministic environment. The next state is totally unpredictable for the agent. So randomness exists in the environment.

Real-life Example: The radio station is a stochastic environment where the listener is not aware about the next song or playing a soccer is stochastic environment.

Episodic vs Sequential

Episodic is an environment where each state is independent of each other. The action on a state has nothing to do with the next state.

Real-life Example: A support bot (agent) answer to a question and then answer to another question and so on. So each question-answer is a single episode.

The sequential environment is an environment where the next state is dependent on the current action. So agent current action can change all of the future states of the environment.

Real-life Example: Playing tennis is a perfect example where a player observes the opponent's shot and takes action.

Static vs Dynamic

The Static environment is completely unchanged while an agent is precepting the environment.

Real-life Example: Cleaning a room (Environment) by a dry-cleaner reboot (Agent) is an example of a static environment where the room is static while cleaning.

Dynamic Environment could be changed while an agent is precepting the environment. So agents keep looking at the environment while taking action.

Real-life Example: Playing soccer is a dynamic environment where players' positions keep changing throughout the game. So a player hit the ball by observing the opposite team.

Discrete vs Continuous

Discrete Environment consists of a finite number of states and agents have a finite number of actions.

Real-life Example: Choices of a move (action) in a tic-tac game are finite on a finite number of boxes on the board (Environment).

While in a Continuous environment, the environment can have an infinite number of states. So the possibilities of taking an action are also infinite.

Real-life Example: In a basketball game, the position of players (Environment) keeps changing continuously and hitting (Action) the ball towards the basket can have different angles and speed so infinite possibilities.

Single Agent vs Multi-Agent

Single agent environment where an environment is explored by a single agent. All actions are performed by a single agent in the environment.

Real-life Example: Playing tennis against the ball is a single agent environment where there is only one player.

If two or more agents are taking actions in the environment, it is known as a multi-agent environment.

Real-life Example: Playing a soccer match is a multi-agent environment.

TYPE OF ARTIFICIAL INTELLIGENCE

Artificial intelligence can be divided into three subfields:

- Artificial intelligence
- Machine learning
- Deep learning.

Machine Learning

Machine learning is the art of study of algorithms that learn from examples and experiences.

Machine learning is based on the idea that there exist some patterns in the data that were identified and used for future predictions.

The difference from hardcoding rules is that the machine learns on its own to find such rules.

Deep learning

Deep learning is a sub-field of machine learning. Deep learning does not mean the machine learns more in-depth knowledge; it means the machine uses different layers to learn from the data. The depth of the model is represented by the number of layers in the model. For instance, Google LeNet model for image recognition counts 22 layers.

In deep learning, the learning phase is done through a neural network. A neural network is an architecture where the layers are stacked on top of each other.

AI vs. Machine Learning

Most of our smartphone, daily device or even the internet uses Artificial intelligence. Very often, AI and machine learning are used interchangeably by big companies that want to announce their latest innovation. However, Machine learning and AI are different in some ways.

AI- artificial intelligence- is the science of training machines to perform human tasks. The term was invented in the 1950s when scientists began exploring how computers could solve problems on their own.

Artificial Intelligence is a computer that is given human-like properties. Take our brain; it works effortlessly and seamlessly to calculate the world around us. Artificial Intelligence is the concept that a computer can do the same. It can be said that AI is the large science that mimics human aptitudes.

Machine learning is a distinct subset of AI that trains a machine how to learn. Machine learning models look for patterns in data and try to conclude.

In a nutshell, the machine does not need to be explicitly programmed by people. The programmers give some examples, and the computer is going to learn what to do from those samples.

Where is AI used? Examples

AI has broad applications-

- Artificial intelligence is used to reduce or avoid the repetitive task. For instance, AI can repeat a task continuously, without fatigue. In fact, AI never rests, and it is indifferent to the task to carry out
- Artificial intelligence improves an existing product. Before the age of machine learning, core products were building upon hard-code rule. Firms introduced artificial intelligence to enhance the functionality of the product rather than starting from scratch to design new products. You can think of a Facebook image. A few years ago, you had to tag your friends manually. Nowadays, with the help of AI, Facebook gives you a friend's recommendation.

AI is used in all the industries, from marketing to supply chain, finance, food-processing sector. According to a McKinsey survey, financial services and high tech communication are leading the AI fields.

Why is AI booming now?

A neural network has been out since the nineties with the seminal paper of Yann LeCun. However, it started to become famous around the year 2012. Explained by three critical factors for its popularity are:

1. Hardware
2. Data
3. Algorithm

Machine learning is an experimental field, meaning it needs to have data to test new ideas or approaches. With the boom of the internet, data became more easily accessible. Besides, giant companies like NVIDIA and AMD have developed high-performance graphics chips for the gaming market.

Hardware

In the last twenty years, the power of the CPU has exploded, allowing the user to train a small deep-learning model on any laptop. However, to process a deep-learning model for computer vision or deep learning, you need a more powerful machine. Thanks to the investment of NVIDIA and AMD, a new generation of GPU (graphical processing unit) are available. These chips allow parallel computations. It means the machine can separate the computations over several GPU to speed up the calculations.

For instance, with an NVIDIA TITAN X, it takes two days to train a model called ImageNet against weeks for a traditional CPU. Besides, big companies use clusters of GPU to train deep learning model with the NVIDIA Tesla K80 because it helps to reduce the data center cost and provide better performances.

Data

Deep learning is the structure of the model, and the data is the fluid to make it alive. Data powers the artificial intelligence. Without data, nothing can be done. Latest Technologies have pushed the boundaries of data storage. It is easier than ever to store a high amount of data in a data center.

Internet revolution makes data collection and distribution available to feed machine learning algorithm. If you are familiar with Flickr, Instagram or any other app with images, you can guess their AI potential.

There are millions of pictures with tags available on these websites. Those pictures can be used to train a neural network model to recognize an object on the picture without the need to manually collect and label the data.

Artificial Intelligence combined with data is the new gold. Data is a unique competitive advantage that no firm should neglect. AI provides the best answers from your data. When all the firms can have the same technologies, the one with data will have a competitive advantage over the other. To give an idea, the world creates about 2.2 exabytes, or 2.2 billion gigabytes, every day.

A company needs exceptionally diverse data sources to be able to find the patterns and learn and in a substantial volume.

Algorithm

Hardware is more powerful than ever, data is easily accessible, but one thing that makes the neural network more reliable is the development of more accurate algorithms. Primary neural networks are a simple multiplication matrix without in-depth statistical properties. Since 2010, remarkable discoveries have been made to improve the neural network. Artificial intelligence uses a progressive learning algorithm to let the data do the programming. It means, the computer can teach itself how to perform different tasks, like finding anomalies, become a chatbot.

ARTIFICIAL GENERAL INTELLIGENCE

Artificial intelligence is a branch of computer science that attempts to understand the essence of intelligence and produce a new intelligent machine that responds in a manner similar to human intelligence.

Research in this area includes robotics, speech recognition, image recognition, Natural language processing and expert systems. Since the birth of artificial intelligence, the theory and technology have become more and more mature, and the application fields have been expanding.

It is conceivable that the technological products brought by artificial intelligence in the future will be the "container" of human wisdom. Artificial intelligence can simulate the information process of human consciousness and thinking.

Artificial intelligence is not human intelligence, but it can be like human thinking, and it may exceed human intelligence. Artificial general intelligence is also referred to as "strong AI", "full AI" or as the ability of a machine to perform "general intelligent action".

Academic sources reserve "strong AI" to refer to machines capable of experiencing consciousness.

REVOLUTION OF AI IN 2020! IS IT REAL?

Well, if you like reading technology news, you may have come across various facts and stats about the revolutionizing technology: Artificial Intelligence.

Some of the facts are like:

- By the year 2025, the global market of AI is expected to be approximately $60 billion; in the year 2016 it was $1.4 billion.

- The global GDP of countries will grow by $15.7 trillion by the year 2030 thanks to AI revolution.
- AI based software can increase the productivity of business by almost 40%.
- Startups based on AI grew almost 14 times over the last two decades.
- Investment in the AI startups and businesses grew 6 times since 2000.
- Already approximately 77% of the devices that we use feature AI in one form or another.
- According to Google analysts, it is believed that by the year 2020, robots will be smart enough to mimic the complex behavior of human like flirting and jokes.

With these stats, facts and quotes, it is quite evident that artificial intelligence will definitely be a big thing in the near future. In fact, the mania of AI is that 41 percent of consumers are expecting that their life will sooner or later be changed with AI. Just like these stats, there have been various predictions about the Artificial intelligence future but have you ever wondered from where the concept of AI started,

But before analyzing that let's know the exact meaning of artificial intelligence

Artificial intelligence means to impart the abilities of human brains to the computer-based software. The AI-based web or mobile applications are able to perform the actual functions which were earlier restricted to humans. The logical & cognitive thinking are now seamlessly performed by this latest technology.

There is no doubt that artificial intelligence along with machine learning and deep learning has revolutionized the working of software. They have become more supportive, interactive and humanly in their performance and preciseness.

Almost six decades ago, various technology enthusiasts started exploring this marvelous technology. Although its concept was in existence from the past few centuries since the 1950s there has been no hindrance in its evolution.

Evolution of Artificial Intelligence: Timeline

1. Alan Turing developed a test to determine whether a machine can have human intelligent behavior or not.

2. A computer scientist from America, John McCarthy coined the term Artificial Intelligence in 1955.
3. In 1961, First Robot was introduced at general motors.
4. In 1965, MIT AI laboratory-created Eliza, the first chatbot on Natural Language Processing (NLP)
5. Garry Kasparov, the chess world champion was defeated in 1997 by a chess-playing computer called IBM's Deep Blue.
6. In 1999, MIT AI laboratory-developed Kismet, the first emotional AI demo.
7. The development of self-driving cars was started by Google in 2009.
8. 2011 saw a lot in AI. Jeopardy champions were defeated by IBM Watson. Cortana, Google Now and Siri became popular.
9. World champion Lee Sedol was defeated in Go (an ancient Chinese board game) by Google DeepMind's AlphaGo in 2016.
10. 2017- the year when skin cancer and heart rhythms were diagnosed by medical breakthroughs with AI.

AI evolution of AI in the past 70 years is really revolutionary. From the 1950s to the present, Artificial intelligence along with its subsets, deep learning and machine learning have reached every industry with the crucial contribution of companies, computer scientists and software developers.

The position of artificial intelligence at present

At present, artificial intelligence is one of the key areas where various software development companies in India & rest of the world are willing to invest.

Neglecting this technology— at the present times— can bring repercussions for your business. Have you ever wondered, total how many companies have no or unclear plans to use AI? Only 22 percent! This statistics was found in a review study by MIT Sloan Management and Boston Consulting group.

This technology is seen as a strategically important service which has to be added in the offerings of businesses of every field, from tourism to healthcare.

There are a number of AI tech that is ruling the software development world. You can incorporate them into your business to reap the benefits of artificial intelligence.

AI technologies

Biometrics

Maximum security has been a major concern in human civilizations from the times they have been established. Different ways have been evolved over centuries to provide maximum security and AI is contributing to the same. Do you have any idea how artificial intelligence is helpful to this world for increasing security? You all can connect with the face recognition and fingerprint unlocking option in your mobile phones. This is one of the best examples of the impact of Artificial Intelligence on human lifestyle.

As per an international digital security company, Gemalto, Biometrics is the most obvious means of authenticating and identifying individuals in the year 2019." It is not fixed to the recognition of image & speech. At present, visual biometric devices can recognize iris, finger, retina and face. These AI based devices are used to effectively authenticate the person when an authorized entry is allowed.

MACHINE LEARNING

Machine learning popularity as a means to achieve artificial intelligence is on the rise. This is the subset of AI and allows the machines to learn but it doesn't require any software programmer to stuff them with the codes!

Machine learning basically works on algorithms based on statistical calculations and methods. Some of the best Machine learning packages which are used extensively in the year 2019 are IBM's SPSS, Apache Spark ML, Scikit learn and Microsoft Azure.

Various software applications and software products are being developed on the concept of machine learning for example image processing, dynamic web search, self-driving cars, to name a few. Google maps which are being used to navigate a place works on the algorithms of machine learning.

Robotic process automation

All across the world, robotics have given a great boom to the rise of AI. Various software products have been replaced with the interactions of human after the onset of robotic process automation. This basically involves

developing robots which can provide process information and sensory feedback.

By using of this AI technologies, you can perform various tasks such as solving a query, performing complex calculations, logging in and logging out. All these tasks can be done with reliability and accuracy at a faster rate.

Nowadays, researchers believe that these services have the capabilities of reaching out to the areas of finance, human resources, and accounting. Many of these processes in various industries from hospitality to healthcare has been automated with robots.

Internet of things

Internet of things or IoT is commonly used in combination with AI. Their applications and benefits are defining the trends of software industry. One of the most popular examples is automated vacuum cleaners which came into existence in 2002.

However, it is not broadly recognized as other AI applications. Some other examples of integration of AI with IoT are self-driving vehicles and smart thermostat solutions.

Also, it is expected that in future, these technologies will be used in combination for emotional analysis, security devices, and face recognition.

This combination can easily be explained in a nutshell as a computer program communicating with the device. You can also assume AI to be the brain of the body (IoT).

Neural networks

The human brain works with neurons that are connected in a complex manner. In the same way, a kind of programming is being developed that makes machines to work on the neural networks. It is known as Artificial Neural Network (ANN) programming and developed under the deep learning technology. Machine learning is the parent technology of deep learning which is succeeded with the AI.

Artificial neural networks first came into application in the year 1989. Some representatives from Carnegie Mellon University were the first people to develop an autonomous vehicle by using neural networks. In addition to this, as the science has evolved over the years, right now we are more concerned about how our brain actually works rather than concentrating on stimulating the brain. Actually, it is improving our

knowledge about ourselves as well as bringing progress in neuroscience with the developments in AI or artificial intelligence.

GENERAL ARTIFICIAL INTELLIGENCE

Before reaching superintelligence, general AI means that a machine will have the same cognitive capabilities as a human being. Again, researchers argue about the point in time when we will reach general AI. It could be around the year 2045. Due to the law of accelerating returns, the phase of general Artificial Intelligence will very soon after transition into the era of superintelligence.

Intelligence is highly related to the ability of an organism to effectively adapt itself to better cope with a changing environment. Adaption also means not only changing oneself but also changing the environment. Here are the key characteristics of general artificial (and human) intelligence (sources: here and here):

- Learning: The ability to alter one's behaviour based on past experiences, e.g. when encountering new and unseen situations.
- Memory: The encoding, storage and retrieval of past experiences.
- Reasoning and abstraction: Draw logical conclusions and have the ability to generalize / derive rules based on sample data.
- Problem solving: The ability to systematically come up with possible solutions and derive the best answer to a problem.
- Divergent thinking: The ability to generate multiple solutions to a given problem.
- Convergent thinking: Narrow down a list of multiple options in order to derive the best possible answer.
- Emotional intelligence: Recognise and interpret human emotions.
- Speed: All characteristics mentioned above must happen in a reasonable time frame / near real time. Also they cannot rely on massive amounts of data, e.g. to retrain a neural network. In some cases, learning can be based on one single example only.

General-purpose AI will not be reached with the current approach and tools like neural networks. In order to achieve true intelligence, cognitive systems might be a solution.

Narrow Artificial Intelligence

Since researches did not make much progress on achieving general

AI, the focus naturally shifted towards narrow AI first. Narrow AI focuses on very specific use cases. This means that an AI (currently deep neural network are mostly used) is trained for a very specific purpose. Therefore the AI can only handle events that it has been trained on.

If a chatbot is trained to answer customer service requests in English for company A, it will not be able to react to any requests that it will a) receive for a different company or b) are stated in a different language or c) be asked for non related topics like "Do I need to take an umbrella to work today?".

In contrast to general AI, narrow AI's capability to learn is very limited. It can learn within the boundaries of the particular use case, e.g. a narrow AI for speech recognition might be able to improve the rate of understanding new dialects of the same language it has been trained on.

In order to learn another language, it needs input from humans, e.g. by providing large amounts of labeled data sets for the new language to be learned.

Narrow AI cannot dynamically adapt to new situations by coming up with alternative solutions, a key trait of general artificial intelligence.

Some typical narrow AI use cases:

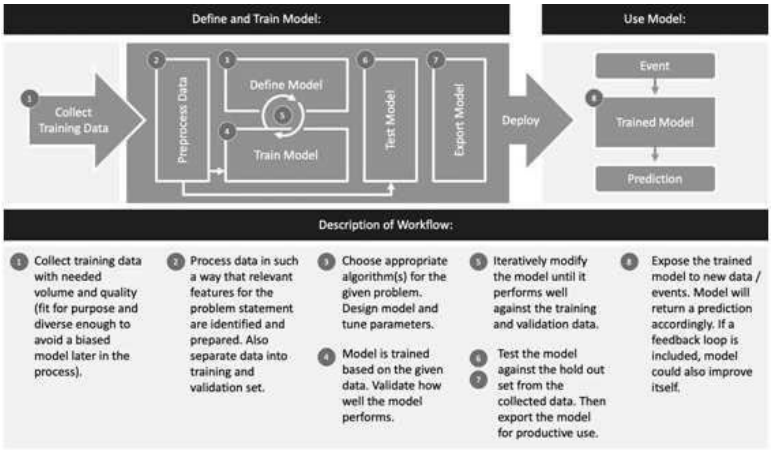
- Natural language recognition and processing
- Autonomous driving
- Visual image recognition and interpretation
- Intrusion detection in cyber security
- Human activity recognition

TOOLS AND METHODS IN MACHINE LEARNING

Machine learning is an AI discipline and the key driver behind the advances of narrow Artificial Intelligence in recent years. It is a collection of tools and methods which allow computers to learn from observations, data and examples in order to improve their performance. It does not require explicit step by step explanation on how to execute a task as it would be necessary with traditional programming.

In essence, machine learning is based on statistical models in order to conduct predictions. During the learning process, the parameters of the statistical model are optimally adapted according to the provided training data. One can say, that the system learns by experience. Based on the given training data, the system is then able to make predictions for unseen

events / data. In order for a model to generalize well for those unseen events, it is important to define a model well suited for the problem statement and to have training data available in the necessary quantity and quality.



Supervised Machine Learning

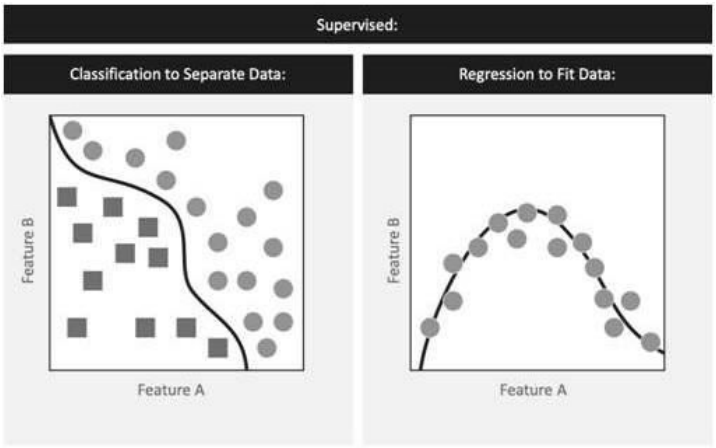
Supervised machine learning simply means that we provide an algorithm with fully labeled training data.

Labeled training data means that every training record also has the associated answer attached to it. Therefore we already provide clear guidance to the algorithm on how to interpret the data. The algorithm will then construct a model and test it against the given labels. For each iteration, an error function will validate, if the prediction that the model came up with is in alignment with the label. The model will then be adjusted slightly with each iteration until the model performs well on the training data.

There are two different types of supervised learning. The key thing to remember: Classification separates the data, regression fits the data.

- Classification: Algorithm that predicts the categorical response value for a given observation / input. It will predict a discrete value. A prerequisite is that we are able to segment the response values into distinct classes. A typical example is the classification of handwritten letters and numbers, e.g. used by the postal office to interpret an address written on a letter.
- Regression: Algorithm that predicts a numerical continuous response value for a given observation / input. It will predict a continuous

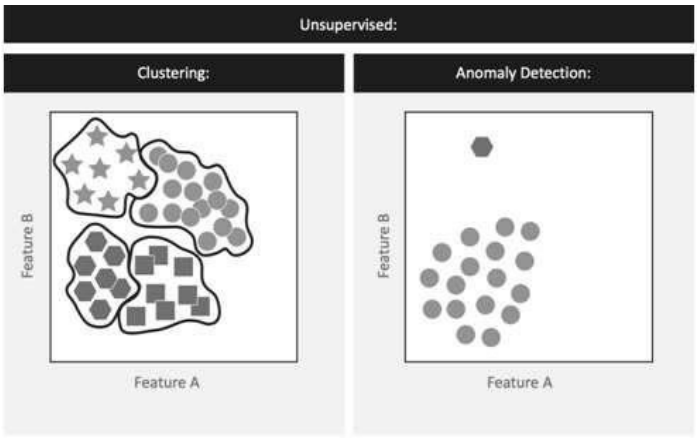
value. A typical example is the prediction of house prices based on their location, size, age, etc.



Unsupervised Machine Learning

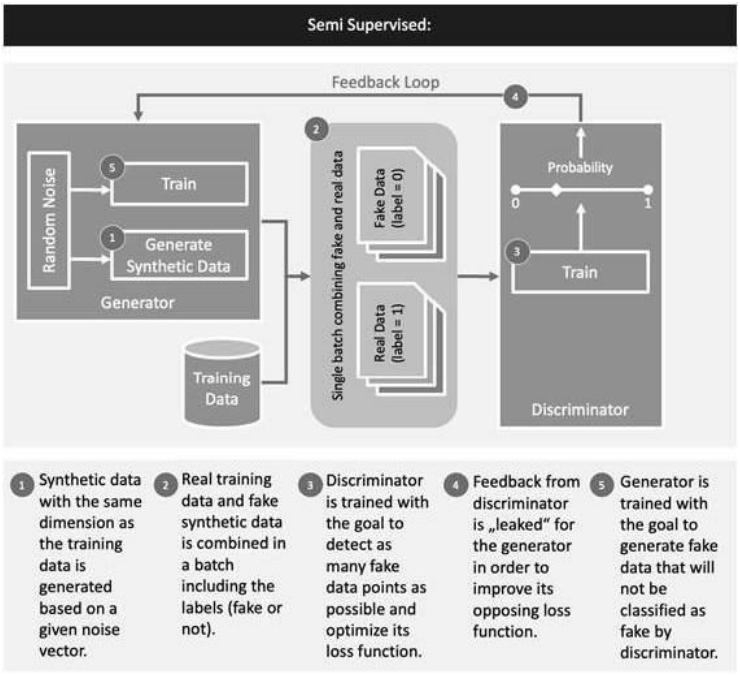
In cases where it is difficult or too expensive to obtain enough labeled training data, unsupervised methods become relevant. With unsupervised learning, we do not provide the algorithm with labeled training data. Instead we want the algorithm to find a way on its own how to classify / separate the data. The following key categories exist:

- Clustering: Algorithm that tries to create clusters of similar data points. For example, one can image that a machine will definitely be able to classify vehicles on its own when the algorithm receives only the front view of millions of cars. A BMW looks different compared to Ford or Audi therefore the algorithm should easily be able to cluster the pictures into different categories (in this case car brands).
- Anomaly Detection: Algorithm that tries to find outliers / anomalies within a given data set. A typical example is the identification of fraudulent behavior in bank transactions. If a person has many transactions while travelling between Europe and the US with relatively small amounts, a USD 10,000 transaction from Nigeria in between might be suspicious.
- Association: Association rules is an algorithm that identifies relationships and dependencies within data sets. A typical example would be a recommender engine of an online store where correlated products can be grouped together based on similar features and then suggested to the consumer.



Semi Supervised Machine Learning

In cases where labeling of massive amounts of training data is too expensive, a mix between supervised and supervised machine learning comes into play.



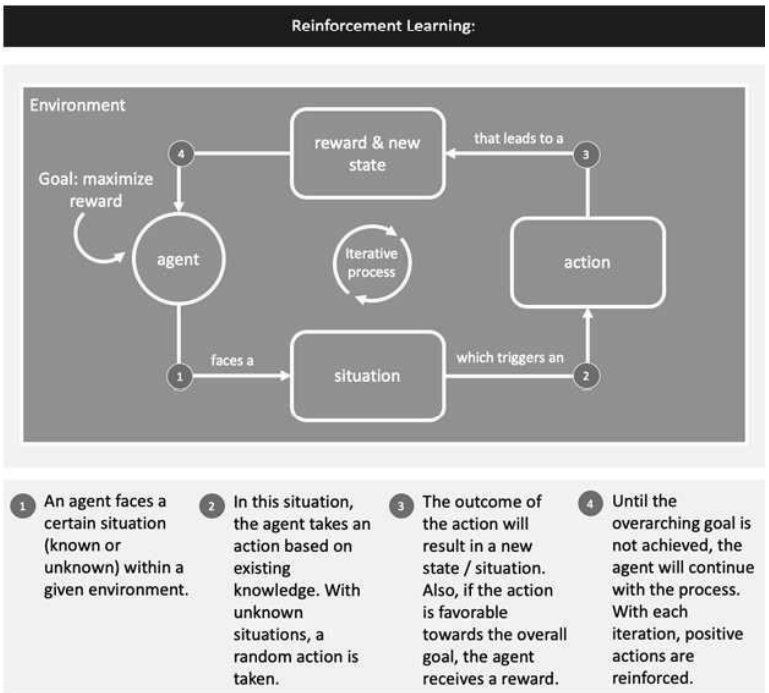
The process starts off with a person labeling a small amount of data (example: CT scans which are interpreted and labeled by an expert radiologist). Currently, the most popular example of semi supervised learning is a a special kind of neural network called GAN (general adversarial network).

It is using labeled training data to generate new data (via the generator network) and then sends this data to another neural network (discriminator) which needs to identify if the data is fake or part of the training data. The generator and discriminator networks jointly improve in a positive feedback loop.

The generator will get better at creating convincing fake data and the discriminator is getting better at separating fake data from training data. A great use case for a GAN is the creation of artificial images that should mimic real images. A GAN generated portrait was recently sold at Christie's for a ridiculously high amount.

Reinforcement Learning

Reinforcement learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment to achieve a specific goal. Unlike supervised learning, where the model is trained on a fixed dataset, RL involves learning through trial and error, receiving feedback in the form of rewards or punishments. This approach is inspired by behavioral psychology and is particularly useful for solving complex problems that require sequential decision-making.



Reinforcement learning is based on the assumption that an optimal behavior or action can be enforced by positive rewards that are given for a favorable action. The basic setup for reinforcement learning comprises an agent which will interact with its environment. Based on the positive / negative feedback the agent receives from its interactions at a given situation, a certain behavior is rewarded and favorable actions are therefore reinforced.

Within many iterations, the agent will be trained and the performance of its actions to achieve the overall goal are improved.

For each action, the agent must decide whether he wants to explore his environment (exploration) in order to search for potentially higher rewards (also the risk of failing is higher) compared to the actions that have already proven to be successful (exploitation).

2

Artificial Intelligence Systems in Computer Projects

AI – CURRENT STATUS OF TECHNOLOGY

AI – A Maturing Technology- A general definition of AI is the capability of a computer system to perform tasks that normally require human intelligence, such as visual perception, speech recognition and decision-making. Functionally, AI enabled machines should have the capability to learn, reason, judge, predict, infer and initiate action. In layman's terms, AI implies trying to emulate the brain. There are three main ingredients that are necessary for simulating intelligence: the brain, the body, and the mind. The brain consists of the software algorithms which work on available data, the body is the hardware and the mind is the computing power that runs the algorithms. Technological breakthroughs and convergence in these areas is enabling the AI field to rapidly mature.

AI, Machine Learning and Deep Learning – Year before last, in a significant development, Google Deepmind's AlphaGo program defeated South Korean Master Lee Se-dol in the popular board game Go, and the terms AI, Machine Learning, and Deep Learning were used to describe how DeepMind won. The easiest way to think of their inter-relationship is to visualise them as concentric circles, with AI the largest, then Machine Learning, and finally Deep Learning – which is driving today's AI explosion – fitting inside both.1 AI is any technique that enables computers to mimic human intelligence.

Machine Learning is a subset of AI, which focuses on the development of computer programs that can change when exposed to new data, by searching through data to look for patterns and adjusting program actions

accordingly. Deep Learning is a further subset of Machine Learning that is composed of algorithms which permit software to train itself by exposing multi-layered neural networks (which are designed on concepts borrowed from a study of the neurological structure of the brain) to vast amounts of data.

AI Technologies – The most significant technologies which are making rapid progress today are natural language processing and generation, speech recognition, text analytics, machine learning and deep learning platforms, decision management, biometrics and robotic process automation.

Some of the major players in this space are: Google, now famous for its artificial neural network based AlphaGo program; Facebook, which has recently announced several new algorithms; IBM, known for Watson, which is a cognitive system that leverages machine learning to derive insights from data; Microsoft, which helps developers to build Android, iOS and Windows apps using powerful intelligence algorithms; Toyota, which has a major focus on automotive autonomy (driver-less cars); and Baidu Research, the Chinese firm which brings together global research talent to work on AI technologies.

AI – Future Prospects -. Today, while AI is most commonly cited for image recognition, natural language processing and voice recognition, this is just an early manifestation of its full potential. The next step will be the ability to reason, and in fact reach a level where an AI system is functionally indistinguishable from a human. With such a capability, AI based systems would potentially have an infinite number of applications.

The Turing Test – In a 1951 paper, Alan Turing proposed the Turing Test to test for artificial intelligence. It envisages two contestants consisting of a human and a machine, with a judge, suitably screened from them, tasked with deciding which of the two is talking to him. While there have been two well-known computer programs claiming to have cleared the Turing Test, the reality is that no AI system has been able to pass it since it was introduced. Turing himself thought that by the year 2000 computer systems would be able to pass the test with flying colours! While there is much disagreement as to when a computer will actually pass the Turing Test, one thing all AI scientists generally agree on is that it is very likely to happen in our lifetime.

Fear of AI – There is a growing fear that machines with artificial intelligence will get so smart that they will take over and end civilisation.

This belief is probably rooted in the fact that most of society does not have an adequate understanding of this technology. AI is less feared in engineering circles because there is a slightly more hands-on understanding of the technology. There is perhaps a potential for AI to be abused in the future, but that is a possibility with any technology. Apprehensions about AI leading to end-of-civilisation scenarios are perhaps largely based on fear of the unknown, and are largely unfounded.

APPROACH OF ARTIFICIAL INTELLIGENCE

Theories

In the quest to create intelligent machines, the field of Artificial Intelligence has split into several different approaches based on the opinions about the most promising methods and theories. These rivaling theories have lead researchers in one of two basic approaches; bottom-up and top-down.

Bottom-up theorists believe the best way to achieve artificial intelligence is to build electronic replicas of the human brain's complex network of neurons, while the top-down approach attempts to mimic the brain's behaviour with computer programmes.

Parallel Computation

The human brain is made up of a web of billions of cells called neurons, and understanding its complexities is seen as one of the last frontiers in scientific research. It is the aim of AI researchers who prefer this bottom-up approach to construct electronic circuits that act as neurons do in the human brain. Although much of the working of the brain remains unknown, the complex network of neurons is what gives humans intelligent characteristics. By itself, a neuron is not intelligent, but when grouped together, neurons are able to pass electrical signals through networks. The neuron "firing", passing a signal to the next in the chain.

Research has shown that a signal received by a neuron travels through the dendrite region, and down the axon. Separating nerve cells is a gap called the synapse. In order for the signal to be transferred to the next neuron, the signal must be converted from electrical to chemical energy. The signal can then be received by the next neuron and processed. Based on experiments with neurons, McCulloch and Pitts showed that neurons might be considered devices for processing binary numbers.

An important back of mathematic logic, binary numbers (represented as 1's and 0's or true and false) were also the basis of the electronic computer. A century earlier the true/false nature of binary numbers was theorized in 1854 by George Boole in his postulates concerning the Laws of Thought. Boole's principles make up what is known as Boolean algebra, the collection of logic concerning AND, OR, NOT operands.

Boole also assumed that the human mind works according to these laws, it performs logical operations that could be reasoned. Ninety years later, Claude Shannon applied Boole's principles in circuits, the blueprint for electronic computers. Boole's contribution to the future of computing and Artificial Intelligence was immeasurable, and his logic is the basis of neural networks.

McCulloch and Pitts, using Boole's principles, wrote a paper on neural network theory. The thesis dealt with how the networks of connected neurons could perform logical operations. It also stated that, one the level of a single neuron, the release or failure to release an impulse was the basis by which the brain makes true/false decisions. Using the idea of feedback theory, they described the loop which existed between the senses —> brain —> muscles, and likewise concluded that Memory could be defined as the signals in a closed loop of neurons. Although we now know that logic in the brain occurs at a level higher then McCulloch and Pitts theorized, their contributions were important to AI because they showed how the firing of signals between connected neurons could cause the brains to make decisions. McCulloch and Pitt's theory is the basis of the artificial neural network theory.

Using this theory, McCulloch and Pitts then designed electronic replicas of neural networks, to show how electronic networks could generate logical processes. They also stated that neural networks may, in the future, be able to learn, and recognize patterns. The results of their research and two of Weiner's books served to increase enthusiasm, and laboratories of computer simulated neurons were set up across the country.

Two major factors have inhibited the development of full scale neural networks. Because of the expense of constructing a machine to simulate neurons, it was expensive even to construct neural networks with the number of neurons in an ant. Although the cost of components have decreased, the computer would have to grow thousands of times larger to be on the scale of the human brain. The second factor is current computer architecture. The standard Von Neuman computer, the architecture of

nearly all computers, lacks an adequate number of pathways between components. Researchers are now developing alternate architectures for use with neural networks.

Even with these inhibiting factors, artificial neural networks have presented some impressive results. Frank Rosenblatt, experimenting with computer simulated networks, was able to create a machine that could mimic the human thinking process, and recognize letters. But, with new top-down methods becoming popular, parallel computing was put on hold. Now neural networks are making a return, and some researchers believe that with new computer architectures, parallel computing and the bottom-up theory will be a driving factor in creating artificial intelligence.

Expert Systems

Because of the large storage capacity of computers, expert systems had the potential to interpret statistics, in order to formulate rules. An expert system works much like a detective solves a mystery. Using the information, and logic or rules, an expert system can solve the problem.

Games

On game with strong AI ties is chess. World-champion chess playing programmes can see ahead twenty plus moves in advance for each move they make. In addition, the programmes have an ability to get progress ably better over time because of the ability to learn.

Chess programmes do not play chess as humans do. In three minutes, Deep Thought (a master programme) considers 126 million moves, while human chess master on average considers less than 2 moves. Herbert Simon suggested that human chess masters are familiar with Favourable board positions, and the relationship with thousands of pieces in small areas. Computers on the other hand, do not take hunches into account. The next move comes from exhaustive searches into all moves, and the consequences of the moves based on prior learning. Chess programmes, running on Cray super computers have attained a rating of senior master, in the range of Gary Kasparov, the Russian world champion.

Frames

On method that many programmes use to represent knowledge are frames. Pioneered by Marvin Minsky, frame theory revolves around packets of information. For example, say the situation was a birthday party. A

computer could call on its birthday frame, and use the information contained in the frame, to apply to the situation. The computer knows that there is usually cake and presents because of the information contained in the knowledge frame. Frames can also overlap, or contain sub-frames.

BRANCHES OF ARTIFICIAL INTELLIGENCE

Logical

The programme decides what to do by inferring that certain actions are appropriate for achieving its goals. The first article proposing this was [McC59]. [McC89] is a more recent summary.lists some of the concepts involved in logical aI. is an important text.

Search

AI programmes often examine large numbers of possibilities, *e.g*. moves in a chess game or inferences by a theorem proving programme. Discoveries are continually made about how to do this more efficiently in various domains.When a programme makes observations of some kind, it is often programmed to compare what it sees with a pattern. For example, a vision programme may try to match a pattern of eyes and a nose in a scene in order to find a face. More complex patterns, *e.g*. in a natural language text, in a chess position, or in the history of some event are also studied.Facts about the world have to be represented in some way. Usually languages of mathematical logic are used.

Inference

Mathematical logical deduction is adequate for some purposes, but new methods of *non-monotonic* inference have been added to logic since the 1970s. The simplest kind of non-monotonic reasoning is default reasoning in which a conclusion is to be inferred by default, but the conclusion can be withdrawn if there is evidence to the contrary.

For example, when we hear of a bird, we man infer that it can fly, but this conclusion can be reversed when we hear that it is a penguin. It is the possibility that a conclusion may have to be withdrawn that constitutes the non-monotonic character of the reasoning. Ordinary logical reasoning is monotonic in that the set of conclusions that can the drawn from a set of premises is a monotonic increasing function of the premises. Circumscription is another form of non-monotonic reasoning.

Knowledge and Reasoning

This is the area in which AI is farthest from human-level, in spite of the fact that it has been an active research area since the 1950s. While there has been considerable progress, *e.g*. in developing systems of *non-monotonic reasoning* and theories of action, yet more new ideas are needed. The Cyc system contains a large but spotty collection of common sense facts.

The approaches to AI based on *connectionism* and *neural nets* specialize in that. There is also learning of laws expressed in logic. [Mit97] is a comprehensive undergraduate text on machine learning. Programmes can only learn what facts or behaviours their formalisms can represent, and unfortunately learning systems are almost all based on very limited abilities to represent information.

Planning programmes start with general facts about the world (especially facts about the effects of actions), facts about the particular situation and a statement of a goal. From these, they generate a strategy for achieving the goal. In the most common cases, the strategy is just a sequence of actions.

Ontology

Ontology is the study of the kinds of things that exist. In AI, the programmes and sentences deal with various kinds of objects, and we study what these kinds are and what their basic properties are. Emphasis on ontology begins in the 1990s.

Heuristics

A heuristic is a way of trying to discover something or an idea imbedded in a programme. The term is used variously in AI. *Heuristic functions* are used in some approaches to search to measure how far a node in a search tree seems to be from a goal. *Heuristic predicates* that compare two nodes in a search tree to see if one is better than the other, *i.e*. constitutes an advance towards the goal, may be more useful..

Genetic Programming

Genetic programming is a technique for getting programmes to solve a task by mating random Lisp programmes and selecting fittest in millions of generations.

OUTLINE AND HISTORY IN ARTIFICIAL INTELLIGENCE

Artificial Intelligence (AI) is the name that has been given to the field of computer science devoted to making machines do things that would require intelligence if done by humans (to paraphrase Minsky). Some are even ambitious enough to say that the goal of AI is to create minds in machines. But it has proven exceedingly difficult to even define words like "intelligence" and "mind", much less achieve computer implementation. The results of the 40-year history of AI have dampened the spirits of legions of would-be enthusiasts. The most "glamorous" contemporary computer applications are GUI or Java programmes, not AI. Artificial Intelligence has been divided into those concerned with processing of symbolic information and those concerned with Neural Networks. But the Symbolic AI camp has made every effort to exclude Neural Networks from the definition of AI, and the field of "Connectionism" is increasingly regarded as a somewhat independent discipline. Marvin Minsky has said that Neural Networks are "too stupid" to be considered Artificial Intelligence. Symbolic AI itself is divided into two subgroups, one primarily concerned with logic, and the other concerned with heuristics ("rule-of-thumb" methods).

Artificial Intelligence as a field dates from the summer of 1956 when a conference on the subject was organized by John McCarthy at Dartmouth College in New Hampshire. In attendance were the four "founding fathers" of AI: John McCarthy, Marvin Minsky, Herbert Simon and Allen Newell. Herbert Simon was a political scientist who was interested in the way decisions are made in bureaucracies. He found that "management manuals" are often used by organizations to provide solutions to problems — and he became interested in the compilation and application of the problem-solving procedures of these manuals — a "heuristic" approach.

Simon teamed-up with mathematician Allen Newell to create the world's first AI programme: "Logic Theorist". Combining logic with "search tree" heuristics, Logic Theorist proved 38 of the first 52 theorems of Russell and Whitehead's *Principal Mathematica*. The proof for one of these theorems was even more elegant that the one given in *Principal*. At the Dartmouth conference, Simon and Newell distinguished themselves by being the only participants who already had a working AI programme.

John McCarthy induced the conference participants to support the term "Artificial Intelligence" to describe their discipline. Discussion centered on the idea that intelligence is based on internal representations of

information, corresponding symbols, and the processing of information and symbols. In this view, intelligence transcends any specific hardware (or "wetware", as brains are often called).

McCarthy believed that the key to AI was being "able to write out the rules that would let a computer think the way we do". He was one of the foremost proponents of the "logic" school, and he attempted to implement machine intelligence through predicate calculus. McCarthy is also the creator of LISP (LISt Processing) language, the most widely-used computer language in AI research in North America. LISP is intended to reproduce what is regarded as the associative features of thought. LISP is based on Alonzo Church's "Lambda Calculus", which can apply functions to functions as readily as it can apply functions to numbers or characters. LISP programming is notoriously recursive (ie, using functions that call themselves).

Marvin Minsky is probably the most famous of the four "founding fathers" of AI. He is particularly well-known among cryonicists because he sponsored Eric Drexler's PhD thesis and wrote the forward to *Engines of Creation*. Minsky cofounded the MIT AI Lab with McCarthy in 1958. McCarthy left in 1962 to start AI research at Stanford, leaving Minsky to "rule the roost" in AI at MIT for many, many years.

In the early days of computing science there were, on the one side, a small group of enthusiasts with unbounded visions of what computers could do. To many, the automating of all mental activities — and ultimately the automation of mind — seemed only a few years away. On the other side was the general public, who (in pre-PC times) had a gross ignorance and no small amount of fear about the capabilities of computers. Managers, professionals and others feared for their jobs. IBM researchers were developing programmes which could play chess, play checkers and prove geometry theorems, but the effect of this work on marketing was not seen to be favourable.

IBM dropped its game-playing and theorem-proving research and adopted the posture that computers can not steal jobs because they are only moronic devices for number-crunching, data-manipulation and word-processing. IBM's pragmatic marketing position has proven prophetic in light of widespread PC use and what many people believe to be forty years of failed attempts to achieve Artificial Intelligence.

Several attempt have been made to create programme "agents" in "Micro Worlds", where simple language could be applied to a limited field

of discourse. One of the most ambitious of such projects was a system by Terry Winograd that represented a robot named SHRDLU. The robot would manipulate coloured blocks and pyramids in response to English-like commands, and could give simple explanations when quizzed about its actions (like how to build a steeple). The programme consisted of many fragments that acted as independent agents. This approach is more "heuristic" than logical, and it typifies Minsky's approach to AI.

In theory, a "Micro World" could be expanded continuously until the robot's "universe of discourse" overlapped the human universe of discourse — and the robot was a genuinely sentient being. In practice, scaling-up a "Micro World" has proven exceedingly difficult.

Minsky directed such a project at MIT which got bigger and bigger until there were so many programmes which had been written by so many people — and the system was so large — that no one understood it. It was abandoned in 1971. Later, Minsky wrote *The Society Of Mind* in which he attempted to decompose the human mind into a "society" of independent "agents", none of which could be called intelligent. Minsky's thesis is that this is the way that real minds work — and it is the way that machine minds will be built.

In the early 1970s the language PROLOG (PROgramming in LOGic) became the favoured AI language outside the United States (LISP was too firmly established in the US). PROLOG is a "higher-level" language that allows the programmer to specify what is to be done, rather than how, due to PROLOG's mechanisms for manipulating logical statements.

The late 1970s and early 1980s witnessed a huge surge in corporate interest and capital into the area of AI known as "expert systems". These systems consist of a knowledge base of "if-then" rules, accumulated from human experts. A car diagnosis system, for example, might contain the rule "If the engine won't turn-over, thencheck the battery strength". And "If the engine won't turn-over and the battery strength is high, then check the spark plugs". Expert systems incorporated the value of knowledge as well as logic in problem-solving, and had the promise of being as useful as human experts.

The first expert system was DENDRAL, a system for narrowing-down the possible chemical structure of a compound based on formula, spectral information and the encoded wisdom of chemists. DENDRAL proved its success by deducing the structure of Di-n-decyl $C_{20}H_{22}$ from 11 million possible combinations.

But as the system accumulated more knowledge it became too difficult to expand and maintain. MYCIN was a better-designed expert system that more cleanly separated the rules (knowledge-base) from the logic used to apply the rules. MYCIN diagnosed infectious blood diseases on the basis of blood tests, bacterial cultures and other information. Other expert systems were developed which proved to be of benefit for certain specialized applications.

But knowledge-engineering — transferring human expert knowledge to expert computer systems — proved to be far more difficult than anyone anticipated. The systems required continual revision to avoid obsolescence. And too often they made gross mistakes that no one with "common sense" would make. The expert system "boom" was followed by an expert system "bust".

Computer systems designed to translate from one human language to another have proven exceedingly difficult to build, for the same reason that computer systems built to understand human language have foundered. Word meanings are context-dependent. The phrase "That is a big drop" has a different meaning in an optometrist's office than it has on the edge of a cliff.

A sentence like "Still waters run deep" is even more challenging, not simply because it is metaphorical, but because every word in the sentence can:

- Have multiple meanings and
- Be used in more than one part-of-speech. Semantics governs syntax, which can make parsing sentences unfathomable for computer programmes.

Based on the assumption that common-sense understanding necessarily must be built upon a large knowledge-base, Doug Lenat has begun a $25 million project to create a computer-system with an enormous amounts of knowledge. Cyc (short for encyclopedia) collects knowledge in "frames" (collections of facts and rules) in a feeding process that is expected to take two person-centuries, and include 100 million frames.

Cyc is capable of "meditating" on its knowledge, searching for analogies. Lenat claims that Cyc will eventually be able to educate itself (by reading and having discussions with "tutors") rather than having to be "spoon fed" its knowledge. But McCarthy thinks Cyc has inadequate logic, and Minsky doesn't think Cyc has a wide-enough variety of procedures to do much with its information. The Artificial Life community has expanded on the "Micro World" concept by creating Micro Worlds in

which the artificial creatures experience "pleasure and pain", have needs and must adapt. In these "Micro Worlds" there is not only a universe of discourse, but a universe of values (motives, "feelings"). In S.W. Wilson's Animat system, the creatures adapt by learning what rules "work" and what rules don't. Genetic algorithms provide a basis for "learning".

AI "founding father" Allen Newell has designed the SOAR system, a collection of psychological mechanisms based on his book *Unified Theories Of Cognition*. SOAR uses representations and ends-means analysis for problem-solving. A "problem" is, by definition, a discrepancy between an existing condition and a goal condition. As with the Artificial Life systems, SOAR presumes that an intelligent system must be "motivated" by goal-states — and the sophisticated heuristic search strategies of ends-means analysis are used to achieve those states. "Learning" occurs in the process.

ARTIFICIAL INTELLIGENCE AND HUMAN MINDS

In 1950 Alan Turing wrote an article for the journal MIND in which he suggested a test wherein an interrogator would submit teletyped questions to both a machine and a person — and receive teletyped answers from both. Turing claimed that if the interrogator could not correctly distinguish between the person and the machine, the machine deserved to be called intelligent. Turing believed that by the year 2000, the average interrogator would be unable to determine the human more than 70% of the time in less than 5 minutes. Despite the flaws in Turing's approach, the "Turing Test" became a focal point of discussions about machine intelligence. It avoids the necessity of definingintelligence, and relies on appearances to an "average" interrogator — under a time limit. And it places a great emphasis on a machine being able to resemble a human. In a conscious effort to expose the irrelevance of deceptive appearances, Joseph Weizenbaum wrote a programme in the mid-1960s called ELIZA, which simulates a nondirective psychotherapist. If a user would type-in "My mother is afraid of cats", the programme might respond with the phrase "Tell me more about your family". To the phrase "He works with computers" it might respond "Do machines frighten you?".

If the user entered a sentence that triggered no pre-defined answer, it might respond with "Go on" or "Earlier you talked of your mother". Ironically, Weizenbaum made every effort to make ELIZA convincing, but was irritated to learn that there were people who seriously interacted with ELIZA by confiding their problems. Weizenbaum believed that human

intuition and wisdom are not "machinable", and he even objected to the goals of AI on moral grounds.

In 1980 philosopher John Searle wrote a paper describing a thought experiment of a person passing the Turing Test in Chinese. In this scenario, an English-speaking person who is ignorant of Chinese would sit in a room in which he/she would receive messages written in Chinese. Detailed scripts would describe what responses to provide. Even though the person in the "Chinese Room" might convince the Chinese interrogator that someone in the room understood Chinese, all that occurred in reality was symbol manipulation. Searle claimed that this is all computers ever do or can do — manipulate symbols without any real understanding of what those symbols mean.

Searle believes that computers will eventually be able to perform every intellectual feat of which humans are capable — and pass every Turing Test with ease — yet still be lacking in subjective consciousness. He believes that 2 different "systems", one conscious and the other unconscious, could produce identical behaviour. Searle's position has been called weak AI, and it is contrasted with strong AI: the idea that intelligent machines will eventually possess consciousness, self-awareness and emotion. (Roger Penrose denies the possibility of both strong AI and weak AI.) Searle characterizes the "strong AI" position as believing that "the mind" is "just a computer programme". Others have characterized the difference between "weak AI" and "strong AI" as the difference between a "third-person" approach to consciousness and a "first-person" approach.

Before computers, many people undoubtedly believed that arithmetic operations or the symbolic manipulations of integral calculus required intelligence. Some people still worry about the fact that computer chess-playing machines may soon be able to defeat all human opponents. But the limitations on this capability are nothing more than those of hardware to exhaustively search the consequences of millions of board positions in a short time. Is the ability of a computer to defeat any human opponent at chess any more a sign of intelligence than the ability to calculate the square root of 2 to hundreds of decimal places in less than a millisecond?

The Church-Turing Thesis essentially asserts that any algorithm can be implemented on a computer. If we can introspectively reduce any intellectual feat to a step-by-step procedure (an algorithm), then a computer can implement it. Yet computers have had the most difficulty with things that humans do without conscious thought, such as recognizing faces.

Much problem-solving actually involves pattern-recognition — intuitive associations of problems with similar problems. The difficulty of experts to reduce this ability to algorithms has been one of the obstacles to implementing expert systems. Yet neural networks may eventually be capable of handling some of these tasks.

Much of what passes for creativity is simply pseudo-random association. Douglas Hofstadter wrote a programme that uses Recursive Transition Network grammar and words randomly selected from word-categories to produce surrealistic prose: "A male pencil who must laugh clumsily would quack". My programme produces phrases like: "Your eye intrigues upon delight" (my word-lists are different). A careful analysis of so-called creative thought often reveals procedures like "find extreme examples" or "invert the situation". The presumption that computers "can only do what you tell them to do" seems contradicted by the fact that game-playing programmes frequently can defeat their creators.

Deciding whether a computer programme possesses intelligence should require a clear definition of the term "intelligence" and a means to measure it. An intelligent system should be able to form internal representations of the external world, be able to learn and be able to reason ("think") about the world and itself.

So-called "IQ tests" exist for human beings, but people are reluctant to apply the same standards to machines. Descriptions of what people would call intelligence might include "common sense" plus a general ability to analyse situations and to solve problems.

Often humans are called intelligent even though their abilities exist in specialized areas — and they are not seen as having "common sense". But the same allowances are not made for computers. Within humanity there is a vast range of intellectual abilities between infants and adults — or between the genius and the mentally-retarded or brain-damaged. I suspect that even so-called "third-person" criteria for intelligence covertly demand signs of subjective consciousness.

What are the signs of consciousness? Searle argued that his Chinese Room example demonstrated that it is possible to effectively manipulate symbols without understanding their meaning. Do awareness, understanding and meaningfulness require subjective consciousness? Does a spider know how a web is created? Does a beaver consciously build a dam? A frog may be aware of its surroundings, but is it conscious or intelligent? Self-awareness in the presence of a mirror has been

demonstrated by chimpanzees, but not by baboons. Yet other animals may possess some form of self-awareness insofar as carnivores do not attempt to eat themselves (although this may be just a response to pain).

Using self-awareness as a criterion for intelligence may not be more defensible than "common sense". Minsky has defined self-awareness as the ability to monitor one's own processes. He believes that computers could easily monitor their own processes and explain their actions more effectively than human beings.

Pattern-recognition, purpose and emotion are said to be attributes of intelligence or consciousness which cannot be reduced to algorithms. Nonetheless, neural networks can achieve pattern recognition. And computers can have a purpose or goal — to win a game of chess, for example.

It might be objected that the goals of computers are only those that have been programmed — but are not the goals of humans and animals those which have been programmed into their genes (pleasures, pains, fears, etc.)?

Certain people with neurological deficits have been incapable of feeling pain. Must this reduce their consciousness or intelligence? And what is emotion? Hans Moravec says that emotions are just drives for channeling behaviour — they focus energy around goals. He imagines robots that would get a "thrill" out of pleasing their owners — and exhibit "love".

The robots could become upset when their batteries run low and would plead with their owners (or express anger) for a recharge. Does emotion require biological organs? Are other forms of feeling possible with other hardware, perhaps even silicon?

The undue emphasis on "humanness" of the Turing Test is a severe limitation as a criterion for either intelligence or consciousness. Dolphins have a larger cerebral cortex than human beings, and appear to communicate with each other. If they do possess intelligence, or consciousness, it is alien to that of a human. Aside from the fact that they don't manipulate their environment with tools, their perceptive apparatus and natural environment is somewhat different:they place more emphasis on sound (echolation) and less on vision. Bats rely on echolation even more than dolphins, and a bat with an expanded brain would doubtless have a very different understanding of the world from a human.

An even more extreme example would be an organism with an expanded brain that relies primarily on taste and smell to perceive the

world. But the alienness of machine intelligence, consciousness and feeling could be even more extreme. Airplanes are not built to flap their wings, but they fly more effectively than birds.

The intelligence, consciousness and feelings of machines might be too remote from our own experience for us to identify with them. If it is true that a computer can only manipulate symbols without any real understanding, how do human beings achieve understanding? Books are full of knowledge, but are not conscious or intelligent.

Computers don't simply manipulate information, they do so in a goal-directed, problem-solving manner. If knowledge is used to solve problems, isn't that an indication of knowledge being meaningful? Symbols can only be meaningfully manipulated if they are meaningfully representative of something real.

Neural networks are said to recognize patterns because they give distinctive outputs corresponding to distinctive input patterns. A computer that does integral calculus gives a distinct solution for a distinct input equation. Can an operational definition be given to "understanding", or is it to be defined only relative to a subjective consciousness? What "magic" is performed by a biological brain that cannot — in theory — be performed by a computer? Why should it be necessary to build machines from organic materials to achieve consciousness?

Symbolic AI adherents have been harshly critical of the idea that intelligence can "emerge" from a neural network. But the Cyc project's attempt to accumulate a large enough knowledge base for a computer to acquire "common sense" seems itself like it is based on "emergence". A dictionary in itself is meaningless to a person because all of the words are defined in terms of other words.

A human being associates words with objects in the environment that themselves acquire meaning through their ability to induce an emotional/physiological response (like pleasure or pain). The meaning of a kiss involves both the physical features of the kiss and emotional reactions associated with who is kissing and being kissed.

Ultimately, meaning seems like another word for "qualia" — but there is also the question of context. Words have meaning not only by virtue of their association with objects, but by association with other words — and some words function only to link words. Is "meaning" an emergent phenomenon — the product of a "critical mass" of knowledge (or of something else?)?

If emergence occurs, there can be degrees of meaning, both for individual words and for the entire context of understanding. As with the way the phenomenon of heat "emerges" from molecular motion — or a traffic jam emerges for an increasing number of vehicles.

Marvin Minsky believes that a mind can be created only by analysing "mind" into its component parts ("agents"), none of which can be intelligent themselves. To quote Minsky: "Unless we can explain the mind in terms of things that have no thoughts or feelings of their own, we'll only have gone around in a circle." But Minsky's conception of the mind as a "society" of unintelligent agents still sounds suspiciously like an "emergent" phenomenon.

I say this to emphasize the "magicalness", although I don't deny that emergent phenomena exist. Connectionists seem to assume that intelligence or consciousness will emerge from a machine that simulates brain processes well-enough (reverse engineering), while Symbolists seem to assume that a machine will become conscious or intelligent if it is given enough knowledge or problem-solving capacity. If a mind cannot be created from increasingly better simulations of mind or brain, why can't it?

In the case of subjective conscious there seems to be the serious problem of objective proof. The only subjective reality we can know about directly is our own. We conclude that other people have feelings because we can identify with their behaviour. A computer need only be a good actor to pass the Turing Test — but then, perhaps, the same can be said of other people. Why should it necessarily be true that all humans have subjective consciousness?

There is simply no way to objectively prove that a machine, an animal or even another human being has subjective consciousness. The question is unfalsifiable and therefore does not even qualify as a scientific question. More seriously, how could an Uploader feel confident that a change from wetware to hardware would not involve a loss of subjective consciousness?

If it is not possible to distinguish a system having consciousness from one that does not when the behaviour of the two systems is identical, how could an Uploader be confident of not losing "self" in the process? A hardware environment may appear to provide a cozy new home for the self — but after the Uploading, all that might be left of the person would be a simulation of the person.

Even for those who do not choose to do a full upload, there could be increasing pressures to augment one's intelligence with computer-chip

add-ons to our biological brains. Those who do not do this will be "left behind" by those who do. Would these add-ons augment the "self", or is there a danger of a loss of identity/subjective consciousness? Could the testimony of others be trusted? Most seriously, how can cryonicists be sure that reconstruction by nanotechnology — or even reconstruction from a perfectly vitrified state — would not result in a simulation of ourselves (or even a very close clone) rather than a reestablishment of our subjective selves? All of our carefully-kept introspective diaries may be little more than additional information on how to produce a more convincing simulation.

Even if there can only be subjective verification of subjective states, those states can only be the product of objective material and objective processes. We are, individually (though not "scientifically"), in a position to observe both objective phenomena and subjective phenomena — and to correlate the two. Even now we can correlate PET scans with subjective processes. We may eventually learn to correlate our subjective experience with the anatomical basis of mind — and this could be the key to our survival.

ACTING HUMANLY: THE TURING TEST APPROACH

The Turing Test, proposed by Alan Turing (Turing, 1950), was designed to provide a satisfactory operational definition of intelligence. Turing defined intelligent behavior as the ability to achieve human-level performance in all cognitive tasks, sufficient to fool an interrogator. Roughly speaking, the test he proposed is that the computer should be interrogated by a human via a teletype, and passes the test if the interrogator cannot tell if there is a computer or a human at the other end. For now, programming a computer to pass the test provides plenty to work on. The computer would need to possess the following capabilities:

- natural language processing to enable it to communicate successfully in English (or some other human language);
- knowledge representation to store information provided before or during the interrogation;
- automated reasoning to use the stored information to answer questions and to draw new conclusions;
- machine learning to adapt to new circumstances and to detect and extrapolate patterns.

Turing's test deliberately avoided direct physical interaction between the interrogator and the computer, because *physical* simulation of a person is unnecessary for intelligence. However, the so-called total Turing Testincludes a video signal so that the interrogator can test the subject's perceptual abilities, as well as the opportunity for the interrogator to pass physical objects "through the hatch." To pass the total Turing Test, the computer will need

- computer vision to perceive objects, and
- robotics to move them about.

Within AI, there has not been a big effort to try to pass the Turing test. The issue of acting like a human comes up primarily when AI programs have to interact with people, as when an expert system explains how it came to its diagnosis, or a natural language processing system has a dialogue with a user. These programs must behave according to certain normal conventions of human interaction in order to make themselves understood. The underlying representation and reasoning in such a system may or may not be based on a human model.

Thinking humanly: The cognitive modelling approach

If we are going to say that a given program thinks like a human, we must have some way of determining how humans think. We need to get *inside* the actual workings of human minds. There are two ways to do this: through introspection—trying to catch our own thoughts as they go by—or through psychological experiments. Once we have a sufficiently precise theory of the mind, it becomes possible to express the theory as a computer program. If the program's input/output and timing behavior matches human behavior, that is evidence that some of the program's mechanisms may also be operating in humans.

For example, Newell and Simon, who developed GPS, the "General Problem Solver" (Newell and Simon, 1961), were not content to have their program correctly solve problems. They were more concerned with comparing the trace of its reasoning steps to traces of human subjects solving the same problems. This is in contrast to other researchers of the same time (such as Wang (1960)), who were concerned with getting the right answers regardless of how humans might do it. The interdisciplinary field of cognitive science brings together computer models from AI and experimental techniques from psychology to try to construct precise and testable theories of the workings of the human mind. Although cognitive

science is a fascinating field in itself, we are not going to be discussing it all that much in this book. We will occasionally comment on similarities or differences between AI techniques and human cognition. Real cognitive science, however, is necessarily based on experimental investigation of actual humans or animals, and we assume that the reader only has access to a computer for experimentation. We will simply note that AI and cognitive science continue to fertilize each other, especially in the areas of vision, natural language, and learning.

Thinking rationally: The laws of thought approach

The Greek philosopher Aristotle was one of the first to attempt to codify "right thinking," that is, irrefutable reasoning processes. His famous syllogisms provided patterns for argument structures that always gave correct conclusions given correct premises. For example, "Socrates is a man; all men are mortal; therefore Socrates is mortal." These laws of thought were supposed to govern the operation of the mind, and initiated the field of logic.

The development of formal logic in the late nineteenth and early twentieth centuries provided a precise notation for statements about all kinds of things in the world and the relations between them. (Contrast this with ordinary arithmetic notation, which provides mainly for equality and inequality statements about numbers.) By 1965, programs existed that could, given enough time and memory, take a description of a problem in logical notation and find the solution to the problem, if one exists. (If there is no solution, the program might never stop looking for it.) The so-called logicist tradition within artificial intelligence hopes to build on such programs to create intelligent systems.

There are two main obstacles to this approach. First, it is not easy to take informal knowledge and state it in the formal terms required by logical notation, particularly when the knowledge is less than 100% certain. Second, there is a big difference between being able to solve a problem "in principle" and doing so in practice. Even problems with just a few dozen facts can exhaust the computational resources of any computer unless it has some guidance as to which reasoning steps to try first. Although both of these obstacles apply to *any* attempt to build computational reasoning systems, they appeared first in the logicist tradition because the power of the representation and reasoning systems are well-defined and fairly well understood.

Acting rationally: The rational agent approach

Acting rationally means acting so as to achieve one's goals, given one's beliefs. An agent is just something that perceives and acts. (This may be an unusual use of the word, but you will get used to it.) In this approach, AI is viewed as the study and construction of rational agents.

In the "laws of thought" approach to AI, the whole emphasis was on correct inferences. Making correct inferences is sometimes *part* of being a rational agent, because one way to act rationally is to reason logically to the conclusion that a given action will achieve one's goals, and then to act on that conclusion. On the other hand, correct inference is not *all* of rationality, because there are often situations where there is no provably correct thing to do, yet something must still be done. There are also ways of acting rationally that cannot be reasonably said to involve inference. For example, pulling one's hand off of a hot stove is a reflex action that is more successful than a slower action taken after careful deliberation.

All the "cognitive skills" needed for the Turing Test are there to allow rational actions. Thus, we need the ability to represent knowledge and reason with it because this enables us to reach good decisions in a wide variety of situations. We need to be able to generate comprehensible sentences in natural language because saying those sentences helps us get by in a complex society. We need learning not just for erudition, but because having a better idea of how the world works enables us to generate more effective strategies for dealing with it. We need visual perception not just because seeing is fun, but in order to get a better idea of what an action might achieve—for example, being able to see a tasty morsel helps one to move toward it.

The study of AI as rational agent design therefore has two advantages. First, it is more general than the "laws of thought" approach, because correct inference is only a useful mechanism for achieving rationality, and not a necessary one. Second, it is more amenable to scientific development than approaches based on human behavior or human thought, because the standard of rationality is clearly defined and completely general. Human behavior, on the other hand, is well-adapted for one specific environment and is the product, in part, of a complicated and largely unknown evolutionary process that still may be far from achieving perfection. *This book will therefore concentrate on general principles of rational agents, and on components for constructing them.* We will see that despite the apparent simplicity with which the problem can be stated, an enormous variety of

issues come up when we try to solve it. One important point to keep in mind: we will see before too long that achieving perfect rationality—always doing the right thing—is not possible in complicated environments. The computational demands are just too high. However, for most of the book, we will adopt the working hypothesis that understanding perfect decision making is a good place to start. It simplifies the problem and provides the appropriate setting for most of the foundational material in the field.

The State of the Art

International grandmaster Arnold Denker studies the pieces on the board in front of him. He realizes there is no hope; he must resign the game. His opponent, Hitech, becomes the first computer program to defeat a grandmaster in a game of chess.

"I want to go from Boston to San Francisco," the traveller says into the microphone. "What date will you be travelling on?" is the reply. The traveller explains she wants to go October 20th, nonstop, on the cheapest available fare, returning on Sunday. A speech understanding program named Pegasus handles the whole transaction, which results in a confirmed reservation that saves the traveller $894 over the regular coach fare. Even though the speech recognizer gets one out of ten words wrong, it is able to recover from these errors because of its understanding of how dialogs are put together.

An analyst in the Mission Operations room of the Jet Propulsion Laboratory suddenly starts paying attention. A red message has flashed onto the screen indicating an "anomaly" with the Voyager spacecraft, which is somewhere in the vicinity of Neptune. Fortunately, the analyst is able to correct the problem from the ground. Operations personnel believe the problem might have been overlooked had it not been for Marvel, a real-time expert system that monitors the massive stream of data transmitted by the spacecraft, handling routine tasks and alerting the analysts to more serious problems. Cruising the highway outside of Pittsburgh at a comfortable 55 mph, the man in the driver's seat seems relaxed. He should be—for the past 90 miles, he has not had to touch the steering wheel. The real driver is a robotic system that gathers input from video cameras, sonar, and laser range finders attached to the van. It combines these inputs with experience learned from training runs and succesfully computes how to steer the vehicle.

A leading expert on lymph-node pathology describes a fiendishly difficult case to the expert system, and examines the system's diagnosis. He scoffs at the system's response. Only slightly worried, the creators of the system suggest he ask the computer for an explanation of the diagnosis. The machine points out the major factors influencing its decision, and explains the subtle interaction of several of the symptoms in this case. The expert admits his error, eventually.

From a camera perched on a street light above the crossroads, the traffic monitor watches the scene. If any humans were awake to read the main screen, they would see "Citroen 2CV turning from Place de la Concorde into Champs Elysees," "Large truck of unknown make stopped on Place de la Concorde," and so on into the night. And occasionally, "Major incident on Place de la Concorde, speeding van collided with motorcyclist," and an automatic call to the emergency services.

These are just a few examples of artificial intelligence systems that exist today. Not magic or science fiction—but rather science, engineering, and mathematics, to which this book provides an introduction.

ARTIFICIAL NEURAL NETWORK

An artificial neural network is a system based on the operation of biological neural networks, in other words, is an emulation of biological neural system. Why would be necessary the implementation of artificial neural networks?

Although computing these days is truly advanced, there are certain tasks that a programme made for a common microprocessor is unable to perform; even so a software implementation of a neural network can be made with their advantages and disadvantages.

Qualities

- A neural network can perform tasks that a linear programme can not.
- When an element of the neural network fails, it can continue without any problem by their parallel nature.
- A neural network learns and does not need to be reprogrammed.
- It can be implemented in any application.
- It can be implemented without any problem.

Drawbacks

- The neural network needs training to operate.
- The architecture of a neural network is different from the architecture of microprocessors therefore needs to be emulated.
- Requires high processing time for large neural networks.

Another aspect of the artificial neural networks is that there are different architectures, which consequently requires different types of algorithms, but despite to be an apparently complex system, a neural network is relatively simple.Artificial neural networks (ANN) are among the newest signal-processing technologies in the engineer's toolbox.

The field is highly interdisciplinary, but our approach will restrict the view to the engineering perspective. In engineering, neural networks serve two important functions: as pattern classifiers and as nonlinear adaptive filters.

We will provide a brief overview of the theory, learning rules, and applications of the most important neural network models. Definitions and Style of Computation An Artificial Neural Network is an adaptive, most often nonlinear system that learns to perform a function (an input/ output map) from data. Adaptive means that the system parametres are changed during operation, normally called the training phase.

After the training phase the Artificial Neural Network parametres are fixed and the system is deployed to solve the problem at hand (the testing phase). The Artificial Neural Network is built with a systematic step-by-step procedure to optimize a performance criterion or to follow some implicit internal constraint, which is commonly referred to as the learning rule.

The input/output training data are fundamental in neural network technology, because they convey the necessary information to 'discover' the optimal operating point. The nonlinear nature of the neural network processing elements (PEs) provides the system with lots of flexibility to achieve practically any desired input/output map, *i.e.*, some Artificial Neural Networks are universal mappers. There is a style in neural computation that is worth describing. An input is presented to the neural network and a corresponding desired or target response set at the output (when this is the case the training is called supervised). An error is composed from the difference between the desired response and the system output.

This error information is fed back to the system and adjusts the system

parametres in a systematic fashion (the learning rule). The process is repeated until the performance is acceptable. It is clear from this description that the performance hinges heavily on the data.

If one does not have data that cover a significant portion of the operating conditions or if they are noisy, then neural network technology is probably not the right solution. On the other hand, if there is plenty of data and the problem is poorly understood to derive an approximate model, then neural network technology is a good choice.

This operating procedure should be contrasted with the traditional engineering design, made of exhaustive subsystem specifications and intercommunication protocols. In artificial neural networks, the designer chooses the network topology, the performance function, the learning rule, and the criterion to stop the training phase, but the system automatically adjusts the parametres.

So, it is difficult to bring a priori information into the design, and when the system does not work properly it is also hard to incrementally refine the solution. But ANN-based solutions are extremely efficient in terms of development time and resources, and in many difficult problems artificial neural networks provide performance that is difficult to match with other technologies. Denker 10 years ago said that 'artificial neural networks are the second best way to implement a solution' motivated by the simplicity of their design and because of their universality, only shadowed by the traditional design obtained by studying the physics of the problem.

At present, artificial neural networks are emerging as the technology of choice for many applications, such as pattern recognition, prediction, system identification, and control.

Biological Model

Artificial neural networks emerged after the introduction of simplified neurons by McCulloch and Pitts in 1943. These neurons were presented as models of biological neurons and as conceptual components for circuits that could perform computational tasks. The basic model of the neuron is founded upon the functionality of a biological neuron. 'Neurons are the basic signaling units of the nervous system' and 'each neuron is a discrete cell whose several processes arise from its cell body'.

The neuron has four main regions to its structure. The cell body, or soma, has two offshoots from it, the dendrites, and the axon, which end

in presynaptic terminals. The cell body is the heart of the cell, containing the nucleus and maintaining protein synthesis. A neuron may have many dendrites, which branch out in a treelike structure, and receive signals from other neurons.

A neuron usually only has one axon which grows out from a part of the cell body called the axon hillock. The axon conducts electric signals generated at the axon hillock down its length. These electric signals are called action potentials.

The other end of the axon may split into several branches, which end in a presynaptic terminal. Action potentials are the electric signals that neurons use to convey information to the brain. All these signals are identical.

Therefore, the brain determines what type of information is being received based on the path that the signal took. The brain analyses the patterns of signals being sent and from that information it can interpret the type of information being received. Myelin is the fatty tissue that surrounds and insulates the axon.

Often short axons do not need this insulation. There are uninsulated parts of the axon. These areas are called Nodes of Ranvier. At these nodes, the signal traveling down the axon is regenerated.

This ensures that the signal traveling down the axon travels fast and remains constant (*i.e.* very short propagation delay and no weakening of the signal). The synapse is the area of contact between two neurons. The neurons do not actually physically touch. They are separated by the synaptic cleft, and electric signals are sent through chemical 13 interaction. The neuron sending the signal is called the presynaptic cell and the neuron receiving the signal is called the postsynaptic cell.

The signals are generated by the membrane potential, which is based on the differences in concentration of sodium and potassium ions inside and outside the cell membrane. Neurons can be classified by their number of processes (or appendages), or by their function.

If they are classified by the number of processes, they fall into three categories. Unipolar neurons have a single process (dendrites and axon are located on the same stem), and are most common in invertebrates. In bipolar neurons, the dendrite and axon are the neuron's two separate processes. Bipolar neurons have a subclass called pseudo-bipolar neurons, which are used to send sensory information to the spinal cord. Finally,

multipolar neurons are most common in mammals. Examples of these neurons are spinal motor neurons, pyramidal cells and Purkinje cells (in the cerebellum). If classified by function, neurons again fall into three separate categories. The first group is sensory, or afferent, neurons, which provide information for perception and motor coordination.

The second group provides information (or instructions) to muscles and glands and is therefore called motor neurons. The last group, interneuronal, contains all other neurons and has two subclasses. One group called relay or projection interneurons have long axons and connect different parts of the brain. The other group called local interneurons are only used in local circuits.

Distributed Representation

An artifcial neural network consists of a pool of simple processing units which communicate by sending signals to each other over a large number of weighted connections. A set of major aspects of a parallel distributed model can be distinguished:

- A set of processing units ('neurons,' 'cells');
- A state of activation yk for every unit, which equivalent to the output of the unit;
- Connections between the units. Generally each connection is defined by a weight wjk which determines the effect which the signal of unit j has on unit k;
- A propagation rule, which determines the effective input sk of a unit from its external inputs;
- An activation function Fk, which determines the new level of activation based on the efective input sk(t) and the current activation yk(t) (*i.e.*, the update);
- An external input (aka bias, offset) øk for each unit;
- A method for information gathering (the learning rule);
- An environment within which the system must operate, providing input signals and | if necessary | error signals.

Processing Units

Each unit performs a relatively simple job: receive input from neighbours or external sources and use this to compute an output signal which is propagated to other units. Apart from this processing, a second

task is the adjustment of the weights. The system is inherently parallel in the sense that many units can carry out their computations at the same time. Within neural systems it is useful to distinguish three types of units: input units (indicated by an index i) which receive data from outside the neural network, output units (indicated by an index o) which send data out of the neural network, and hidden units (indicated by an index h) whose input and output signals remain within the neural network.

During operation, units can be updated either synchro-nously or asynchronously. With synchronous updating, all units update their activation simultaneously; with asynchronous updating, each unit has a (usually fixed) probability of updating its activation at a time t, and usually only one unit will be able to do this at a time. In some cases the latter model has some advantages.

3

Categories of Model-Based AI Agents

INTELLIGENT AGENT

In artificial intelligence, an intelligent agent (IA) is anything which perceives its environment, takes actions autonomously in order to achieve goals, and may improve its performance with learning or may use knowledge. They may be simple or complex — a thermostat is considered an example of an intelligent agent, as is a human being, as is any system that meets the definition, such as a firm, a state, or a biome.

Leading AI textbooks define "artificial intelligence" as the "study and design of intelligent agents", a definition that considers goal-directed behavior to be the essence of intelligence. Goal-directed agents are also described using a term borrowed from economics, "rational agent".

An agent has an "objective function" that encapsulates all the IA's goals. Such an agent is designed to create and execute whatever plan will, upon completion, maximize the expected value of the objective function. For example, a reinforcement learning agent has a "reward function" that allows the programmers to shape the IA's desired behavior, and an evolutionary algorithm's behavior is shaped by a "fitness function".

Intelligent agents in artificial intelligence are closely related to agents in economics, and versions of the intelligent agent paradigm are studied in cognitive science, ethics, the philosophy of practical reason, as well as in many interdisciplinary socio-cognitive modeling and computer social simulations.

Intelligent agents are often described schematically as an abstract functional system similar to a computer program. Abstract descriptions of

intelligent agents are called abstract intelligent agents (AIA) to distinguish them from their real world implementations. An autonomous intelligent agent is designed to function in the absence of human intervention. Intelligent agents are also closely related to software agents (an autonomous computer program that carries out tasks on behalf of users).

Definition of artificial intelligence

Computer science defines AI research as the study of intelligent agents. The leading AI textbook defines an "agent" as:

- "Anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators" defines a "rational agent" as:
- "An agent that acts so as to maximize the expected value of a performance measure based on past experience and knowledge." and defines the field of "artificial intelligence" research as:
- "The study and design of rational agents"

A similar definition of AI is given by Kaplan and Haenlein: "A system's ability to correctly interpret external data, to learn from such data, and to use those learnings to achieve specific goals and tasks through flexible adaptation."

Padgham & Winikoff (2005) agree that an intelligent agent is situated in an environment and responds in a timely (though not necessarily real-time) manner to environment changes. However, intelligent agents must also proactively pursue goals in a flexible and robust way. Optional desiderata include that the agent be rational, and that the agent be capable of belief-desire-intention analysis.

Advantages of this definition

Philosophically, it avoids several lines of criticism. Unlike the Turing test, it does not refer to human intelligence in any way. Thus there is no need to discuss if it is "real" vs "simulated" intelligence (i.e., "synthetic" vs "artificial" intelligence), and does not indicate that such a machine has a mind, consciousness or true understanding (i.e., it does not imply John Searle's "strong AI hypothesis"). It also doesn't attempt to draw a sharp dividing line between behaviors that are "intelligent" and behaviors that are "unintelligent" — programs need only be measured in terms of their objective function.

More importantly, it has a number of practical advantages that have

helped move AI research forward. It provides a reliable and scientific way to test programs; researchers can directly compare or even combine different approaches to isolated problems, by asking which agent is best at maximizing a given "goal function". It also gives them a common language to communicate with other fields — such as mathematical optimization (which is defined in terms of "goals") or economics (which uses the same definition of a "rational agent").

Objective function

An agent that is assigned an explicit "goal function" is considered more intelligent if it consistently takes actions that successfully maximize its programmed goal function. The goal can be simple ("1 if the IA wins a game of Go, 0 otherwise") or complex ("Perform actions mathematically similar to ones that succeeded in the past"). The "goal function" encapsulates all of the goals the agent is driven to act on; in the case of rational agents, the function also encapsulates the acceptable trade-offs between accomplishing conflicting goals. (Terminology varies; for example, some agents seek to maximize or minimize a "utility function", "objective function", or "loss function".)

Goals can be explicitly defined or induced. If the AI is programmed for "reinforcement learning", it has a "reward function" that encourages some types of behavior and punishes others. Alternatively, an evolutionary system can induce goals by using a "fitness function" to mutate and preferentially replicate high-scoring AI systems, similar to how animals evolved to innately desire certain goals such as finding food. Some AI systems, such as nearest-neighbor, instead of reason by analogy, these systems are not generally given goals, except to the degree that goals are implicit in their training data. Such systems can still be benchmarked if the non-goal system is framed as a system whose "goal" is to accomplish its narrow classification task.

Systems that are not traditionally considered agents, such as knowledge-representation systems, are sometimes subsumed into the paradigm by framing them as agents that have a goal of (for example) answering questions as accurately as possible; the concept of an "action" is here extended to encompass the "act" of giving an answer to a question. As an additional extension, mimicry-driven systems can be framed as agents who are optimizing a "goal function" based on how closely the IA succeeds in mimicking the desired behavior. In the generative adversarial networks of the 2010s, an "encoder"/"generator" component attempts to

mimic and improvise human text composition. The generator is attempting to maximize a function encapsulating how well it can fool an antagonistic "predictor"/"discriminator" component.

While GOFAI systems often accept an explicit goal function, the paradigm can also be applied to neural networks and to evolutionary computing. Reinforcement learning can generate intelligent agents that appear to act in ways intended to maximize a "reward function". Sometimes, rather than setting the reward function to be directly equal to the desired benchmark evaluation function, machine learning programmers will use reward shaping to initially give the machine rewards for incremental progress in learning. Yann LeCun stated in 2018 that "Most of the learning algorithms that people have come up with essentially consist of minimizing some objective function." AlphaZero chess had a simple objective function; each win counted as +1 point, and each loss counted as -1 point. An objective function for a self-driving car would have to be more complicated. Evolutionary computing can evolve intelligent agents that appear to act in ways intended to maximize a "fitness function" that influences how many descendants each agent is allowed to leave.

The theoretical and uncomputable AIXI design is a maximally intelligent agent in this paradigm; however, in the real world, the IA is constrained by finite time and hardware resources, and scientists compete to produce algorithms that can achieve progressively higher scores on benchmark tests with real-world hardware.

Classes of intelligent agents

Russel and Norvig's classification

Russell & Norvig (2003) group agents into five classes based on their degree of perceived intelligence and capability:

Simple reflex agents

Simple reflex agents act only on the basis of the current percept, ignoring the rest of the percept history. The agent function is based on the *condition-action rule*: "if condition, then action".

This agent function only succeeds when the environment is fully observable. Some reflex agents can also contain information on their current state which allows them to disregard conditions whose actuators are already triggered.

Infinite loops are often unavoidable for simple reflex agents operating in partially observable environments. Note: If the agent can randomize its actions, it may be possible to escape from infinite loops.

Model-based reflex agents

A model-based agent can handle partially observable environments. Its current state is stored inside the agent maintaining some kind of structure that describes the part of the world which cannot be seen. This knowledge about "how the world works" is called a model of the world, hence the name "model-based agent".

A model-based reflex agent should maintain some sort of internal model that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state. Percept history and impact of action on the environment can be determined by using the internal model. It then chooses an action in the same way as reflex agent.

An agent may also use models to describe and predict the behaviors of other agents in the environment.

Goal-based agents

Goal-based agents further expand on the capabilities of the model-based agents, by using "goal" information. Goal information describes situations that are desirable. This provides the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state. Search and planning are the subfields of artificial intelligence devoted to finding action sequences that achieve the agent's goals.

Utility-based agents

Goal-based agents only distinguish between goal states and non-goal states. It is also possible to define a measure of how desirable a particular state is. This measure can be obtained through the use of a *utility function* which maps a state to a measure of the utility of the state.

A more general performance measure should allow a comparison of different world states according to how well they satisfied the agent's goals. The term utility can be used to describe how "happy" the agent is.

A rational utility-based agent chooses the action that maximizes the expected utility of the action outcomes - that is, what the agent expects to derive, on average, given the probabilities and utilities of each outcome.

A utility-based agent has to model and keep track of its environment,

tasks that have involved a great deal of research on perception, representation, reasoning, and learning.

Learning agents

Learning has the advantage that it allows the agents to initially operate in unknown environments and to become more competent than its initial knowledge alone might allow. The most important distinction is between the "learning element", which is responsible for making improvements, and the "performance element", which is responsible for selecting external actions.

The learning element uses feedback from the "critic" on how the agent is doing and determines how the performance element, or "actor", should be modified to do better in the future. The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions.

The last component of the learning agent is the "problem generator". It is responsible for suggesting actions that will lead to new and informative experiences.

Weiss's classification

Weiss (2013) defines four classes of agents:

- Logic-based agents – in which the decision about what action to perform is made via logical deduction;
- Reactive agents – in which decision making is implemented in some form of direct mapping from situation to action;
- Belief-desire-intention agents – in which decision making depends upon the manipulation of data structures representing the beliefs, desires, and intentions of the agent; and finally,
- Layered architectures – in which decision making is realized via various software layers, each of which is more or less explicitly reasoning about the environment at different levels of abstraction.

Hierarchies of agents

To actively perform their functions, Intelligent Agents today are normally gathered in a hierarchical structure containing many "sub-agents". Intelligent sub-agents process and perform lower level functions. Taken together, the intelligent agent and sub-agents create a complete system that can accomplish difficult tasks or goals with behaviors and responses that display a form of intelligence.

Generally, an agent can be constructed by separating the body into the sensors and actuators, and so that it operates with a complex perception system that takes the description of the world as input for a controller and outputs commands to the actuator. However, a hierarchy of controller layers is often necessary to balance the immediate reaction desired for low-level tasks and the slow reasoning about complex, high-level goals.

INTELLIGENT AGENTS IN ARTIFICIAL INTELLIGENCE

An intelligent agent (IA) is an entity that makes a decision, that enables artificial intelligence to be put into action. It can also be described as a software entity that conducts operations in the place of users or programs after sensing the environment. It uses actuators to initiate action in that environment.

An Intelligent Agent (IA)

This agent has some level of autonomy that allows it to perform specific, predictable, and repetitive tasks for users or applications. It's also termed as 'intelligent' because of its ability to learn during the process of performing tasks. The two main functions of intelligent agents include perception and action. Perception is done through sensors while actions are initiated through actuators.

Intelligent agents consist of sub-agents that form a hierarchical structure. Lower-level tasks are performed by these sub-agents.

The higher-level agents and lower-level agents form a complete system that can solve difficult problems through intelligent behaviors or responses.

Characteristics of intelligent agents

Intelligent agents have the following distinguishing characteristics:

- They have some level of autonomy that allows them to perform certain tasks on their own.
- They have a learning ability that enables them to learn even as tasks are carried out.
- They can interact with other entities such as agents, humans, and systems.
- New rules can be accommodated by intelligent agents incrementally.
- They exhibit goal-oriented habits.
- They are knowledge-based. They use knowledge regarding communications, processes, and entities.

The structure of intelligent agents

The IA structure consists of three main parts: architecture, agent function, and agent program.

1. Architecture: This refers to machinery or devices that consists of actuators and sensors. The intelligent agent executes on this machinery. Examples include a personal computer, a car, or a camera.
2. Agent function: This is a function in which actions are mapped from a certain percept sequence. Percept sequence refers to a history of what the intelligent agent has perceived.
3. Agent program: This is an implementation or execution of the agent function. The agent function is produced through the agent program's execution on the physical architecture.

Categories of intelligent agents

There are 5 main categories of intelligent agents. The grouping of these agents is based on their capabilities and level of perceived intelligence.

Simple reflex agents

These agents perform actions using the current percept, rather than the percept history. The condition-action rule is used as the basis for the agent function. In this category, a fully observable environment is ideal for the success of the agent function.

Model-based reflex agents

Unlike simple reflex agents, model-based reflex agents consider the percept history in their actions. The agent function can still work well even in an environment that is not fully observable. These agents use an internal model that determines the percept history and effect of actions. They reflect on certain aspects of the present state that have been unobserved.

Goal-based agents

These agents have higher capabilities than model-based reflex agents. Goal-based agents use goal information to describe desirable capabilities. This allows them to choose among various possibilities. These agents select the best action that enhances the attainment of the goal.

Utility-based agents

These agents make choices based on utility. They are more advanced than goal-based agents because of an extra component of utility

measurement. Using a utility function, a state is mapped against a certain measure of utility. A rational agent selects the action that optimizes the expected utility of the outcome.

Learning agents

These are agents that have the capability of learning from their previous experience.

Learning agents have the following elements.

- The learning element: This element enables learning agents to learn from previous experiences.
- The critic: It provides feedback on how the agent is doing.
- The performance element: This element decides on the external action that needs to be taken.
- The problem generator: This acts as a feedback agent that performs certain tasks such as making suggestions (new) and keeping history.

How intelligent agents work

Intelligent agents work through three main components: sensors, actuators, and effectors. Getting an overview of these components can improve our understanding of how intelligent agents work.

- Sensors: These are devices that detect any changes in the environment. This information is sent to other devices. In artificial intelligence, the environment of the system is observed by intelligent agents through sensors.
- Actuators: These are components through which energy is converted into motion. They perform the role of controlling and moving a system. Examples include rails, motors, and gears.
- Effectors: The environment is affected by effectors. Examples include legs, fingers, wheels, display screen, and arms.

Inputs (percepts) from the environment are received by the intelligent agent through sensors. This agent uses artificial intelligence to make decisions using the acquired information/ observations. Actions are then triggered through actuators. Future decisions will be influenced by percept history and past actions.

Applications of intelligent agents

Intelligent agents in artificial intelligence have been applied in many real-life situations.

Information search, retrieval, and navigation

Intelligent agents enhance the access and navigation of information. This is achieved through the search of information using search engines. The internet consists of many data objects that may take users a lot of time to search for a specific data object. Intelligent agents perform this task on behalf of users within a short time.

Repetitive office activities

Some companies have automated certain administrative tasks to reduce operating costs. Some of the functional areas that have been automated include customer support and sales. Intelligent agents have also been used to enhance office productivity.

Medical diagnosis

Intelligent agents have also been applied in healthcare services to improve the health of patients. In this case, the patient is considered as the environment. The computer keyboard is used as the sensor that receives data on the symptoms of the patient.

The intelligent agent uses this information to decide the best course of action. Medical care is given through actuators such as tests and treatments.

Vacuum cleaning

AI agents are also used to enhance efficiency and cleanness in vacuum cleaning.

In this case, the environment can be a room, table, or carpet. Some of the sensors employed in vacuum cleaning include cameras, bump sensors, and dirt detection sensors. Action is initiated by actuators such as brushes, wheels, and vacuum extractors.

Autonomous driving

Intelligent agents enhance the operation of self-driving cars. In autonomous driving, various sensors are employed to collect information from the environment.

These include cameras, GPS, and radar. In this application, the environment can be pedestrians, other vehicles, roads, or road signs. Various actuators are used to initiate actions. For example, brakes are used to bring the car to a stop.

EXPLORING INTELLIGENT AGENTS IN ARTIFICIAL INTELLIGENCE

Artificial Intelligence, typically abbreviated to AI, is a fascinating field of Information Technology that finds its way into many aspects of modern life. Although it may seem complex, and yes, it is, we can gain a greater familiarity and comfort with AI by exploring its components separately. When we learn how the pieces fit together, we can better understand and implement them.

That's why today we're tackling the intelligent Agent in AI. This chapter defines intelligent agents in Artificial Intelligence, AI agent functions and structure, and the number and types of agents in AI.

An Agent in AI

Okay, did anyone, upon hearing the term "intelligent agent," immediately picture a well-educated spy with a high IQ? No? Anyway, in the context of the AI field, an "agent" is an independent program or entity that interacts with its environment by perceiving its surroundings via sensors, then acting through actuators or effectors.

Agents use their actuators to run through a cycle of perception, thought, and action. Examples of agents in general terms include:

- Software: This Agent has file contents, keystrokes, and received network packages that function as sensory input, then act on those inputs, displaying the output on a screen.
- Human: Yes, we're all agents. Humans have eyes, ears, and other organs that act as sensors, and hands, legs, mouths, and other body parts act as actuators.
- Robotic: Robotic agents have cameras and infrared range finders that act as sensors, and various servos and motors perform as actuators.

Intelligent agents in AI are autonomous entities that act upon an environment using sensors and actuators to achieve their goals. In addition, intelligent agents may learn from the environment to achieve those goals. Driverless cars and the Siri virtual assistant are examples of intelligent agents in AI.

These are the main four rules all AI agents must adhere to:

- Rule 1: An AI agent must be able to perceive the environment.

- Rule 2: The environmental observations must be used to make decisions.
- Rule 3: The decisions should result in action.
- Rule 4: The action taken by the AI agent must be a rational. Rational actions are actions that maximize performance and yield the best positive outcome.

The Functions of an Artificial Intelligence Agent

Artificial Intelligence agents perform these functions continuously:

- Perceiving dynamic conditions in the environment
- Acting to affect conditions in the environment
- Using reasoning to interpret perceptions
- Problem-solving
- Drawing inferences
- Determining actions and their outcomes

THE NUMBER AND TYPES OF AGENTS IN ARTIFICIAL INTELLIGENCE

There are five different types of intelligent agents used in AI. They are defined by their range of capabilities and intelligence level:

- Reflex Agents: These agents work here and now and ignore the past. They respond using the event-condition-action rule. The ECA rule applies when a user initiates an event, and the Agent turns to a list of pre-set conditions and rules, resulting in pre-programmed outcomes.
- Model-based Agents: These agents choose their actions like reflex agents do, but they have a better comprehensive view of the environment. An environmental model is programmed into the internal system, incorporating into the Agent's history.
- Goal-based agents: These agents build on the information that a model-based agent stores by augmenting it with goal information or data regarding desirable outcomes and situations.
- Utility-based agents: These are comparable to the goal-based agents, except they offer an extra utility measurement. This measurement rates each possible scenario based on the desired result and selects the action that maximizes the outcome. Rating criteria examples

include variables such as success probability or the number of resources required.

- Learning agents: These agents employ an additional learning element to gradually improve and become more knowledgeable over time about an environment. The learning element uses feedback to decide how the performance elements should be gradually changed to show improvement.

The Structure of Agents in Artificial Intelligence

Agents in Artificial Intelligence follow this simple structural formula:

Architecture + Agent Program = Agent

These are the terms most associated with agent structure:

- Architecture: This is the machinery or platform that executes the agent.
- Agent Function: The agent function maps a precept to the Action, represented by the following formula: f:P* - A
- Agent Program: The agent program is an implementation of the agent function. The agent program produces function f by executing on the physical architecture.

Many AI Agents use the PEAS model in their structure. PEAS is an acronym for Performance Measure, Environment, Actuators, and Sensors. For instance, take a vacuum cleaner.

- Performance: Cleanliness and efficiency
- Environment: Rug, hardwood floor, living room
- Actuator: Brushes, wheels, vacuum bag
- Sensors: Dirt detection sensor, bump sensor

Agents in Artificial Intelligence

Agents in Artificial Intelligence contain the following properties:

Environment

The agent is situated in a given environment.

Autonomous

The agent can operate without direct human intervention or other software methods. It controls its activities and internal environment. The agent independently which steps it will take in its current condition to

achieve the best improvements. The agent achieves autonomy if its performance is measured by its experiences in the context of learning and adapting.

Flexibility

- Reactive: Agents must recognize their surroundings and react to the changes within them.
- Proactive: Agents shouldn't only act in response to their surroundings but also be able to take the initiative when appropriate and effect an opportunistic, goal-directed performance.
- Social: Agents should work with humans or other non-human agents.

Reactive

- Reactive systems maintain ongoing interactions with their environment, responding to its changes.
- The program's environment may be guaranteed, not concerned about its success or failure.
- Most environments are dynamic, meaning that things are constantly in a state of change, and information is incomplete.
- Programs must make provisions for the possibility of failure.

Pro-Activeness

Taking the initiative to create goals and try to meet them.

Using Response Rules

The goal for the agent is directed behavior, having it do things for the user.

- Mobility: The agent must have the ability to actuate around a system.
- Veracity: If an agent's information is false, it will not communicate.
- Benevolence: Agents don't have contradictory or conflicting goals. Therefore, every Agent will always try to do what it is asked.
- Rationality: The agent will perform to accomplish its goals and not work in a way that opposes or blocks them.
- Learning: An agent must be able to learn.

Improve the Performance of Intelligent Agents

When tackling the issue of how to improve intelligent Agent performances, all we need to do is ask ourselves, "How do we improve

our performance in a task?" The answer, of course, is simple. We perform the task, remember the results, then adjust based on our recollection of previous attempts.

Artificial Intelligence Agents improve in the same way. The Agent gets better by saving its previous attempts and states, learning how to respond better next time. This place is where Machine Learning and Artificial Intelligence meet.

All About Problem-Solving Agents in Artificial Intelligence

Problem-solving Agents in Artificial Intelligence employ several algorithms and analyses to develop solutions. They are:

- Search Algorithms: Search techniques are considered universal problem-solving methods. Problem-solving or rational agents employ these algorithms and strategies to solve problems and generate the best results.

Uninformed Search Algorithms: Also called a Blind search, uninformed searches have no domain knowledge, working instead in a brute-force manner.

Informed Search Algorithms: Also known as a Heuristic search, informed searches use domain knowledge to find the search strategies needed to solve the problem.

- Hill Climbing Algorithms: Hill climbing algorithms are local search algorithms that continuously move upwards, increasing their value or elevation until they find the best solution to the problem or the mountain's peak.

Hill climbing algorithms are excellent for optimizing mathematical problem-solving.

This algorithm is also known as a "greedy local search" because it only checks out its good immediate neighbor.

- Means-Ends Analysis: The means-end analysis is a problem-solving technique used to limit searches in Artificial Intelligence programs, combining Backward and Forward search techniques.

The means-end analysis evaluates the differences between the Initial State and the Final State, then picks the best operators that can be used for each difference. The analysis then applies the operators to each matching difference, reducing the current and goal state difference.

AGENTS IN ARTIFICIAL INTELLIGENCE

Artificial intelligence is defined as the study of rational agents. A rational agent could be anything that makes decisions, as a person, firm, machine, or software. It carries out an action with the best outcome after considering past and current percepts(agent's perceptual inputs at a given instance). An AI system is composed of an agent and its environment. The agents act in their environment. The environment may contain other agents.

An agent is anything that can be viewed as :

- perceiving its environment through sensors and
- acting upon that environment through actuators

To understand the structure of Intelligent Agents, we should be familiar with *Architecture* and *Agent* programs. Architecture is the machinery that the agent executes on. It is a device with sensors and actuators, for example, a robotic car, a camera, a PC. Agent program is an implementation of an agent function. An agent function is a map from the percept sequence(history of all that an agent has perceived to date) to an action.

Agent = Architecture + Agent Program

Examples of Agent:

- A software agent has Keystrokes, file contents, received network packages which act as sensors and displays on the screen, files, sent network packets acting as actuators.
- A Human-agent has eyes, ears, and other organs which act as sensors, and hands, legs, mouth, and other body parts acting as actuators.
- A Robotic agent has Cameras and infrared range finders which act as sensors and various motors acting as actuators.

Types of Agents

Agents can be grouped into five classes based on their degree of perceived intelligence and capability :

- Simple Reflex Agents
- Model-Based Reflex Agents
- Goal-Based Agents
- Utility-Based Agents
- Learning Agent

Simple reflex agents

Simple reflex agents ignore the rest of the percept history and act only on the basis of the current percept. Percept history is the history of all that an agent has perceived to date. The agent function is based on the condition-action rule. A condition-action rule is a rule that maps a state i.e, condition to an action. If the condition is true, then the action is taken, else not. This agent function only succeeds when the environment is fully observable. For simple reflex agents operating in partially observable environments, infinite loops are often unavoidable. It may be possible to escape from infinite loops if the agent can randomize its actions.

Problems with Simple reflex agents are :

- Very limited intelligence.
- No knowledge of non-perceptual parts of the state.
- Usually too big to generate and store.
- If there occurs any change in the environment, then the collection of rules need to be updated.

Model-based reflex agents

It works by finding a rule whose condition matches the current situation. A model-based agent can handle partially observable environments by the use of a model about the world. The agent has to keep track of the internal state which is adjusted by each percept and that depends on the percept history. The current state is stored inside the agent which maintains some kind of structure describing the part of the world which cannot be seen.

Updating the state requires information about :

- how the world evolves independently from the agent, and
- how the agent's actions affect the world.

Goal-based agents

These kinds of agents take decisions based on how far they are currently from their goal(description of desirable situations). Their every action is intended to reduce its distance from the goal. This allows the agent a way to choose among multiple possibilities, selecting the one which reaches a goal state. The knowledge that supports its decisions is represented explicitly and can be modified, which makes these agents more flexible. They usually require search and planning. The goal-based agent's behavior can easily be changed.

Utility-based agents

The agents which are developed having their end uses as building blocks are called utility-based agents. When there are multiple possible alternatives, then to decide which one is best, utility-based agents are used. They choose actions based on a preference (utility) for each state. Sometimes achieving the desired goal is not enough. We may look for a quicker, safer, cheaper trip to reach a destination. Agent happiness should be taken into consideration.

Utility describes how "happy" the agent is. Because of the uncertainty in the world, a utility agent chooses the action that maximizes the expected utility. A utility function maps a state onto a real number which describes the associated degree of happiness.

INTELLIGENT AGENTS

An intelligent agent is an autonomous entity which act upon an environment using sensors and actuators for achieving goals. An intelligent agent may learn from the environment to achieve their goals. A thermostat is an example of an intelligent agent.

Following are the main four rules for an AI agent:

- Rule 1: An AI agent must have the ability to perceive the environment.
- Rule 2: The observation must be used to make decisions.
- Rule 3: Decision should result in an action.
- Rule 4: The action taken by an AI agent must be a rational action.

Rational Agent

A rational agent is an agent which has clear preference, models uncertainty, and acts in a way to maximize its performance measure with all possible actions.

A rational agent is said to perform the right things. AI is about creating rational agents to use for game theory and decision theory for various real-world scenarios.

For an AI agent, the rational action is most important because in AI reinforcement learning algorithm, for each best possible action, agent gets the positive reward and for each wrong action, an agent gets a negative reward.

Rationality

The rationality of an agent is measured by its performance measure. Rationality can be judged on the basis of following points:

- Performance measure which defines the success criterion.
- Agent prior knowledge of its environment.
- Best possible actions that an agent can perform.
- The sequence of percepts.

Structure of an AI Agent

The task of AI is to design an agent program which implements the agent function. The structure of an intelligent agent is a combination of architecture and agent program. It can be viewed as:

1. Agent = Architecture + Agent program

Following are the main three terms involved in the structure of an AI agent:

Architecture: Architecture is machinery that an AI agent executes on.

Agent Function: Agent function is used to map a percept to an action.

1. $f:P^* \rightarrow A$

Agent program: Agent program is an implementation of agent function. An agent program executes on the physical architecture to produce function f.

PEAS Representation

PEAS is a type of model on which an AI agent works upon. When we define an AI agent or rational agent, then we can group its properties under PEAS representation model. It is made up of four words:

- P: Performance measure
- E: Environment
- A: Actuators
- S: Sensors

Here performance measure is the objective for the success of an agent's behavior.

PEAS for self-driving cars:

Let's suppose a self-driving car then PEAS representation will be:

Performance: Safety, time, legal drive, comfort

Environment: Roads, other vehicles, road signs, pedestrian

Actuators: Steering, accelerator, brake, signal, horn

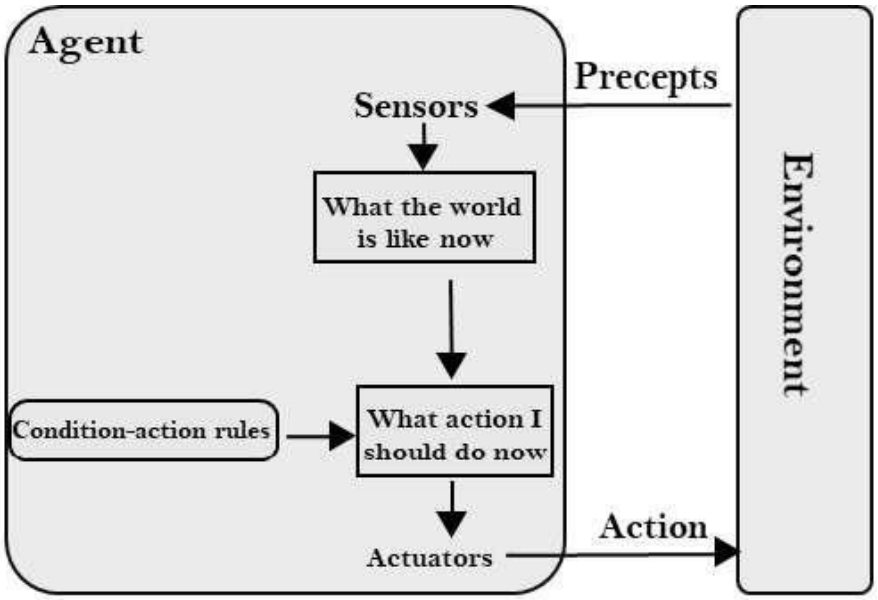
Sensors: Camera, GPS, speedometer, odometer, accelerometer, sonar.

TYPES OF AI AGENTS

Agents can be grouped into five classes based on their degree of perceived intelligence and capability. All these agents can improve their performance and generate better action over the time. These are given below:

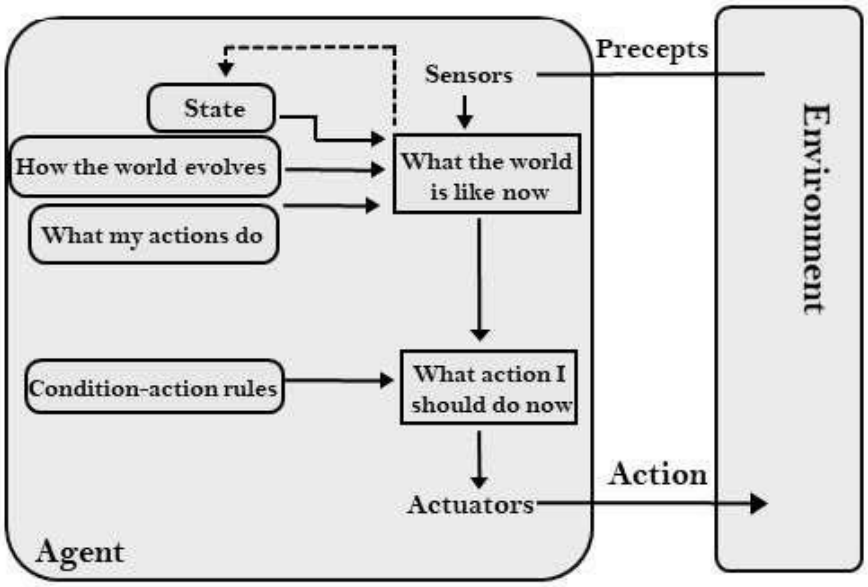
Simple Reflex agent:

- The Simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts and ignore the rest of the percept history.
- These agents only succeed in the fully observable environment.
- The Simple reflex agent does not consider any part of percepts history during their decision and action process.
- The Simple reflex agent works on Condition-action rule, which means it maps the current state to action. Such as a Room Cleaner agent, it works only if there is dirt in the room.
- Problems for the simple reflex agent design approach:
  - o They have very limited intelligence
  - o They do not have knowledge of non-perceptual parts of the current state
  - o Mostly too big to generate and to store.
  - o Not adaptive to changes in the environment.



Model-based reflex agent

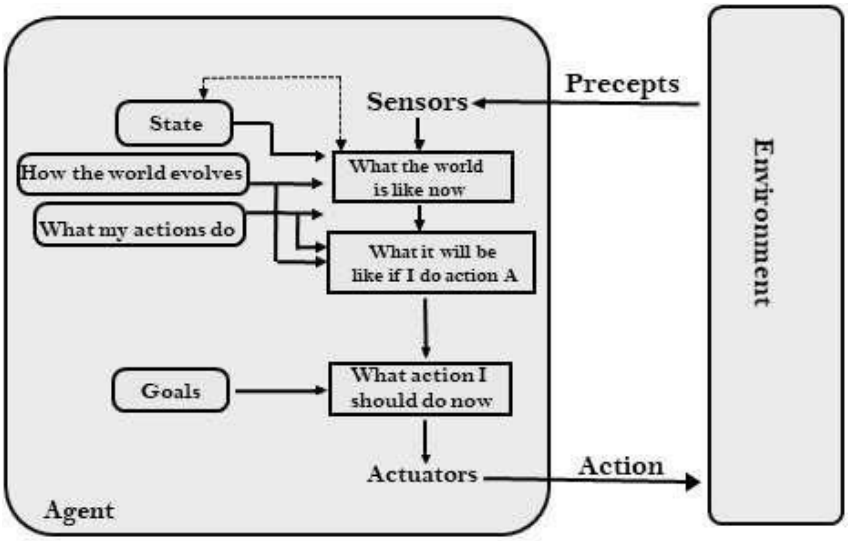
- The Model-based agent can work in a partially observable environment, and track the situation.



- A model-based agent has two important factors:
  - o Model: It is knowledge about "how things happen in the world," so it is called a Model-based agent.
  - o Internal State: It is a representation of the current state based on percept history.
- These agents have the model, "which is knowledge of the world" and based on the model they perform actions.
- Updating the agent state requires information about:
  1. How the world evolves
  2. How the agent's action affects the world.

Goal-based agents

- The knowledge of the current state environment is not always sufficient to decide for an agent to what to do.

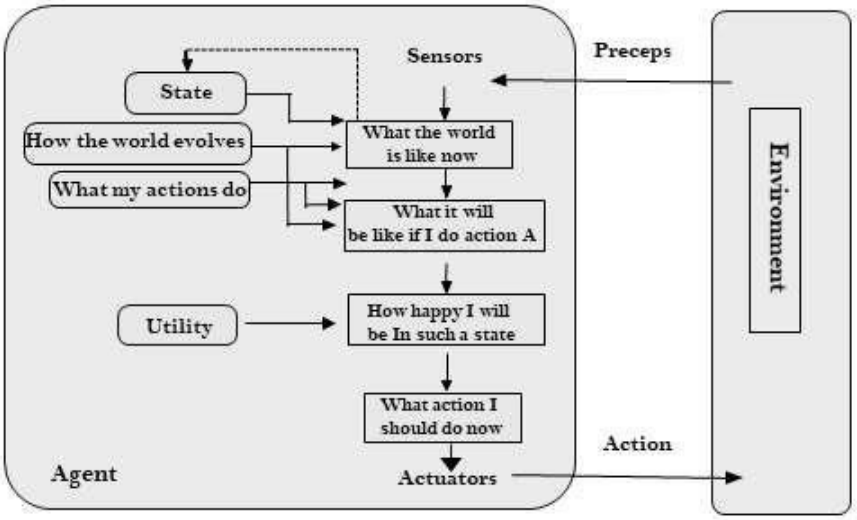


- The agent needs to know its goal which describes desirable situations.
- Goal-based agents expand the capabilities of the model-based agent by having the "goal" information.
- They choose an action, so that they can achieve the goal.
- These agents may have to consider a long sequence of possible actions before deciding whether the goal is achieved or not. Such

considerations of different scenario are called searching and planning, which makes an agent proactive.

Utility-based agents

- These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.
- Utility-based agent act based not only goals but also the best way to achieve the goal.
- The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- The utility function maps each state to a real number to check how efficiently each action achieves the goals.

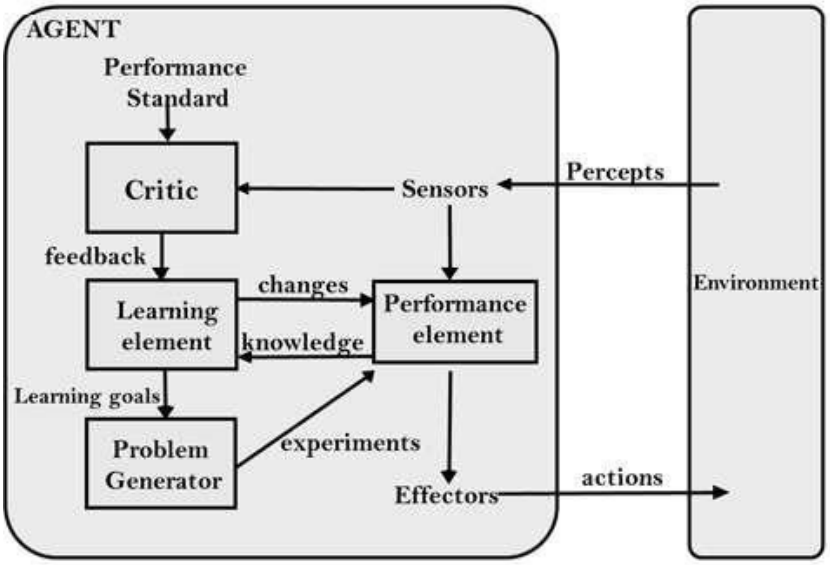


Learning Agents

- A learning agent in AI is the type of agent which can learn from its past experiences, or it has learning capabilities.
- It starts to act with basic knowledge and then able to act and adapt automatically through learning.
- A learning agent has mainly four conceptual components, which are:

1. Learning element: It is responsible for making improvements by learning from environment
2. Critic: Learning element takes feedback from critic which describes that how well the agent is doing with respect to a fixed performance standard.
3. Performance element: It is responsible for selecting external action
4. Problem generator: This component is responsible for suggesting actions that will lead to new and informative experiences.

- Hence, learning agents are able to learn, analyze performance, and look for new ways to improve the performance.



AI - AGENTS & ENVIRONMENTS

An AI system is composed of an agent and its environment. The agents act in their environment. The environment may contain other agents.

Agent and Environment

An agent is anything that can perceive its environment through sensors and acts upon that environment through effectors.

- A human agent has sensory organs such as eyes, ears, nose, tongue

and skin parallel to the sensors, and other organs such as hands, legs, mouth, for effectors.

- A robotic agent replaces cameras and infrared range finders for the sensors, and various motors and actuators for effectors.
- A software agent has encoded bit strings as its programs and actions.

Agent Terminology

- Performance Measure of Agent " It is the criteria, which determines how successful an agent is.
- Behavior of Agent " It is the action that agent performs after any given sequence of percepts.
- Percept " It is agent's perceptual inputs at a given instance.
- Percept Sequence " It is the history of all that an agent has perceived till date.
- Agent Function " It is a map from the precept sequence to an action.

Rationality

Rationality is nothing but status of being reasonable, sensible, and having good sense of judgment.

Rationality is concerned with expected actions and results depending upon what the agent has perceived. Performing actions with the aim of obtaining useful information is an important part of rationality.

Ideal Rational Agent

An ideal rational agent is the one, which is capable of doing expected actions to maximize its performance measure, on the basis of "

- Its percept sequence
- Its built-in knowledge base

Rationality of an agent depends on the following "

- The performance measures, which determine the degree of success.
- Agent's Percept Sequence till now.
- The agent's prior knowledge about the environment.
- The actions that the agent can carry out.

A rational agent always performs right action, where the right action means the action that causes the agent to be most successful in the given percept sequence. The problem the agent solves is characterized by Performance Measure, Environment, Actuators, and Sensors (PEAS).

The Structure of Intelligent Agents

Agent's structure can be viewed as "

- Agent = Architecture + Agent Program
- Architecture = the machinery that an agent executes on.
- Agent Program = an implementation of an agent function.

Simple Reflex Agents

- They choose actions only based on the current percept.
- They are rational only if a correct decision is made only on the basis of current precept.
- Their environment is completely observable.

Condition-Action Rule " It is a rule that maps a state (condition) to an action.

Model Based Reflex Agents

They use a model of the world to choose their actions. They maintain an internal state.

Model " knowledge about "how the things happen in the world".

Internal State " It is a representation of unobserved aspects of current state depending on percept history.

Updating the state requires the information about "

- How the world evolves.
- How the agent's actions affect the world.

Goal Based Agents

They choose their actions in order to achieve goals. Goal-based approach is more flexible than reflex agent since the knowledge supporting a decision is explicitly modeled, thereby allowing for modifications.

Goal " It is the description of desirable situations.

Utility Based Agents

They choose actions based on a preference (utility) for each state.

Goals are inadequate when "

- There are conflicting goals, out of which only few can be achieved.
- Goals have some uncertainty of being achieved and you need to weigh likelihood of success against the importance of a goal.

The Nature of Environments

Some programs operate in the entirely artificial environment confined to keyboard input, database, computer file systems and character output on a screen.

In contrast, some software agents (software robots or softbots) exist in rich, unlimited softbots domains. The simulator has a very detailed, complex environment. The software agent needs to choose from a long array of actions in real time. A softbot designed to scan the online preferences of the customer and show interesting items to the customer works in the real as well as an artificial environment.

The most famous artificial environment is the Turing Test environment, in which one real and other artificial agents are tested on equal ground. This is a very challenging environment as it is highly difficult for a software agent to perform as well as a human.

Turing Test

The success of an intelligent behavior of a system can be measured with Turing Test.

Two persons and a machine to be evaluated participate in the test. Out of the two persons, one plays the role of the tester. Each of them sits in different rooms. The tester is unaware of who is machine and who is a human. He interrogates the questions by typing and sending them to both intelligences, to which he receives typed responses. This test aims at fooling the tester. If the tester fails to determine machine's response from the human response, then the machine is said to be intelligent.

Properties of Environment

The environment has multifold properties "

- Discrete / Continuous " If there are a limited number of distinct, clearly defined, states of the environment, the environment is discrete (For example, chess); otherwise it is continuous (For example, driving).
- Observable / Partially Observable " If it is possible to determine the complete state of the environment at each time point from the percepts it is observable; otherwise it is only partially observable.
- Static / Dynamic " If the environment does not change while an agent is acting, then it is static; otherwise it is dynamic.

- Single agent / Multiple agents " The environment may contain other agents which may be of the same or different kind as that of the agent.
- Accessible / Inaccessible " If the agent's sensory apparatus can have access to the complete state of the environment, then the environment is accessible to that agent.
- Deterministic / Non-deterministic " If the next state of the environment is completely determined by the current state and the actions of the agent, then the environment is deterministic; otherwise it is non-deterministic.
- Episodic / Non-episodic " In an episodic environment, each episode consists of the agent perceiving and then acting. The quality of its action depends just on the episode itself. Subsequent episodes do not depend on the actions in the previous episodes. Episodic environments are much simpler because the agent does not need to think ahead.

METHODS AND GOALS IN AI

Symbolic vs. connectionist approaches

AI research follows two distinct, and to some extent competing, methods, the symbolic (or "top-down") approach, and the connectionist (or "bottom-up") approach. The top-down approach seeks to replicate intelligence by analyzing cognition independent of the biological structure of the brain, in terms of the processing of symbols—whence the *symbolic* label. The bottom-up approach, on the other hand, involves creating artificial neural networks in imitation of the brain's structure—whence the *connectionist* label. To illustrate the difference between these approaches, consider the task of building a system, equipped with an optical scanner, that recognizes the letters of the alphabet.

A bottom-up approach typically involves training an artificial neural network by presenting letters to it one by one, gradually improving performance by "tuning" the network. (Tuning adjusts the responsiveness of different neural pathways to different stimuli.)

In contrast, a top-down approach typically involves writing a computer program that compares each letter with geometric descriptions. Simply put, neural activities are the basis of the bottom-up approach, while symbolic descriptions are the basis of the top-down approach.

In *The Fundamentals of Learning* (1932), Edward Thorndike, a psychologist at Columbia University, New York City, first suggested that human learning consists of some unknown property of connections between neurons in the brain. In *The Organization of Behavior* (1949), Donald Hebb, a psychologist at McGill University, Montreal, Canada, suggested that learning specifically involves strengthening certain patterns of neural activity by increasing the probability (weight) of induced neuron firing between the associated connections.

In 1957 two vigorous advocates of symbolic AI—Allen Newell, a researcher at the RAND Corporation, Santa Monica, California, and Herbert Simon, a psychologist and computer scientist at Carnegie Mellon University, Pittsburgh, Pennsylvania—summed up the top-down approach in what they called the physical symbol system hypothesis. This hypothesis states that processing structures of symbols is sufficient, in principle, to produce artificial intelligence in a digital computer and that, moreover, human intelligence is the result of the same type of symbolic manipulations.

During the 1950s and '60s the top-down and bottom-up approaches were pursued simultaneously, and both achieved noteworthy, if limited, results. During the 1970s, however, bottom-up AI was neglected, and it was not until the 1980s that this approach again became prominent.

Nowadays both approaches are followed, and both are acknowledged as facing difficulties. Symbolic techniques work in simplified realms but typically break down when confronted with the real world; meanwhile, bottom-up researchers have been unable to replicate the nervous systems of even the simplest living things. *Caenorhabditis elegans*, a much-studied worm, has approximately 300 neurons whose pattern of interconnections is perfectly known. Yet connectionist models have failed to mimic even this worm. Evidently, the neurons of connectionist theory are gross oversimplifications of the real thing.

Strong AI, applied AI, and cognitive simulation

Employing the methods outlined above, AI research attempts to reach one of three goals: strong AI, applied AI, or cognitive simulation. Strong AI aims to build machines that think. (The term *strong AI* was introduced for this category of research in 1980 by the philosopher John Searle of the University of California at Berkeley.) The ultimate ambition of strong AI is to produce a machine whose overall intellectual ability is indistinguishable from that of a human being. As is described in the section Early milestones

in AI, this goal generated great interest in the 1950s and '60s, but such optimism has given way to an appreciation of the extreme difficulties involved. To date, progress has been meagre. Some critics doubt whether research will produce even a system with the overall intellectual ability of an ant in the forseeable future. Indeed, some researchers working in AI's other two branches view strong AI as not worth pursuing.

Applied AI, also known as advanced information processing, aims to produce commercially viable "smart" systems—for example, "expert" medical diagnosis systems and stock-trading systems. Applied AI has enjoyed considerable success, as described in the section Expert systems.

In cognitive simulation, computers are used to test theories about how the human mind works—for example, theories about how people recognize faces or recall memories. Cognitive simulation is already a powerful tool in both neuroscience and cognitive psychology.

ALAN TURING AND THE BEGINNING OF AI

Theoretical work

The earliest substantial work in the field of artificial intelligence was done in the mid-20th century by the British logician and computer pioneer Alan Mathison Turing. In 1935 Turing described an abstract computing machine consisting of a limitless memory and a scanner that moves back and forth through the memory, symbol by symbol, reading what it finds and writing further symbols.

The actions of the scanner are dictated by a program of instructions that also is stored in the memory in the form of symbols. This is Turing's stored-program concept, and implicit in it is the possibility of the machine operating on, and so modifying or improving, its own program. Turing's conception is now known simply as the universal Turing machine. All modern computers are in essence universal Turing machines. During World War II, Turing was a leading cryptanalyst at the Government Code and Cypher School in Bletchley Park, Buckinghamshire, England. Turing could not turn to the project of building a stored-program electronic computing machine until the cessation of hostilities in Europe in 1945. Nevertheless, during the war he gave considerable thought to the issue of machine intelligence. One of Turing's colleagues at Bletchley Park, Donald Michie (who later founded the Department of Machine Intelligence and Perception at the University of Edinburgh), later recalled that Turing often discussed

how computers could learn from experience as well as solve new problems through the use of guiding principles—a process now known as heuristic problem solving.

Turing gave quite possibly the earliest public lecture (London, 1947) to mention computer intelligence, saying, "What we want is a machine that can learn from experience," and that the "possibility of letting the machine alter its own instructions provides the mechanism for this." In 1948 he introduced many of the central concepts of AI in a report entitled "Intelligent Machinery." However, Turing did not publish this paper, and many of his ideas were later reinvented by others. For instance, one of Turing's original ideas was to train a network of artificial neurons to perform specific tasks, an approach described in the section Connectionism.

Chess

At Bletchley Park, Turing illustrated his ideas on machine intelligence by reference to chess—a useful source of challenging and clearly defined problems against which proposed methods for problem solving could be tested. In principle, a chess-playing computer could play by searching exhaustively through all the available moves, but in practice this is impossible because it would involve examining an astronomically large number of moves.

Heuristics are necessary to guide a narrower, more discriminative search. Although Turing experimented with designing chess programs, he had to content himself with theory in the absence of a computer to run his chess program. The first true AI programs had to await the arrival of stored-program electronic digital computers.

In 1945 Turing predicted that computers would one day play very good chess, and just over 50 years later, in 1997, Deep Blue, a chess computer built by the International Business Machines Corporation (IBM), beat the reigning world champion, Garry Kasparov, in a six-game match. While Turing's prediction came true, his expectation that chess programming would contribute to the understanding of how human beings think did not. The huge improvement in computer chess since Turing's day is attributable to advances in computer engineering rather than advances in AI—Deep Blue's 256 parallel processors enabled it to examine 200 million possible moves per second and to look ahead as many as 14 turns of play. Many agree with Noam Chomsky, a linguist at the Massachusetts Institute of Technology (MIT), who opined that a computer beating a

grandmaster at chess is about as interesting as a bulldozer winning an Olympic weightlifting competition.

ARTIFICIAL INTELLIGENCE FRAMEWORK: A VISUAL INTRODUCTION TO MACHINE LEARNING AND AI

The transformative nature of Artificial Intelligence in business and our society is evident. Like the internet and the smartphone, AI is an enabler technology that will have a far-reaching impact on all areas of our life. You will find many great articles here on medium or elsewhere which give a very good introduction to AI (I have linked some at the end of this chapter). Nevertheless, what I have not seen so far is an AI overview which is easy to digest visually.

If you know something I could use instead, go ahead and let me know in the comments section below. But until then, let me share the visualisation that I am using right now. We will also briefly walk through the key components of the figure in this chapter.

Artificial Intelligence in the First Place

In brief, Artificial Intelligence is a branch within computer science that studies how to create machines which possess capabilities similar to human intelligence. There are different maturity levels of artificial intelligence which I will briefly touch upon in the upcoming chapters: superintelligence, general Artificial Intelligence and narrow Artificial Intelligence.

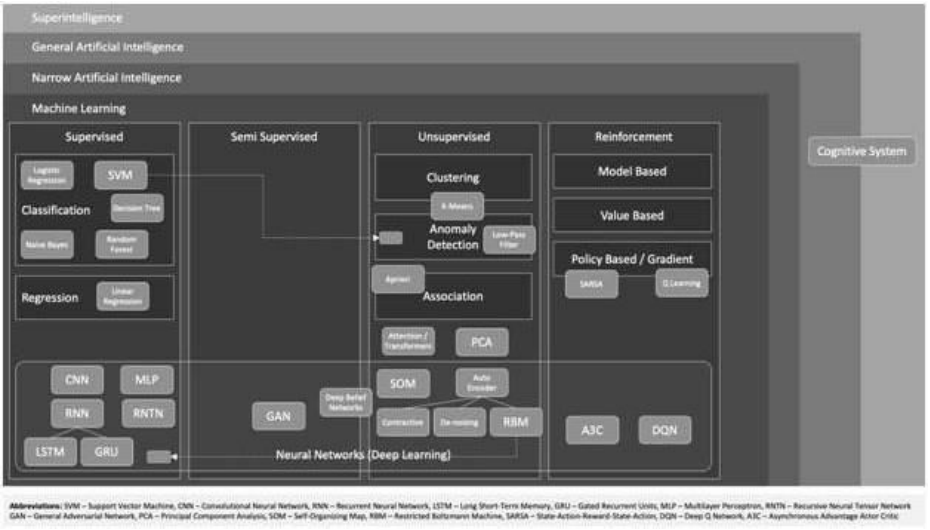
Artificial Intelligence is very useful in order to digitize cognitive capabilities where the exact rules to follow are difficult to explain. A good AI use case would be face recognition. Trying to use handcrafted knowledge to code all relevant rules for face recognition would be an approach sometimes referred to as the first wave of AI (source). But with mature technology like machine learning and deep learning available.

Key Drivers of Artificial Intelligence

Before we dig into the details, let's take a quick look at the key factors that drive recent advances in Artificial intelligence:

- Computing power: The price performance of computing power has grown exponentially in alignment with Moore's Law. Exponential means that the computing speed doubles and/or the price drops by half year over year. In recent years, machine learning as one of the key drivers of AI advances has greatly benefited from GPUs

(Graphics Processing Unit). GPUs are very performant for conducting vectorised numerical operations which are needed for all machine learning calculations. Google's TPU (Tensor Processing Unit) is another example where (co)processors are optimized for machine learning problems. With great advances in quantum computing it is very likely that this trend will continue and accelerate.



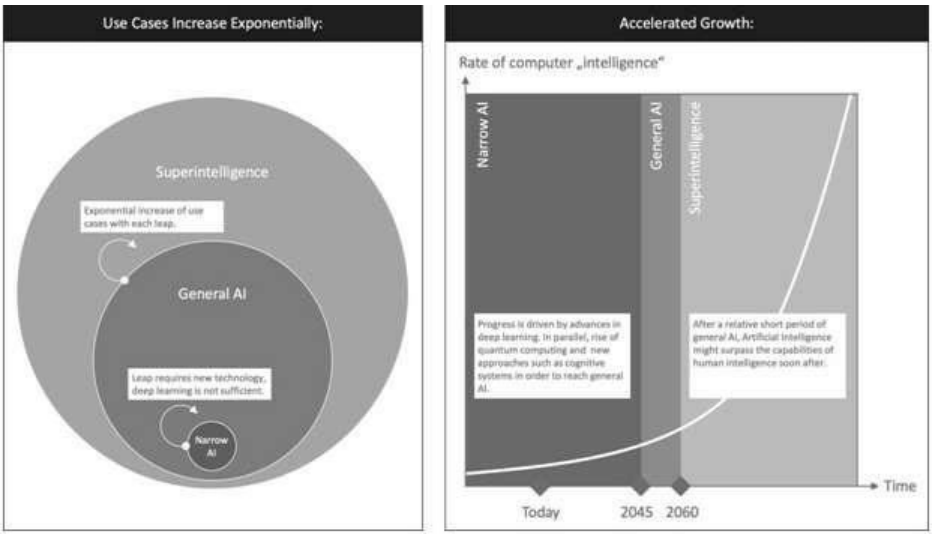
- Availability of data: There is an accelerated generation and availability of data fueled by the increased use of mobile technology and social media. In the last two years alone, an impressive 90% of the world's data was generated (source). Large amounts of data are the key success factor in order to train neural networks (more information further below) and therefore achieve a high accuracy of their predictions for unseen events.
- Algorithms: The AI research community is very active and new advances are published frequently. The biggest attention is on machine learning, or to be more specific on deep neural networks. There are also various tools and frameworks developed and ready for public use by Google (Tensorflow), Facebook (PyTorch) or Microsoft (Azure) to name a few.

Superintelligence

Some researchers predict and believe that at a certain point in the future, machines will be smarter than humans. This might happen somewhere between the years 2050 and 2100. Few researches argue that

this state will never be reached. But for most researchers familiar with the subject, it is not a matter whether superintelligence will be reached but rather when it will be reached.

Superintelligence is the state where a machine's cognitive capability surpasses that of humans. As Nick Bostrom puts it, it is an organism with an "intellect that greatly exceeds the cognitive performance of humans in virtually all domains of interest".



The advent of superintelligence is based on the assumption that the rate of progress in an evolutionary learning environment is evolving exponentially (Ray Kurzweil's law of accelerating returns). This is difficult to grasp because most people think rather linearly and try to predict the future from what we already know and have experienced in the past. This will of course lead to a wrong trajectory.

There is a great and fun article by Tim Urban explaining this journey (link at the end of the article). It explains superintelligence and all its implications much better than I will ever be able to. I highly recommend to read his post if you are in the mood.

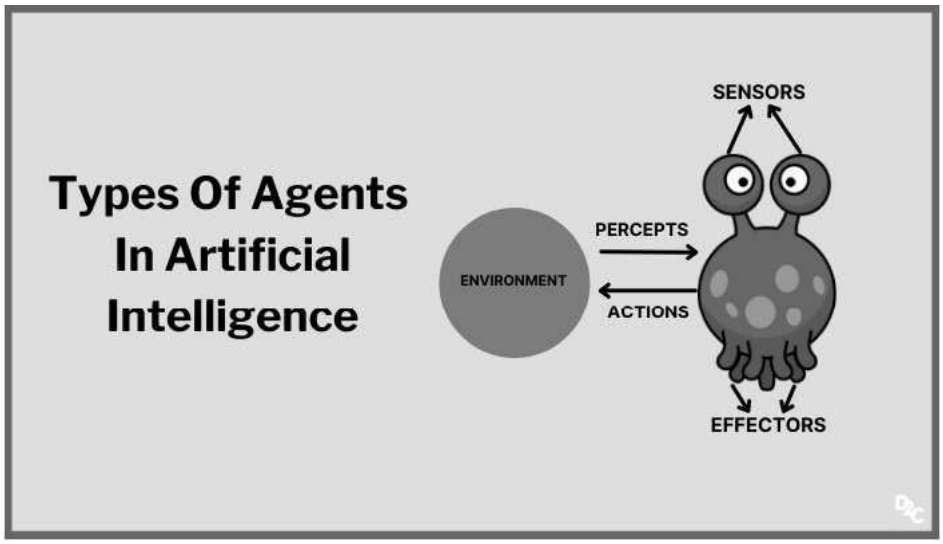
In the best case, superintelligence will lead to a future of abundance and equity (singularity). In the worst scenario it will lead to the extinction of mankind, not because superintelligence is evil but simply because humans are in its way on achieving its goals (goals that we might not be able to understand any longer since they surpass our cognitive capabilities).

TYPES OF AGENTS IN ARTIFICIAL INTELLIGENCE

Artificial intelligence refers to the study of rational agents to make decisions related to a person, firm, machine or software. Considering the past and present perceptual inputs of an agent at a particular instant, AI carries out a task with the best outcome possible. AI system comprises of agent and its environment. One particular environment consists of various agents.

Agent is a part of AI system that takes actions or decisions based on the information it perceives from the environment. For example, Robot Agent utilizes information it senses from the environment using the sensors in order to carry out a particular action.

On the other hand, Human Agent uses sensory organs to sense the environment and takes particular actions and decisions regarding the body parts of the human.



Structure of AI Agents

An AI agent comprises of Architecture and an Agent program. Architecture involves machinery for execution of tasks by agents. It consists of a device with sensors and effectors or actuators. An agent program refers to the process of implementation of an agent function, which is map of the percept sequence or the perceptual history of the agent for a particular action.

RATIONAL AGENT

A rational agent or rational being is a person or entity that always aims to perform optimal actions based on given premises and information. A rational agent can be anything that makes decisions, typically a person, firm, machine, or software. The concept of rational agents can be found in various disciplines such as artificial intelligence, cognitive science, decision theory, economics, ethics, game theory, and the study of practical reason.

Economics

In reference to economics, rational agent refers to hypothetical consumers and how they make decisions in a free market. This concept is one of the assumptions made in neoclassical economic theory. The concept of economic rationality arises from a tradition of marginal analysis used in neoclassical economics. The idea of a rational agent is important to the philosophy of utilitarianism, as detailed by philosopher Jeremy Bentham's theory of the felicific calculus, also known as the hedonistic calculus.

The action a rational agent takes depends on:

- the preferences of the agent
- the agent's information of its environment, which may come from past experiences
- the actions, duties and obligations available to the agent
- the estimated or actual benefits and the chances of success of the actions.

In game theory and classical economics, it is often assumed that the actors, people, and firms are rational. However, the extent to which people and firms behave rationally is subject to debate. Economists often assume the models of rational choice theory and bounded rationality to formalize and predict the behavior of individuals and firms. Rational agents sometimes behave in manners that are counter-intuitive to many people, as in the traveler's dilemma.

Criticisms

Many economic theories reject utilitarianism and rational agency, especially those that might be considered heterodox.

For example, Thorstein Veblen, known as the father of institutional

economics, rejects the notion of hedonistic calculus and pure rationality saying: "The hedonistic conception of man is that of a lightning calculator of pleasures and pains who oscillates like a homogeneous globule of desire of happiness under the impulse of stimuli that shift him about the area, but leave him intact."

Veblen instead perceives human economic decisions as the result of multiple complex cumulative factors: "It is the characteristic of man to do something, not simply to suffer pleasures and pains through the impact of suitable forces. He is ... a coherent structure of propensities and habits which seeks realization and expression in an unfolding activity They are the products of his hereditary traits and his past experience, cumulatively wrought out under a given body of traditions conventionalities, and material circumstances; and they afford the point of departure for the next step in the process. The economic life history of the individual is a cumulative process of adaptation of means to ends that cumulatively change as the process goes on, both the agent and his environment being at any point the outcome of the last process." Evolutionary economics also provides criticisms of the Rational Agent, citing the "parental bent" (the idea that biological impulses can and do frequently override rational decision making based on utility). Arguments against rational agency have also cited the enormous influence of marketing as proof that humans can be persuaded to make economic decisions that are "non-rational" in nature.

Alternate theories

Neuroeconomics is a concept that uses neuroscience, social psychology and other fields of science to better understand how people make decisions. Unlike rational agent theory, neuroeconomics does not attempt to predict large-scale human behavior but rather how individuals make decisions in case-by-case scenarios.

Artificial intelligence

Artificial intelligence has borrowed the term "rational agents" from economics to describe autonomous programs that are capable of goal directed behavior. Today there is a considerable overlap between AI research, game theory and decision theory. Rational agents in AI are closely related to *intelligent agents*, autonomous software programs that display intelligence.

INTERACTION OF AGENTS WITH ENVIRONMENT

Interaction of the Agent with the environment uses Sensors and Effectors. Sensors perceive the environment and the actuators or effectors act upon that environment.

This interaction can occur in two different ways:

1. Perception: Perception is a passive interaction between the agent and the environment where the environment remains unchanged when the agent takes up information from the environment. This involves gaining information using 'Sensors' from the surroundings without any change to the surroundings.
2. Action: Action is an active interaction between the agent and the environment where the environment changes when the action is performed. This involves utilization of an 'Effector' or an 'Actuator' which completes an action but leads to changes in the surroundings while doing so.

For example, in case of a virtual agent, when the virtual agent reads and interprets the information provided by the user, it is known as 'Perception' while when it replies to the user based on the interpretation it is known as 'Action'.

Action of Agents In Artificial Intelligence

Agents in Artificial Intelligence act by:

1. Mapping of the Percept sequences or Perceptual history to the Actions: Mapping refers to a list that maps a particular percept sequence to the action. The design for an ideal agent can be figured out by specifying an action corresponding to the percept sequence or the perceptual history.
2. Autonomy: The agent designer determines the behavior of the agent by determining its experience and its built-in knowledge. Autonomy refers to taking actions based on the experience of the agent. If the system comprises of an autonomous intelligent agent then it is able to operate and adapt successfully in a wide range of environments.

Examples of Agents

Some examples of agents are as follows:

1. Software agent: It comprises of sensors like Keystrokes, file contents,

received network packages and actuators or effectors like displays on the screen, files, sent network packets.

2. Human agent: It comprises of sensors like eyes, ears, and other sensory organs and actuators or effectors like hands, legs, mouth, and other body parts.
3. Robotic agent: It comprises of sensors like Cameras and infrared range finders and actuators or effectors in the form of various motors.

Types of Agents

Based on their degree of perceived intelligence and capability, Agents can be divided into the following types:

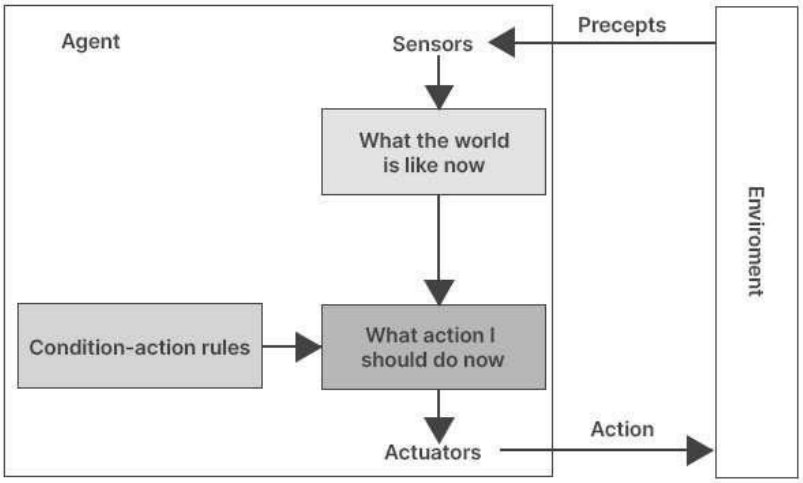
1. SIMPLE REFLEX AGENTS
2. MODEL-BASED AGENTS
3. GOAL-BASED AGENTS
4. UTILITY-BASED AGENTS
5. LEARNING AGENTS

Performance can be improved and better action can be generated for each of these types of agents in AI.

Simple Reflex Agents

1. This is a simple type of agent which works on the basis of current percept and not based on the rest of the percepts history.
2. The agent function, in this case, is based on condition-action rule where the condition or the state is mapped to the action such that action is taken only when condition is true or else it is not.
3. If the environment associated with this agent is fully observable, only then is the agent function successful, if it is partially observable, in that case the agent function enters into infinite loops that can be escaped only on randomization of its actions.
4. The problems associated with this type include very limited intelligence, No knowledge of non-perceptual parts of the state, huge size for generation and storage and inability to adapt to changes in the environment.

This can be illustrated using the following image:



Model-Based Agents

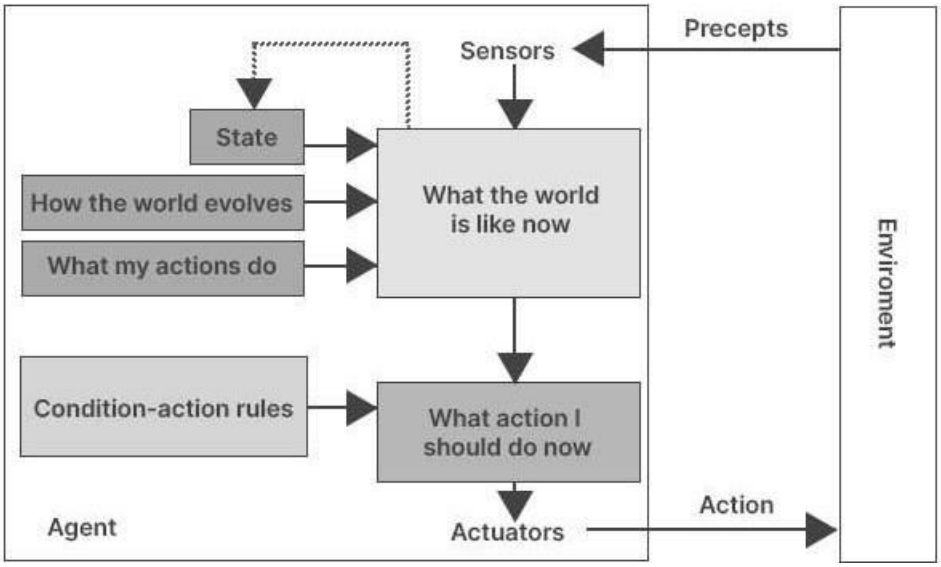
1. Model-Based Reflex Agent utilizes condition-action rule where it works by finding a rule that will allow the condition, which is based on the current situation, to be satisfied.
2. Irrespective of the first type, it can handle partially observable environments by tracking the situation and using a particular model related to the world.
3. It consists of two important factors, which are Model and Internal State.
4. Model provides knowledge and understanding of the process of occurrence of different things in the surroundings such that the current situation can be studied and a condition can be created. Actions are performed by the agent based on this model.
5. Internal State uses the perceptual history to represent a current percept. The agent keeps a track of this internal state and is adjusted by each of the percepts. The current internal state is stored by the agent inside it to maintain a kind of structure that can describe the unseen world.
6. The state of the agent can be updated by gaining information about how the world evolves and how the agent's action affects the world.

This can be illustrated as:

Goal-Based Agents

1. This type takes decisions on the basis of its goal or desirable situations

so that it can choose such an action that can achieve the goal required.

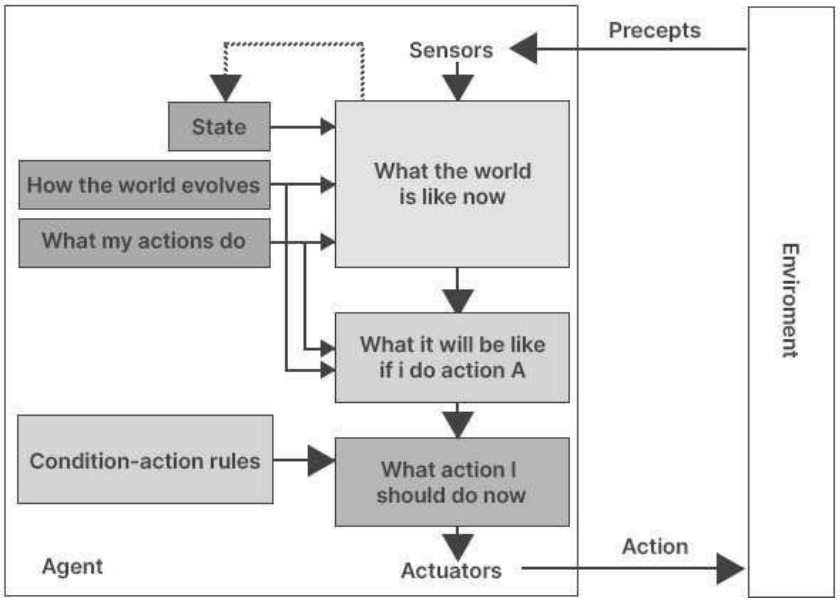


2. It is an improvement over model based agent where information about the goal is also included. This is because it is not always sufficient to know just about the current state, knowledge of the goal is a more beneficial approach.
3. The aim is to reduce the distance between action and the goal so that the best possible way can be chosen from multiple possibilities. Once the best way is found, the decision is represented explicitly which makes the agent more flexible.
4. It carries out considerations of different situations called searching and planning by considering long sequence of possible actions for confirming its ability to achieve the goal. This makes the agent proactive.
5. It can easily change its behavior if required.

This can be illustrated as follows:

Utility-Based Agents

1. Utility agent have their end uses as their building blocks and is used when best action and decision needs to be taken from multiple alternatives.



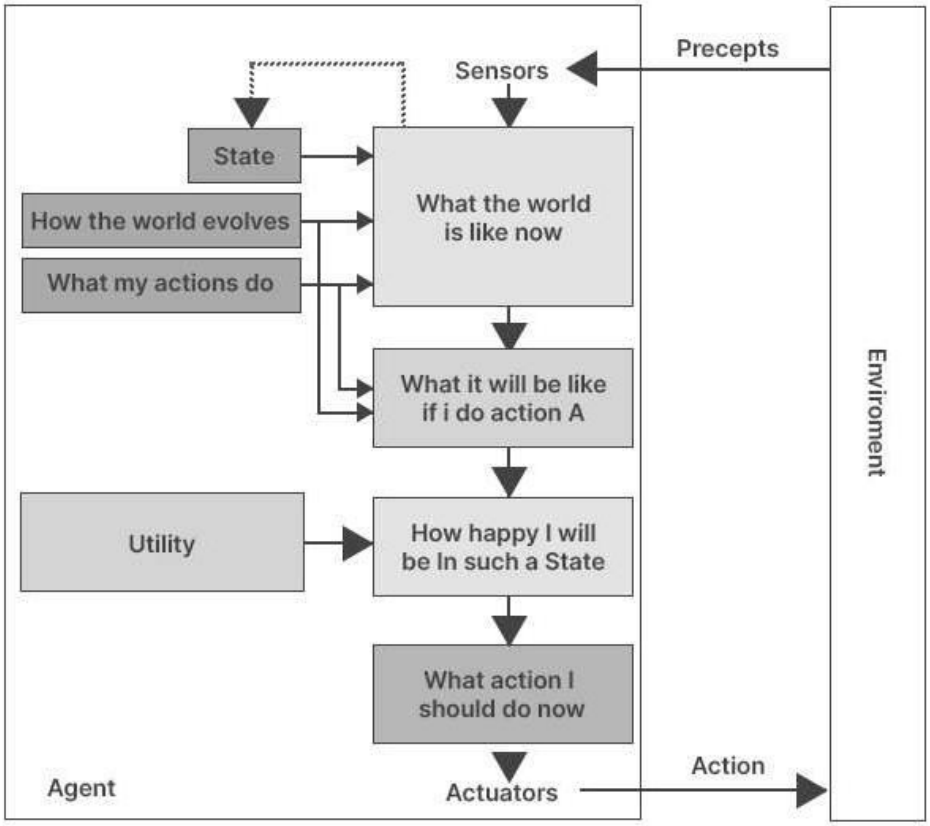
2. It is an improvement over goal based agent as it not only involves the goal but also the way the goal can be achieved such that the goal can be achieved in a quicker, safer, cheaper way.
3. The extra component of utility or method to achieve a goal provides a measure of success at a particular state that makes the utility agent different.
4. It takes the agent happiness into account and gives an idea of how happy the agent is because of the utility and hence, the action with maximum utility is considered. This associated degree of happiness can be calculated by mapping a state onto a real number.
5. Mapping of a state onto a real number with the help of utility function gives the efficiency of an action to achieve the goal.

This can be illustrated as follows:

Learning Agents

1. Learning agent, as the name suggests, has the capability to learn from past experiences and takes actions or decisions based on learning capabilities.
2. It gains basic knowledge from past and uses that learning to act and adapt automatically.

3. It comprises of four conceptual components, which are given as follows:
   - Learning element: It makes improvements by learning from the environment.
   - Critic: Critic provides feedback to the learning agent giving the performance measure of the agent with respect to the fixed performance standard.



   - Performance element: It selects the external action.
   - Problem generator: This suggests actions that lead to new and informative experiences.

This can be illustrated as follows:

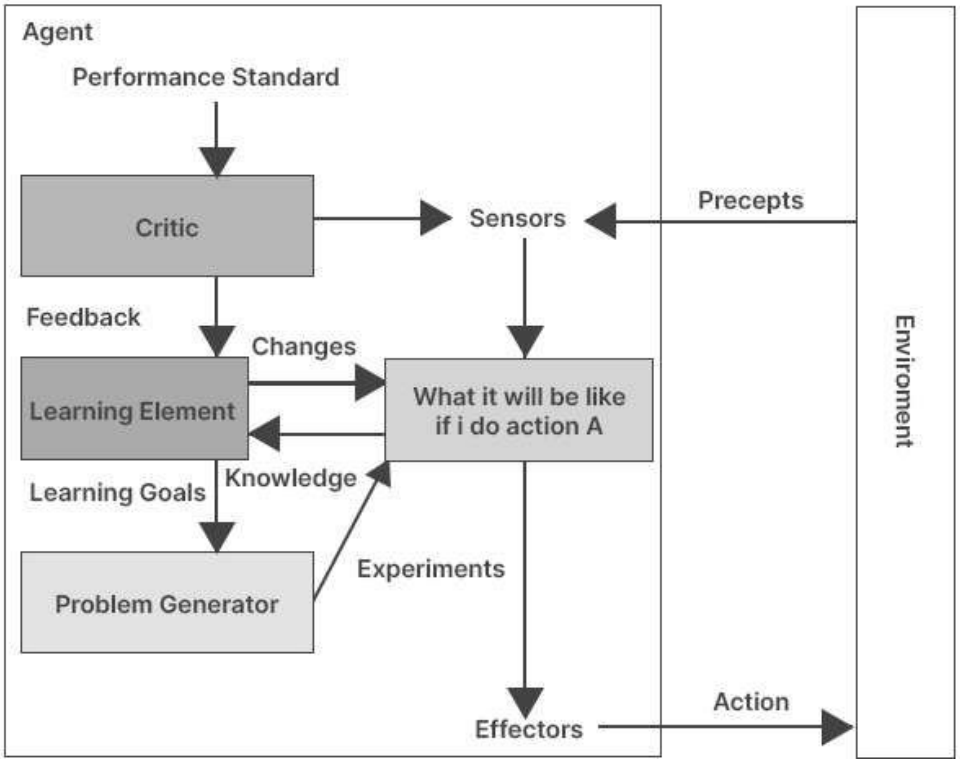
Summing Up

This chapter brings the following points to the attention of the readers:

1. Artificial intelligence refers to the study of rational agents to make

decisions related to a person, firm, machine or software. AI system comprises of agent and its environment.

2. Agent is a part of AI system that takes actions or decisions based on the information it perceives from the environment.
3. Agents interact with the environment using sensors and actuators or effectors in two different ways which are perception and Action.



4. Perception is a passive interaction between the agent and the environment where the environment remains unchanged when the agent takes up information from the environment while Action is an active interaction between them where the environment changes when the action is performed.
5. The agents in AI act by Mapping of the Percept sequences or Perceptual history to the Actions and Autonomy.
6. Based on their degree of perceived intelligence and capability, Agents can be divided into five types which are Simplex reflex agent, Model Based agent, Goal based agent, Utility agent and Learning agent.

4

Applications of Artificial Intelligence

Artificial intelligence, defined as intelligence exhibited by machines, has many applications in today's society. More specifically, it is Weak AI, the form of AI where programs are developed to perform specific tasks, that is being utilized for a wide range of activities including medical diagnosis, electronic trading platforms, robot control, and remote sensing. AI has been used to develop and advance numerous fields and industries, including finance, healthcare, education, transportation, and more.

AI for Good

AI for Good is an ITU initiative supporting institutions employing AI to tackle some of the world's greatest economic and social challenges. For example, the University of Southern California launched the Center for Artificial Intelligence in Society, with the goal of using AI to address socially relevant problems such as homelessness. At Stanford, researchers are using AI to analyze satellite images to identify which areas have the highest poverty levels.

Agriculture

In agriculture new AI advancements show improvements in gaining yield and to increase the research and development of growing crops. New artificial intelligence now predicts the time it takes for a crop like a tomato to be ripe and ready for picking thus increasing efficiency of farming. These advances go on including Crop and Soil Monitoring, Agricultural Robots, and Predictive Analytics. Crop and soil monitoring uses new algorithms and data collected on the field to manage and track the health of crops making it easier and more sustainable for the farmers.

More specializations of AI in agriculture is one such as greenhouse automation, simulation, modeling, and optimization techniques.

Due to the increase in population and the growth of demand for food in the future there will need to be at least a 70% increase in yield from agriculture to sustain this new demand. More and more of the public perceives that the adaption of these new techniques and the use of Artificial intelligence will help reach that goal.

Aviation

The Air Operations Division (AOD) uses AI for the rule based expert systems. The AOD has use for artificial intelligence for surrogate operators for combat and training simulators, mission management aids, support systems for tactical decision making, and post processing of the simulator data into symbolic summaries.

The use of artificial intelligence in simulators is proving to be very useful for the AOD. Airplane simulators are using artificial intelligence in order to process the data taken from simulated flights. Other than simulated flying, there is also simulated aircraft warfare. The computers are able to come up with the best success scenarios in these situations. The computers can also create strategies based on the placement, size, speed and strength of the forces and counter forces. Pilots may be given assistance in the air during combat by computers. The artificial intelligent programs can sort the information and provide the pilot with the best possible maneuvers, not to mention getting rid of certain maneuvers that would be impossible for a human being to perform. Multiple aircraft are needed to get good approximations for some calculations so computer simulated pilots are used to gather data. These computer simulated pilots are also used to train future air traffic controllers.

The system used by the AOD in order to measure performance was the Interactive Fault Diagnosis and Isolation System, or IFDIS. It is a rule based expert system put together by collecting information from TF-30 documents and the expert advice from mechanics that work on the TF-30. This system was designed to be used for the development of the TF-30 for the RAAF F-111C. The performance system was also used to replace specialized workers. The system allowed the regular workers to communicate with the system and avoid mistakes, miscalculations, or having to speak to one of the specialized workers.

The AOD also uses artificial intelligence in speech recognition software.

The air traffic controllers are giving directions to the artificial pilots and the AOD wants to the pilots to respond to the ATC's with simple responses. The programs that incorporate the speech software must be trained, which means they use neural networks. The program used, the Verbex 7000, is still a very early program that has plenty of room for improvement. The improvements are imperative because ATCs use very specific dialog and the software needs to be able to communicate correctly and promptly every time.

The Artificial Intelligence supported Design of Aircraft, or AIDA, is used to help designers in the process of creating conceptual designs of aircraft. This program allows the designers to focus more on the design itself and less on the design process. The software also allows the user to focus less on the software tools. The AIDA uses rule based systems to compute its data. This is a diagram of the arrangement of the AIDA modules. Although simple, the program is proving effective.

In 2003, NASA's Dryden Flight Research Center, and many other companies, created software that could enable a damaged aircraft to continue flight until a safe landing zone can be reached. The software compensates for all the damaged components by relying on the undamaged components. The neural network used in the software proved to be effective and marked a triumph for artificial intelligence.

The Integrated Vehicle Health Management system, also used by NASA, on board an aircraft must process and interpret data taken from the various sensors on the aircraft. The system needs to be able to determine the structural integrity of the aircraft. The system also needs to implement protocols in case of any damage taken the vehicle. Haitham Baomar and Peter Bentley are leading a team from the University College of London to develop an artificial intelligence based Intelligent Autopilot System (IAS) designed to teach an autopilot system to behave like a highly experienced pilot who is faced with an emergency situation such as severe weather, turbulence, or system failure. Educating the autopilot relies on the concept of supervised machine learning "which treats the young autopilot as a human apprentice going to a flying school". The autopilot records the actions of the human pilot generating learning models using artificial neural networks. The autopilot is then given full control and observed by the pilot as it executes the training exercise.

The Intelligent Autopilot System combines the principles of Apprenticeship Learning and Behavioural Cloning whereby the autopilot

observes the low-level actions required to maneuver the airplane and high-level strategy used to apply those actions. IAS implementation employs three phases; pilot data collection, training, and autonomous control. Baomar and Bentley's goal is to create a more autonomous autopilot to assist pilots in responding to emergency situations.

Computer science

AI researchers have created many tools to solve the most difficult problems in computer science. Many of their inventions have been adopted by mainstream computer science and are no longer considered a part of AI. According to Russell & Norvig , all of the following were originally developed in AI laboratories: time sharing, interactive interpreters, graphical user interfaces and the computer mouse, Rapid application development environments, the linked list data structure, automatic storage management, symbolic programming, functional programming, dynamic programming and object-oriented programming.

AI can be used to potentially determine the developer of anonymous binaries.

AI can be used to create other AI. For example, around November 2017, Google's AutoML project to evolve new neural net topologies created NASNet, a system optimized for ImageNet and COCO. According to Google, NASNet's performance exceeded all previously published ImageNet performance.

Deepfake

In June 2016, a research team from the visual computing group of the Technical University of Munich and from Stanford University developed Face2Face, a program which animates the face of a target person, transposing the facial expressions of an exterior source. The technology has been demonstrated animating the lips of people including Barack Obama and Vladimir Putin. Since then, other methods have been demonstrated based on deep neural network, from which the name "deepfake" was taken.

Hollywood film studios had already used the technique in animated films, but it took time and efforts from professionals. The main difference is that today anyone can use a deep fake software and rig videos.

In September 2018, the U.S. Senator Mark Warner proposed to penalize social media companies that allow sharing of deepfake documents on their platform.

Vincent Nozick, a researcher from the Institut Gaspard Monge, found a way to detect rigged documents by analyzing the movements of the eyelid. The DARPA (a research group associated with the U.S. Department of Defense) has given 68 million dollars to work on deepfake detection. In Europe, the Horizon 2020 program financed InVid, software designed to help journalists to detect fake documents.

Education

AI tutors could allow for students to get extra, one-on-one help. They could also reduce anxiety and stress for some students, that may be caused by tutor labs or human tutors. In future classrooms, ambient informatics can play a beneficial role. Ambient informatics is the idea that information is everywhere in the environment and that technologies automatically adjust to your personal preferences.

Study devices could be able to create lessons, problems, and games to tailor to the specific student's needs, and give immediate feedback.

But AI can also create a disadvantageous environment with revenge effects, if technology is inhibiting society from moving forward and causing negative, unintended effects on society. An example of a revenge effect is that the extended use of technology may hinder students' ability to focus and stay on task instead of helping them learn and grow. Also, AI has been known to lead to the loss of both human agency and simultaneity.

Finance

Algorithmic trading

Algorithmic trading involves the use of complex AI systems to make trading decisions at speeds several orders of magnitudes greater than any human is capable of, often making millions of trades in a day without any human intervention. Such trading is called High-frequency Trading, and it represents one of the fastest growing sectors in financial trading. Many banks, funds, and proprietary trading firms now have entire portfolios which are managed purely by AI systems. Automated trading systems are typically used by large institutional investors, but recent years have also seen an influx of smaller, proprietary firms trading with their own AI systems.

Market analysis and data mining

Several large financial institutions have invested in AI engines to

assist with their investment practices. BlackRock's AI engine, Aladdin, is used both within the company and to clients to help with investment decisions. Its wide range of functionalities includes the use of natural language processing to read text such as news, broker reports, and social media feeds. It then gauges the sentiment on the companies mentioned and assigns a score. Banks such as UBS and Deutsche Bank use an AI engine called Sqreem (Sequential Quantum Reduction and Extraction Model) which can mine data to develop consumer profiles and match them with the wealth management products they'd most likely want. Goldman Sachs uses Kensho, a market analytics platform that combines statistical computing with big data and natural language processing. Its machine learning systems mine through hoards of data on the web and assess correlations between world events and their impact on asset prices. Information Extraction, part of artificial intelligence, is used to extract information from live news feed and to assist with investment decisions.

Personal finance

Several products are emerging that utilize AI to assist people with their personal finances. For example, Digit is an app powered by artificial intelligence that automatically helps consumers optimize their spending and savings based on their own personal habits and goals.

The app can analyze factors such as monthly income, current balance, and spending habits, then make its own decisions and transfer money to the savings account. Wallet.AI, an upcoming startup in San Francisco, builds agents that analyze data that a consumer would leave behind, from Smartphone check-ins to tweets, to inform the consumer about their spending behavior.

Portfolio management

Robo-advisors are becoming more widely used in the investment management industry. Robo-advisors provide financial advice and portfolio management with minimal human intervention. This class of financial advisers work based on algorithms built to automatically develop a financial portfolio according to the investment goals and risk tolerance of the clients. It can adjust to real-time changes in the market and accordingly calibrate the portfolio.

Underwriting

An online lender, Upstart, analyze vast amounts of consumer data and

utilizes machine learning algorithms to develop credit risk models that predict a consumer's likelihood of default. Their technology will be licensed to banks for them to leverage for their underwriting processes as well.

ZestFinance developed their Zest Automated Machine Learning (ZAML) Platform specifically for credit underwriting as well. This platform utilizes machine learning to analyze tens of thousands traditional and nontraditional variables (from purchase transactions to how a customer fills out a form) used in the credit industry to score borrowers. The platform is particularly useful to assign credit scores to those with limited credit histories, such as millennials.

History

The 1980s is really when AI started to become prominent in the finance world. This is when expert systems became more of a commercial product in the financial field. "For example, Dupont had built 100 expert systems which helped them save close to $10 million a year." One of the first systems was the Protrader expert system designed by K.C. Chen and Ting-peng Lian that was able to predict the 87-point drop in DOW Jones Industrial Average in 1986. "The major junctions of the system were to monitor premiums in the market, determine the optimum investment strategy, execute transactions when appropriate and modify the knowledge base through a learning mechanism." One of the first expert systems that helped with financial plans was created by Applied Expert Systems (APEX) called the PlanPower. It was first commercially shipped in 1986. Its function was to help give financial plans for people with incomes over $75,000 a year. That then led to the Client Profiling System that was used for incomes between $25,000 and $200,000 a year. The 1990s was a lot more about fraud detection. One of the systems that was started in 1993 was the FinCEN Artificial Intelligence system (FAIS). It was able to review over 200,000 transactions per week and over two years it helped identify 400 potential cases of money laundering which would have been equal to $1 billion. Although expert systems did not last in the finance world, it did help jump-start the use of AI and help make it what it is today.

Heavy industry

Robots have become common in many industries and are often given jobs that are considered dangerous to humans. Robots have proven effective in jobs that are very repetitive which may lead to mistakes or accidents due to a lapse in concentration and other jobs which humans may find

degrading. In 2014, China, Japan, the United States, the Republic of Korea and Germany together amounted to 70% of the total sales volume of robots. In the automotive industry, a sector with particularly high degree of automation, Japan had the highest density of industrial robots in the world: 1,414 per 10,000 employees.

Hospitals and medicine

Artificial neural networks are used as clinical decision support systems for medical diagnosis, such as in Concept Processing technology in EMR software.

Other tasks in medicine that can potentially be performed by artificial intelligence and are beginning to be developed include:

- Computer-aided interpretation of medical images. Such systems help scan digital images, *e.g.* from computed tomography, for typical appearances and to highlight conspicuous sections, such as possible diseases. A typical application is the detection of a tumor.
- Heart sound analysis
- Companion robots for the care of the elderly
- Mining medical records to provide more useful information.
- Design treatment plans.
- Assist in repetitive jobs including medication management.
- Provide consultations.
- Drug creation
- Using avatars in place of patients for clinical training
- Predict the likelihood of death from surgical procedures
- Predict HIV progression

There are over 90 AI startups in the health industry working in these fields.

IDx's first solution, IDx-DR, is the first autonomous AI-based diagnostic system authorized for commercialization by the FDA.

Human resources and recruiting

Another application of AI is in the human resources and recruiting space. There are three ways AI is being used by human resources and recruiting professionals: to screen resumes and rank candidates according to their level of qualification, to predict candidate success in given roles

through job matching platforms, and rolling out recruiting chat bots that can automate repetitive communication tasks. Typically, resume screening involves a recruiter or other HR professional scanning through a database of resumes.

Job search

The job market has seen a notable change due to artificial intelligence implementation. It has simplified the process for both recruiters and job seekers (i.e., Google for Jobs and applying online). According to Raj Mukherjee from Indeed.com, 65% of people launch a job search again within 91 days of being hired. AI-powered engine streamlines the complexity of job hunting by operating information on job skills, salaries, and user tendencies, matching people to the most relevant positions. Machine intelligence calculates what wages would be appropriate for a particular job, pulls and highlights resume information for recruiters using natural language processing, which extracts relevant words and phrases from text using specialized software. Another application is an AI resume builder which requires 5 minutes to compile a CV as opposed to spending hours doing the same job. In the AI age chatbots assist website visitors and solve daily workflows. Revolutionary AI tools complement people's skills and allow HR managers to focus on tasks of higher priority. However, Artificial Intelligence impact on jobs research suggests that by 2030 intelligent agents and robots can eliminate 30% of the world's human labor. Moreover, the research proves automation will displace between 400 and 800 million employees. Glassdoor's research report states that recruiting and HR are expected to see much broader adoption of AI in job market 2018 and beyond.

Media and e-commerce

Some AI applications are geared towards the analysis of audiovisual media content such as movies, TV programs, advertisement videos or user-generated content. The solutions often involve computer vision, which is a major application area of AI.

Typical use case scenarios include the analysis of images using object recognition or face recognition techniques, or the analysis of video for recognizing relevant scenes, objects or faces. The motivation for using AI-based media analysis can be — among other things — the facilitation of media search, the creation of a set of descriptive keywords for a media item, media content policy monitoring (such as verifying the suitability

of content for a particular TV viewing time), speech to text for archival or other purposes, and the detection of logos, products or celebrity faces for the placement of relevant advertisements.

Media analysis AI companies often provide their services over a REST API that enables machine-based automatic access to the technology and allows machine-reading of the results. For example, IBM, Microsoft, Amazon and the video AI company Valossa allow access to their media recognition technology by using RESTful APIs.

AI is also widely used in E-commerce applications like visual search, chatbots, and automated product tagging. Another generic application is to increase search discoverability and making social media content shoppable.

Military

The main military applications of Artificial Intelligence and Machine Learning are to enhance Command and Control, Communications, Sensors, Integration and Interoperability. Artificial Intelligence technologies enables coordination of sensors and effectors, threat detection and identification, marking of enemy positions, target acquisition, coordination and deconfliction of distributed Join Fires between networked combat vehicles and tanks also inside Manned and Unmanned Teams (MUM-T).

Music

While the evolution of music has always been affected by technology, artificial intelligence has enabled, through scientific advances, to emulate, at some extent, human-like composition.

Among notable early efforts, David Cope created an AI called Emily Howell that managed to become well known in the field of Algorithmic Computer Music. The algorithm behind Emily Howell is registered as a US patent.

The AI Iamus created 2012 the first complete classical album fully composed by a computer.

Other endeavours, like AIVA (Artificial Intelligence Virtual Artist), focus on composing symphonic music, mainly classical music for film scores. It achieved a world first by becoming the first virtual composer to be recognized by a musical professional association.

Artificial intelligences can even produce music usable in a medical setting, with Melomics's effort to use computer-generated music for stress

and pain relief. Moreover, initiatives such as Google Magenta, conducted by the Google Brain team, want to find out if an artificial intelligence can be capable of creating compelling art.

At Sony CSL Research Laboratory, their Flow Machines software has created pop songs by learning music styles from a huge database of songs. By analyzing unique combinations of styles and optimizing techniques, it can compose in any style.

Another artificial intelligence musical composition project, The Watson Beat, written by IBM Research, doesn't need a huge database of music like the Google Magenta and Flow Machines projects, since it uses Reinforcement Learning and Deep Belief Networks to compose music on a simple seed input melody and a select style. Since the software has been open sourced musicians, such as Taryn Southern have been collaborating with the project to create music.

News, publishing and writing

The company Narrative Science makes computer-generated news and reports commercially available, including summarizing team sporting events based on statistical data from the game in English. It also creates financial reports and real estate analyses. Similarly, the company Automated Insights generates personalized recaps and previews for Yahoo Sports Fantasy Football. The company is projected to generate one billion stories in 2014, up from 350 million in 2013. The organisation OpenAI has also created an AI capable of writing text.

Echobox is a software company that helps publishers increase traffic by 'intelligently' posting articles on social media platforms such as Facebook and Twitter. By analysing large amounts of data, it learns how specific audiences respond to different articles at different times of the day. It then chooses the best stories to post and the best times to post them. It uses both historical and real-time data to understand to what has worked well in the past as well as what is currently trending on the web.

Another company, called Yseop, uses artificial intelligence to turn structured data into intelligent comments and recommendations in natural language. Yseop is able to write financial reports, executive summaries, personalized sales or marketing documents and more at a speed of thousands of pages per second and in multiple languages including English, Spanish, French & German.

Boomtrain's is another example of AI that is designed to learn how

to best engage each individual reader with the exact articles—sent through the right channel at the right time—that will be most relevant to the reader. It's like hiring a personal editor for each individual reader to curate the perfect reading experience.

IRIS.TV is helping media companies with its AI-powered video personalization and programming platform. It allows publishers and content owners to surface contextually relevant content to audiences based on consumer viewing patterns.

Beyond automation of writing tasks given data input, AI has shown significant potential for computers to engage in higher-level creative work.

AI Storytelling has been an active field of research since James Meehan's development of TALESPIN, which made up stories similar to the fables of Aesop. The program would start with a set of characters who wanted to achieve certain goals, with the story as a narration of the characters' attempts at executing plans to satisfy these goals.

Since Meehan, other researchers have worked on AI Storytelling using similar or different approaches. Mark Riedl and Vadim Bulitko argued that the essence of storytelling was an experience management problem, or "how to balance the need for a coherent story progression with user agency, which are often at odds."

While most research on AI storytelling has focused on story generation (e.g. character and plot), there has also been significant investigation in story communication.

In 2002, researchers at North Carolina State University developed an architectural framework for narrative prose generation. Their particular implementation was able faithfully reproduced text variety and complexity of a number of stories, such as red riding hood, with human-like adroitness. This particular field continues to gain interest. In 2016, a Japanese AI co-wrote a short story and almost won a literary prize.

INTELLIGENT AGENTS: CHARACTERISTICS AND APPLICATIONS | AI

Meaning of Intelligent Agents

Intelligent Agents (IA) are software programs which represent a new technology with the potential to become one of the most important tools of information technology in the twenty-first century. IA can alleviate the

most critical limitation of the Internet- information overflow, and can facilitate electronic commerce. Before we look at its capabilities.

Several names are used to describe intelligent agents- software agents, wizards, knowbots and softbots. The names tend to reflect the nature of the agent; the term agent is derived from the concept of agency, which means employing someone to act on the behalf of the user. A computerised agent represents a person and interacts with others to accomplish a predefined task.

A good working definition is this: An intelligent agent is a software entity which senses its environment and then carries out some set of operations on behalf of a user (or a program), with some degree of autonomy, and in so doing employs some knowledge or representation of the user's goals or desires or in other words IA are software programs which work in the background to carry out specific, repetitive, predictable tasks for an individual user, business processor software application.

Characteristics and Applications of Intelligent Agents

Several traits or abilities exist which many people think of when they are discussing about intelligent agents:

a. Capability to work on their own (autonomy)
b. Exhibition of goal-oriented behaviour
c. Transportable over networks (mobility)
d. Dedication to a single repetitive task
e. Ability to interact with humans, systems, and other agents
f. Inclusion of a knowledge base
g. Ability to learn

Although not all intelligent agents have all of these capabilities, they are very useful in facilitating some tasks such as:

1. Information Access and Navigation: Information access is today's major application of intelligent agents, and it is done by use of different search engines.
2. Decision Support and Empowerment: Knowledge workers need support, especially in decision-making.
3. Repetitive Office Activities: There is a pressing need to automate tasks performed by administrative and clerical personnel in functional areas, such as sales or customer support, in order to reduce labour costs and increase office productivity. Today, labour

costs are estimated to be as much as 60 percent of the total cost of information delivery.

4. Mundane Personal Activities: In our fast-paced society, time-strapped individuals need new ways to minimise the time spent on routine personal tasks like booking airline tickets. One specific form of intelligent agents is, the voice- activated interface agent which reduces the burden on the user of having to explicitly command the computer.
5. Search and Retrieval: It is not possible to directly manipulate a distributed database system in a business setting which involves millions of data objects. Users have to delegate the task of searching and cost comparison to agents. These agents perform the tedious, time-consuming, and repetitive tasks of searching databases, retrieving and filtering information, and delivering results to the user.
6. Domain Experts: It is advisable to model costly expertise and make it widely available. "Expert" software agents could be models of real-world agents, such as translators, lawyers, diplomats, union negotiators, stock-brokers, and even clergy.
7. Management Activities: Intelligent agents can even be used to assist managers in performing their activities. Some management-oriented tasks which an agent can do: advise, alert, broadcast, browse, critique, distribute, enlist, empower, explain, filter, guide, identify, match, monitor, navigate, negotiate, organize, present, query, report, remind, retrieve, schedule, search, secure, solicit, store, suggest, summaries, reach, translate and watch.

For example, the wizards found in Microsoft Office software tools have built-in capabilities to show users how to accomplish various tasks, such as formatting documents or a creating graphs, and to anticipate when users need assistance.

At the Almade Research Centre of IBM an IA works which facilitates the learning process for computer programmers who are learning the programming language LISP. COACH (Cognitive Adaptive Computer Help) contains three knowledge components which enable it to function.

One component compiles information about the user's LISP capabilities, including the frequent mistakes. Another component maintains information about LISP itself and the final component stores strategies for coaching. COACH ensures that students of LISP receive a lot more through learning

experience than they would otherwise. Of special interest are intelligent agents used to cruise networks, including the Internet, in search of information AT & T pioneered in this area with its Personal Link service. Personal Link used an object-oriented remote programming language called Telescript from California's General Magic Inc. to establish an environment for e-mail, on-line news and an electronic market place.

The Cambridge, Massachusetts USA Company, Agents Inc., sells an agent which caters to consumers on the Internet. Users send critiques of movies and music to Agents' website, Firefly. When they want to select a new movie or buy a CD, they can supply data on their personal favourites and Firefly will produce a list of similar items based on the critiques.

THE ROOTS OF ARTIFICIAL INTELLIGENCE

The postindustrial society will be fueled not by oil but by a new commodity called artificial intelligence (AI). We might regard it as a commodity because it has value and can be traded. Indeed, as will be made clear, the knowledge imbedded in AI software and hardware architectures will become even more salient as a foundation of wealth than the raw materials that fueled the first Industrial Revolution. It is an unusual commodity, because it has no material form. It can be a flow of information with no more physical reality than electrical vibrations in a wire.

If artificial intelligence is the fuel of the second industrial revolution, then we might ask what it is. One of the difficulties in addressing this issue is the amount of confusion and disagreement regarding the definition of the field. Other fields do not seem to have this problem. Books on biology do not generally begin with the question, What is biology, anyway? Predicting the future is always problematic, but it will be helpful if we attempt to define what it is we are predicting the future of. One view is that AI is an attempt to answer a central question that has been debated by scientists, philosophers, and theologians for thousands of years. How does the human brain-three pounds of "ordinary" matter-give rise to thoughts, feelings, and consciousness? While certainly very complex, our brains are clearly governed by the same physical laws as our machines. Viewed in this way, the human brain may be regarded as a very capable machine. Conversely, given sufficient capacity and the right techniques, our machines may ultimately be able to replicate human intelligence. Some philosophers and even a few AI scientists are offended by this characterization of the human mind as a machine, albeit an immensely

complicated one. Others find the view inspiring: it means that we will ultimately be able to understand our minds and how they work. One does not need to accept fully the notion that the human mind is "just" a machine to appreciate both the potential for machines to master many of our intellectual capabilities and the practical implications of doing so.

The Usual Definition

Artificial Stupidity (AS) may be defined as the attempt by computer scientists to create computer programs capable of causing problems of a type normally associated with human thought.

- Wallace Marshal, Journal of Irreproducible Results (1987)

Probably the most durable definition of artificial intelligence, and the one most often quoted, states that: "Artificial Intelligence is the art of creating machines that perform functions that require intelligence when performed by people." It is reasonable enough as definitions go, although it suffers from two problems. First, it does not say a great deal beyond the words "artificial intelligence." The definition refers to machines and that takes care of the word "artificial." There is no problem here: we have never had much difficulty defining artificial. For the more problematic word "intelligence" the definition provides only a circular definition: an intelligent machine does what an intelligent person does.

A more serious problem is that the definition does not appear to fit actual usage. Few AI researchers refer to the chess-playing machines that one can buy in the local drug store as examples of true artificial intelligence, yet chess is *still* considered an intellectual game. Some equation-manipulation packages perform transformations that would challenge most college students. We consider these to be quite useful packages, but again, they are rarely pointed to as examples of artificial intelligence.

The Moving Frontier Definition

Mr. Jabez Wilson laughed heavily. "Well, I never!" said he. "I thought at first that you had done something clever, but I see that there was nothing in it, after all?" "I began to think, Watson," said Holmes, "that I made a mistake in explaining. 'Omne ignatum pro magnifico,' you know, and my poor little reputation, such as it is, will suffer shipwreck if I am so candid."

- Sir Arthur Conan Doyle, *The Complete Sherlock Holmes*

"The extent to which we regard something as behaving in an intelligent manner is determined as much by our own state of mind and training as by the

properties of the object under consideration. If we are able to explain and predict its behavior or if there seems to be little underlying plan, we have little temptation to imagine intelligence. With the same object, therefore, it is possible that one man would consider it as intelligent and another would not; the second man would have found out the rules of its behavior."

- Alan Turing (1947)

"AI is the study of how to make computers do things at which, at the moment, people are better."

- Elaine Rich

This leads us to another approach, which I like to call the "moving frontier" definition: artificial intelligence is the study of computer problems that have not yet been solved. This definition, which Marvin Minsky has been advocating since the 1960s, is unlike those found in other fields. A gene-splicing technique does not stop being part of bioengineering the moment it is perfected. Yet, if we examine the shifting judgments as to what has qualified as "true artificial intelligence" over the years, we find this definition has more validity than one might expect.

When the artificial intelligence field was first named at a now famous conference held in 1956 at Dartmouth College, programs that could play chess or checkers or manipulate equations, even at crude levels of performance, were very much in the mainstream of AI. As I noted above, we no longer consider such gameplaying programs to be prime examples of AI, although perhaps we should.

One might say that this change in perception simply reflects a tightening of standards. I feel that there is something more profound going on. We are of two minds when it comes to thinking. On the one hand, there is the faith in the AI community that most definable problems can be solved, often by successively breaking them down into hierarchies of simpler problems. While some problems will take longer to solve than others, we presently have no clear limit to what can be achieved.

On the other hand, coexisting with the faith that most cognitive problems can be solved is the feeling that thinking or true intelligence is not an automatic technique. In other words, there is something in the concept of thinking that goes beyond the automatic opening and closing of switches. Thus, when a method has been perfected in a computerized system, we see it as just another useful technique, not as an example of true artificial intelligence. We know exactly how the system works, so it does not seem fundamentally different from any other computer program.

A problem that has *not yet* been solved, on the other hand, retains its mystique. While we may have confidence that such a problem will eventually be solved, we do not yet know its solution. So we do not yet think of it as just an automatic technique and thus allow ourselves to view it as true cybernetic cognition. Consider as a current example the area of artificial intelligence known as expert systems. Such a system consists of a data base of facts about a particular discipline, a *knowledge base* of codified rules for drawing inferences from the data base, and a high-speed *inference engine* for systematically applying the rules to the facts to solve problems. Such systems have been successfully used to locate fuel deposits, design and assemble complex computer systems, analyze electronic circuits, and diagnose diseases. The judgments of expert systems are beginning to rival those of human experts, at least within certain well-defined areas of expertise.

Today expert systems are widely regarded as a central part of artificial intelligence, and hundreds of projects exist today to apply this set of techniques to dozens of fields. It seems likely that expert systems will become within the next ten years as widespread as computer spreadsheet programs and data-base management systems are today. I predict that when this happens, AI researchers will shift their attention to other issues, and we will no longer consider expert systems to be prime examples of AI technology. They will probably be regarded as just obvious extensions of data-base-management techniques.

Roger Schank uses the example of a pool sweep, a robot pool cleaner, to illustrate our tendency to view an automatic procedure as not intelligent. When we first see a pool sweep mysteriously weaving its way around the bottom of a pool, we are impressed with its apparent intelligence in systematically finding its way around.

When we figure out the method or pattern behind its movements, which is a deceptively simple algorithm of making preprogrammed changes in direction every time it encounters a wall of the pool, we realize that it is not very intelligent after all.

Another example is a computer program named ELIZA designed in 1966 by Joseph Weizenbaum to simulate a psychotherapist. When interacting with ELIZA, users type statements about themselves and ELIZA responds with questions and comments. Many persons have been impressed with the apparent appropriateness and insight of ELIZA's ability to engage in psychoanalytic dialog. Those users who have been given the opportunity

to examine ELIZA's algorithms have been even more impressed at how simple some of its methods are.

We often respond to people the same way. When we figure out how an expert operates and understand his or her methods and rules of thumb, what once seemed very intelligent somehow seems less so.

It will be interesting to see what our reaction will be when a computer takes the world chess championship. Playing a master game of chess is often considered an example of high intellectual (even creative) achievement. When a computer does become the chess champion, which I believe will happen before the end of the century, we will either think more of computers, less of ourselves, or less of chess.

Our ambivalence on the issue of the ability of a machine to truly emulate human thought tends to regard a *working*system as possibly useful but not truly intelligent. Computer-science problems are only AI problems until they are solved. This could be seen to be a frustrating state of affairs. As with the carrot on a stick, the AI practitioner can never quite achieve the goal.

ANTICIPATORY SOCIALIZATION AND INTELLIGENCE ANALYSIS

Every organization has a unique culture that is defined partly by its individual members and partly by its structure, history, and policies. For that culture to endure, it must be transmitted from current members to new members. This process, known as organizational socialization, is especially important in organizations with strong, insular cultures, as those with weak cultures have less to transmit and will tend to experience culture changes as members come and go.

Although socialization begins prior to a person's first day on the job and is a continuous process, it is experienced most intensely by new employees. The cultural symbols acquired and interpreted during their initial interaction with the institution create potent and lasting impressions.

For them, socialization is the process of learning the ropes; training; and becoming formally and informally acquainted with what is actually of value within the organization.

It is also the time when one learns the organization's norms and taboos and the extent of its social capital. In sum, formal and informal socialization are types of control mechanism for maintaining the norms, or *status quo*, within any organization.

Organizational Socialization

According to Daniel Feldman, organizational socialization is "the process through which individuals are transformed from outsiders to participating, effective members of an organization." Feldman divides this process into three stages: getting in (or anticipatory socialization), breaking in (or accommodation), and settling in (often referred to as role management). During the getting-in stage, potential employees try to acquire information about an organization from available sources, such as Web sites, professional journals, and corporate annual reports. The breaking-in stage includes orientation and learning organizational as well as job-related procedures. The settling-in stage concludes when an individual attains full member status in the organization.

While each of the three stages of socialization is important, the focus of this chapter is on the first, or anticipatory, stage. There are several reasons for this. Clearly, the expectations people develop about an organization they are joining are important to a new recruit's eventual satisfaction, retention, and performance. Moreover, because it can control several aspects of the recruitment process, this stage is often the easiest for an organization to change. This chapter will take both a descriptive and prescriptive approach to easing the socialization of new employees.

Anticipatory Socialization

Anticipatory socialization encompasses all of the learning that occurs prior to a recruit's entering on duty. At this stage, an individual forms expectations about the job and makes decisions about the suitability of fit between himself and the organization. What a person has heard about working for a particular organization, such as an intelligence agency, provides an idea of what to expect if hired. Conversely, individuals who do not believe they would fit in may decide not to apply.

There are two variables that are particularly useful for tracking a potential employee's progress through the anticipatory stage: The first is *realism*, or the extent to which an individual acquires an accurate picture of daily life in the organization. Realism is influenced by the level of success recruits achieve during the information-sharing and information-evaluation part of their recruitment. The second is *congruence*, or the extent to which the organization's resources and the individual's needs and skills are mutually satisfying. Congruence is influenced by the level of success an individual has achieved in making decisions about employment.

Although it cannot directly influence congruence, which is an inherently personal experience, an organization can present relevant information in order to provide a realistic and accurate description of the work performed and the work environment.

Organizations often use interviews to begin the socialization of new recruits. For example, an interviewer will attempt to provide an accurate description of what to expect from the job and the organization, the purpose being to reduce the likelihood that a recruit will be disturbed by unanticipated situations. Interviewing is also used to determine the degree to which there is a match between the values of potential recruits and the values of the organization. New recruits with personal values matching those of the organization have been found to adjust to the organization's culture more quickly than recruits with nonmatching values.

Organizations also send cultural messages to new recruits during interviews. When there are several rounds of interviews with progressively senior members of the organization, for example, the message conveyed is that finding the best person for the position is important. In contrast, hiring for a part-time job at the lowest level of the organization is often accomplished quickly, to the extent that a person having minimally acceptable qualifications may often be hired on the spot. The cultural message in this case is that such employees are easily let in to and out of the organization.

Another, particularly pertinent example is intelligence work, which requires that recruits undergo employment screenings unlike those found in most civilian jobs. Potential CIA analysts must submit to a thorough background investigation, a polygraph examination, and financial and credit reviews. Further, a battery of psychological and medical exams must be passed prior to a formal employment offer. The timeframe for the background check eliminates the possibility of a rapid hiring decision. Even more important are the nonverbal messages sent to the recruit that this is a position of secrecy and high importance.

Several sources of information contribute to beliefs about any organization. Friends or relatives who are already part of the organization might share their experiences with the person considering employment. Information might also be acquired from other sources, such as professional journals, magazines, newspaper articles, television, governmental and private Web sites, public statements or testimony, and annual reports. While these sources of information about an organization are far from

perfect (all may contain positive and negative hyperbole), they are still useful from the point of view of forming preliminary ideas about what it might be like to work for that organization.

Because competition for highly qualified employees is fierce, successful recruitment usually involves a skillful combination of salesmanship and diplomacy. Recruiters tend to describe their organizations in glowing terms, glossing over internal problems and external threats, while emphasizing positive features. The result is that potential employees often receive unrealistically positive impressions of conditions prevailing in a specific organization. When they arrive on the job and find that their expectations are not met, they experience disappointment, dissatisfaction, and even resentment that they have been misled. In fact, research findings indicate that the less employees' job expectations are met, the less satisfied and committed they are and the more likely they are to think about quitting or actually to do so.

These negative reactions are sometimes termed entry shock, referring to the confusion and disorientation experienced by many newcomers to an organization. In order to avoid entry shock, it is important for organizations to provide job candidates with accurate information about the organization. Research supports the notion that people exposed to realistic job previews later report higher satisfaction and show lower turnover than those who receive glowing, but often misleading, information about their companies. Moreover, having realistic expectations helps to ease the accommodation stage of the socialization process.

Consequences of Culture Mismatch

There are several consequences of a cultural mismatch between an employee and an organization. Among these consequences are culture shock, low job satisfaction, low employee morale, increased absenteeism, increased turnover, and increased costs.

Culture Shock. People often have to be confronted with different cultures before they become conscious of their own culture. In fact, when people are faced with new cultures, it is not unusual for them to become confused and disoriented, a phenomenon commonly referred to as culture shock.

Beryl Hesketh and Stephen Bochner, among others, have observed that the process of adjusting to another culture generally follows a U-shaped curve. At first, people are optimistic about learning a new culture. This excitement is followed by frustration and confusion as they struggle to learn the new culture. After six months or so with the organization,

people adjust to their new cultures, become more accepting of them, and are more satisfied by them. For those who enter a mismatched culture, the productivity issue is clear: the several months required to adjust and accept the new work style results in several months of even lower productivity than is obtainable with those who fit in right away.

Job Satisfaction. Job satisfaction is defined by one scholar as "people's positive or negative feelings about their jobs." It is hardly surprising that dissatisfied employees may try to find ways of reducing their exposure to their jobs. This is especially significant when one considers that people spend roughly one-third of their lives at work.

Interestingly, research suggests that the relationship between satisfaction and task performance, although positive, is not especially strong. Thus, while job satisfaction may be important to the longevity of any individual career cycle, it is not a major factor in individual job performance. It does, however, increase absenteeism, which has a negative effect on overall organizational productivity.

Absenteeism and Turnover. Research indicates that the lower an individual's job satisfaction, the more likely he or she is to be absent from work. As with job satisfaction and task performance, this relationship is modest but also statistically significant. An employee may even choose to leave an organization altogether. This voluntary resignation is measured as employee turnover and has fiscal consequences for both the individual and the organization.

Fiscal Cost. Employee turnover is a critical cost element. The expense of recruiting and training new employees, along with lost productivity from vacant positions and overtime pay for replacement workers, increases operating costs and also reduces employee organizational output.

A 2002 study by the Employment Policy Foundation found that the estimated turnover cost is $12,506 per year per full-time vacancy for the average employee with total compensation (wages and benefits) of $50,025. As the average annual turnover benchmark within the Fortune 500 is 23.8 percent, one can clearly see how critical it is for organizations to lessen the number of employees who leave voluntarily. Even unscheduled absences can be expensive—averaging between $247 and $534 per employee, per day, according to the same study.

Anticipatory Socialization in the Intelligence Community

Accepting a job with one of the 14 members of the Intelligence Community differs from other professions in that it is difficult for new

employees to have a clear and precise understanding of the roles and responsibilities they are about to assume. This is all the more pronounced because, for the most part, the Intelligence Community organizations lack a civilian counterpart.

Occasionally, the anticipatory socialization of people entering the intelligence analysis discipline will derive from accounts of current or former practitioners. More generally, however, a newcomer's initial impressions stem from the fictional media portrayals, which tend to emphasize the supposed glamour of operational tasks and pay little attention to the reality of research-based analytic work. The absence of hard knowledge about intelligence work is attributable, in part, to the organizational secrecy of the Intelligence Community and, in part, to the actual socialization process that occurs after one has been accepted for employment and has passed the required background investigation.

A newcomer's experience is often contrary to initial expectations. Employees are discouraged from talking about the specifics of their work outside of the organization or with those who have not been "cleared." On an individual level, this experience translates into professional culture shock and social isolation. Organizationally, an intentionally closed system of this kind has a number of potential performance-related consequences, among them perpetuation of the existing organizational culture by hiring familial legacies or those most likely to "fit in," job dissatisfaction, low morale and consequent reduction in employee readiness, increased employee turnover, greater likelihood of "groupthink," and strong internal resistance to organizational change.

Since the attacks of 11 September, the Intelligence Community has become more open about its role in government, its day-to-day working environment, and its employees' functions and responsibilities. While this openness is an extension of an ongoing trend towards public outreach—an example is the CIA's Officer-in-Residence programme established in 1985—the community has accelerated this trend towards openness in an effort to help the public, and its representatives, understand the missions and value of the Intelligence Community.

RECONFIGURABLE COMPUTING

Reconfigurable computing is a computer architecture combining some of the flexibility of software with the high performance of hardware by processing with very flexible high speed computing fabrics like field-

programmable gate arrays (FPGAs). The principal difference when compared to using ordinary microprocessors is the ability to make substantial changes to the datapath itself in addition to the control flow. On the other hand, the main difference with custom hardware, i.e. application-specific integrated circuits (ASICs) is the possibility to adapt the hardware during runtime by "loading" a new circuit on the reconfigurable fabric. The concept of reconfigurable computing has existed since the 1960s, when Gerald Estrin's paper proposed the concept of a computer made of a standard processor and an array of "reconfigurable" hardware. The main processor would control the behavior of the reconfigurable hardware. The latter would then be tailored to perform a specific task, such as image processing or pattern matching, as quickly as a dedicated piece of hardware. Once the task was done, the hardware could be adjusted to do some other task. This resulted in a hybrid computer structure combining the flexibility of software with the speed of hardware.

In the 1980s and 1990s there was a renaissance in this area of research with many proposed reconfigurable architectures developed in industry and academia, such as: Copacobana, Matrix, GARP, Elixent, NGEN, Polyp, MereGen, PACT XPP, Silicon Hive, Montium, Pleiades, Morphosys, and PiCoGA. Such designs were feasible due to the constant progress of silicon technology that let complex designs be implemented on one chip. Some of the these massively parallel reconfigurable computers were built primarily for special subdomains such as molecular evolution, neural or image processing. The world's first commercial reconfigurable computer, the Algotronix CHS2X4, was completed in 1991. It was not a commercial success, but was promising enough that Xilinx (the inventor of the Field-Programmable Gate Array, FPGA) bought the technology and hired the Algotronix staff. Later machines enabled first demonstrations of scientific principles, such as the spontaneous spatial self-organisation of genetic coding with MereGen.

Theories

Tredennick's Classification

The fundamental model of the reconfigurable computing machine paradigm, the data-stream-based anti machine is well illustrated by the differences to other machine paradigms that were introduced earlier, as shown by Nick Tredennick's following classification scheme of computing paradigms

Hartenstein's Xputer

Computer scientist Reiner Hartenstein describes reconfigurable computing in terms of an *anti-machine* that, according to him, represents a fundamental paradigm shift away from the more conventional von Neumann machine.

Hartenstein calls it Reconfigurable Computing Paradox, that software-to-configware (software-to-FPGA) migration results in reported speed-up factors of up to more than four orders of magnitude, as well as a reduction in electricity consumption by up to almost four orders of magnitude—although the technological parameters of FPGAs are behind the Gordon Moore curve by about four orders of magnitude, and the clock frequency is substantially lower than that of microprocessors. This paradox is partly explained by the Von Neumann syndrome.

High-performance Computing

High-Performance Reconfigurable Computing (HPRC) is a computer architecture combining reconfigurable computing-based accelerators like field-programmable gate array with CPUs or multi-core processors. The increase of logic in an FPGA has enabled larger and more complex algorithms to be programmed into the FPGA. The attachment of such an FPGA to a modern CPU over a high speed bus, like PCI express, has enabled the configurable logic to act more like a coprocessor rather than a peripheral. This has brought reconfigurable computing into the high-performance computing sphere.

Furthermore, by replicating an algorithm on an FPGA or the use of a multiplicity of FPGAs has enabled reconfigurable SIMD systems to be produced where several computational devices can concurrently operate on different data, which is highly parallel computing. This heterogeneous systems technique is used in computing research and especially in supercomputing. A 2008 paper reported speed-up factors of more than 4 orders of magnitude and energy saving factors by up to almost 4 orders of magnitude. Some supercomputer firms offer heterogeneous processing blocks including FPGAs as accelerators. One research area is the twin-paradigm programming tool flow productivity obtained for such heterogeneous systems.

The US National Science Foundation has a center for high-performance reconfigurable computing (CHREC). In April 2011 the fourth Many-core and Reconfigurable Supercomputing Conference was held in Europe.

Commercial high-performance reconfigurable computing systems are beginning to emerge with the announcement of IBM integrating FPGAs with its POWER processor.

Partial re-configuration

Partial re-configuration is the process of changing a portion of reconfigurable hardware circuitry while the other part is still running/ operating. Field programmable gate arrays are often used as a support to partial reconfiguration. Electronic hardware, like software, can be designed modularly, by creating subcomponents and then higher-level components to instantiate them. In many cases it is useful to be able to swap out one or several of these subcomponents while the FPGA is still operating.

Normally, reconfiguring an FPGA requires it to be held in reset while an external controller reloads a design onto it. Partial reconfiguration allows for critical parts of the design to continue operating while a controller either on the FPGA or off of it loads a partial design into a reconfigurable module. Partial reconfiguration also can be used to save space for multiple designs by only storing the partial designs that change between designs.

A common example for when partial reconfiguration would be useful is the case of a communication device. If the device is controlling multiple connections, some of which require encryption, it would be useful to be able to load different encryption cores without bringing the whole controller down.

Partial reconfiguration is not supported on all FPGAs. A special software flow with emphasis on modular design is required. Typically the design modules are built along well defined boundaries inside the FPGA that require the design to be specially mapped to the internal hardware.

From the functionality of the design, partial reconfiguration can be divided into two groups:

- *Dynamic partial reconfiguration*, also known as an active partial reconfiguration - permits to change the part of the device while the rest of an FPGA is still running;
- *Static partial reconfiguration* - the device is not active during the reconfiguration process. While the partial data is sent into the FPGA, the rest of the device is stopped (in the shutdown mode) and brought up after the configuration is completed.

Current systems

Computer Emulation

With the advent of affordable FPGA boards, students' and hobbyists' projects seek to recreate vintage computers or implement more novel architectures. Such projects are built with reconfigurable hardware (FPGAs), and some devices support emulation of multiple vintage computers using a single reconfigurable hardware (C-One).

COPACOBANA

A fully FPGA-based computer is the COPACOBANA, the Cost Optimized Codebreaker and Analyzer and its successor RIVYERA. A spin-off company SciEngines GmbH of the COPACOBANA-Project of the Universities of Bochum and Kiel in Germany continues the development of fully FPGA-based computers.

Mitrionics

Mitrionics has developed a SDK that enables software written using a single assignment language to be compiled and executed on FPGA-based computers. The Mitrion-C software language and Mitrion processor enable software developers to write and execute applications on FPGA-based computers in the same manner as with other computing technologies, such as graphical processing units ("GPUs"), cell-based processors, parallel processing units ("PPUs"), multi-core CPUs, and traditional single-core CPU clusters. (out of business)

National Instruments

National Instruments have developed a hybrid embedded computing system called CompactRIO. It consists of reconfigurable chassis housing the user-programmable FPGA, hot swappable I/O modules, real-time controller for deterministic communication and processing, and graphical LabVIEW software for rapid RT and FPGA programming.

Xilinx

Xilinx has developed two styles of partial reconfiguration of FPGA devices: *module-based* and *difference-based*. *Module-based partial reconfiguration* permits to reconfigure distinct modular parts of the design, while *difference-based partial reconfiguration* can be used when a small change is made to a design.

Intel

Intel supports partial reconfiguration of their FPGA devices on 28 nm devices such as Stratix V, and on the 20 nm Arria 10 devices. The Intel FPGA partial reconfiguration flow for Arria 10 is based on the hierarchical design methodology in the Quartus Prime Pro software where users create physical partitions of the FPGA that can be reconfigured at runtime while the remainder of the design continues to operate. The Quartus Prime Pro software also support hierarchical partial reconfiguration and simulation of partial reconfiguration.

Comparison of systems

As an emerging field, classifications of reconfigurable architectures are still being developed and refined as new architectures are developed; no unifying taxonomy has been suggested to date. However, several recurring parameters can be used to classify these systems.

Granularity

The granularity of the reconfigurable logic is defined as the size of the smallest functional unit (configurable logic block, CLB) that is addressed by the mapping tools. High granularity, which can also be known as fine-grained, often implies a greater flexibility when implementing algorithms into the hardware. However, there is a penalty associated with this in terms of increased power, area and delay due to greater quantity of routing required per computation. Fine-grained architectures work at the bit-level manipulation level; whilst coarse grained processing elements (reconfigurable datapath unit, rDPU) are better optimised for standard data path applications. One of the drawbacks of coarse grained architectures are that they tend to lose some of their utilisation and performance if they need to perform smaller computations than their granularity provides, for example for a one bit add on a four bit wide functional unit would waste three bits. This problem can be solved by having a coarse grain array (reconfigurable datapath array, rDPA) and a FPGA on the same chip.

Coarse-grained architectures (rDPA) are intended for the implementation for algorithms needing word-width data paths (rDPU). As their functional blocks are optimized for large computations and typically comprise word wide arithmetic logic units (ALU), they will perform these computations more quickly and with more power efficiency than a set of interconnected smaller functional units; this is due to the connecting wires

being shorter, resulting in less wire capacitance and hence faster and lower power designs. A potential undesirable consequence of having larger computational blocks is that when the size of operands may not match the algorithm an inefficient utilisation of resources can result. Often the type of applications to be run are known in advance allowing the logic, memory and routing resources to be tailored to enhance the performance of the device whilst still providing a certain level of flexibility for future adaptation. Examples of this are domain specific arrays aimed at gaining better performance in terms of power, area, throughput than their more generic finer grained FPGA cousins by reducing their flexibility.

Rate of Reconfiguration

Configuration of these reconfigurable systems can happen at deployment time, between execution phases or during execution. In a typical reconfigurable system, a bit stream is used to program the device at deployment time. Fine grained systems by their own nature require greater configuration time than more coarse-grained architectures due to more elements needing to be addressed and programmed. Therefore, more coarse-grained architectures gain from potential lower energy requirements, as less information is transferred and utilised. Intuitively, the slower the rate of reconfiguration the smaller the energy consumption as the associated energy cost of reconfiguration are amortised over a longer period of time. Partial re-configuration aims to allow part of the device to be reprogrammed while another part is still performing active computation. Partial re-configuration allows smaller reconfigurable bit streams thus not wasting energy on transmitting redundant information in the bit stream. Compression of the bit stream is possible but careful analysis is to be carried out to ensure that the energy saved by using smaller bit streams is not outweighed by the computation needed to decompress the data.

Host Coupling

Often the reconfigurable array is used as a processing accelerator attached to a host processor. The level of coupling determines the type of data transfers, latency, power, throughput and overheads involved when utilising the reconfigurable logic. Some of the most intuitive designs use a peripheral bus to provide a coprocessor like arrangement for the reconfigurable array. However, there have also been implementations where the reconfigurable fabric is much closer to the processor, some are even implemented into the data path, utilising the processor registers. The job

of the host processor is to perform the control functions, configure the logic, schedule data and to provide external interfacing.

Routing/interconnects

The flexibility in reconfigurable devices mainly comes from their routing interconnect. One style of interconnect made popular by FPGAs vendors, Xilinx and Altera are the island style layout, where blocks are arranged in an array with vertical and horizontal routing. A layout with inadequate routing may suffer from poor flexibility and resource utilisation, therefore providing limited performance. If too much interconnect is provided this requires more transistors than necessary and thus more silicon area, longer wires and more power consumption.

REACTION, PROACTION AND ANTICIPATION

Elementary forms of artificial intelligence can be constructed using a policy based on simple if-then rules. An example of such a system would be an agent following the rules

```
If it rains outside,
take the umbrella.
Otherwise
leave the umbrella home
```

A system such as the one defined above might be viewed as inherently reactive because the decision making is based on the current state of the environment with no explicit regard to the future. An agent employing anticipation would try to predict the future state of the environment (weather in this case) and make use of the predictions in the decision making. For example:

```
If the sky is cloudy and the air pressure is low,
it will probably rain soon
so take the umbrella with you.
Otherwise
leave the umbrella home.
```

These rules appear more proactive, because they explicitly take into account possible future events. Notice though that in terms of representation and reasoning, these two rule sets are identical, both behave in response to existing conditions. Note too that both systems assume the agent is proactively

```
leaving the house, and
trying to stay dry.
```

In practice, systems incorporating reactive planning tend to be autonomous systems proactively pursuing at least one, and often many, goals. What defines anticipation in an AI model is the explicit existence of an inner model of the environment for the anticipatory system (sometimes including the system itself). For example, if the phrase *it will probably rain* were computed on line in real time, the system would be seen as anticipatory.

In 1985, Robert Rosen defined an anticipatory system as follows:

A system containing a predictive model of itself and/or its environment, which allows it to change state at an instant in accord with the model's predictions pertaining to a later instant.

In Rosen's work, analysis of the example: "It's raining outside, therefore take the umbrella" does involve a prediction. It involves the prediction that "If it is raining, I will get wet out there unless I have my umbrella". In that sense, even though it is already raining outside, the decision to take an umbrella is not a purely reactive thing. It involves the use of predictive models which tell us what will happen if we don't take the umbrella, when it is already raining outside.

To some extent, Rosen's definition of anticipation applies to any system incorporating machine learning. At issue is how much of a system's behaviour should or indeed can be determined by reasoning over dedicated representations, how much by on-line planning, and how much must be provided by the system's designers.

ANTICIPATION IN EVOLUTION AND COGNITION

The anticipation of future states is also a major evolutionary and cognitive advance. Anticipatory agents belonging to Rosen's definition are easy to see in human mental capabilities of taking decisions at a certain time T taking into account the effects of their own actions at different future timescales T+k. However, Rosen (a theoretical biologist) describes ALL living organisms as examples of naturally occurring anticipatory systems, which means that there must be somatic predictive models (meaning, "of the body"; physical) as components within the organization of all living organisms. No mental process is required for anticipation. In his book, Anticipatory Systems, Rosen describes how even single cellular organisms manifest this behaviour pattern. It is logical to hypothesize therefore: If it is true that life is anticipatory in this sense, then the evolution of the conscious mind (such as human beings experience) may be a natural concentration and amplification of the anticipatory nature of life, itself.

Machine learning methods started to integrate anticipatory capabilities in an implicit form as in reinforcement learning systems where they learn to anticipate future rewards and punishments caused by current actions. Moreover, anticipation enhanced performance of machine learning techniques to face with complex environments where agents have to guide their attention to collect important information to act.

From Anticipation to Curiosity

Jürgen Schmidhuber modifies error back propagation algorithm to change neural network weights in order to decrease the mismatch between anticipated states and states actually experienced in the future. He introduces the concept of *curiosity* for agents as a measure of the mismatch between expectations and future experienced reality. Agents able to monitor and control their own curiosity explore situations where they expect to engage with novel experiences and are generally able to deal with complex environments more than the others.

5

Artificial Intelligence and Machine Learning in Autonomous Systems

CURRENT MACHINE LEARNING APPLICATIONS IN ROBOTICS

Computer Vision

Though related, some would argue the correct term is machine vision or robot vision rather than computer vision, because "robots seeing" involves more than just computer algorithms; engineers and roboticists also have to account for camera hardware that allow robots to process physical data. Robot vision is very closely linked to machine vision, which can be given credit for the emergence of robot guidance and automatic inspection systems.

The slight difference between the two may be in *kinematics* as applied to robot vision, which encompasses reference frame calibration and a robot's ability to physically affect its environment.

An influx of big data i.e. visual information available on the web (including annotated/labeled photos and videos) has propelled advances in computer vision, which in turn has helped further machine-learning based structured prediction learning techniques at universities like Carnegie Mellon and elsewhere, leading to robot vision applications like identification and sorting of objects.

One offshoot example of this is anomaly detection with unsupervised learning, such as building systems capable of finding and assessing faults

in silicon wafers using convolutional neural networks, as engineered by researchers at the Biomimetic Robotics and Machine Learning Lab, which is part of the nonprofit Assistenzrobotik e.V. in Munich.

Extrasensory technologies like radar, lidar, and ultrasound, like those from Nvidia, are also driving the development of 360-degree vision-based systems for autonomous vehicles and drones.

Imitation Learning

Imitation learning is closely related to observational learning, a behavior exhibited by infants and toddlers. Imitation learning is also an umbrella category for reinforcement learning, or the challenge of getting an agent to act in the world so as to maximize its rewards. *Bayesian or probabilistic models* are a common feature of this machine learning approach. The question of whether imitation learning could be used for humanoid-like robots was postulated as far back as 1999.

Imitation learning has become an integral part of field robotics, in which characteristics of mobility outside a factory setting in domains like domains like construction, agriculture, search and rescue, military, and others, make it challenging to manually program robotic solutions. Examples include inverse optimal control methods, or "programming by demonstration,"which has been applied by CMU and other organizations in the areas of humanoid robotics, legged locomotion, and off-road rough-terrain mobile navigators. Researchers from Arizona State published this video two years ago showing a humanoid robot using imitation learning to acquire different grasping techniques:

Bayesian belief networks have also been applied toward forward learning models, in which a robot learns without *a priori* knowledge of it motor system or the external environment. An example of this is "motor babbling", as demonstrated by the Language Acquisition and Robotics Group at University of Illinois at Urbana-Champaign (UIUC) with Bert, the "iCub" humanoid robot.

Self-Supervised Learning

Self-supervised learning approaches enable robots to generate their own training examples in order to improve performance; this includes using *a priori* training and data captured close range to interpret "long-range ambiguous sensor data." It's been incorporated into robots and optical devices that can detect and reject objects (dust and snow, for

example); identify vegetables and obstacles in rough terrain; and in 3D-scene analysis and modeling vehicle dynamics

Watch-Bot is a concrete example, created by researchers from Cornell and Stanford, that uses a 3D sensor (a Kinect), a camera, laptop and laser pointer to detect 'normal human activity', which are patterns that it learns through probabilistic methods.

Watch-Bot uses a laser pointer to target the object as a reminder (for example, the milk that was left out of the fridge). In initial tests, the bot was able to successfully remind humans 60 percent of time (it has no conception of what it's doing or why), and the researchers expanded trials by allowing its robot to learn from online videos (called project RoboWatch).

Other examples of self-supervised learning methods applied in robotics include a road detection algorithm in a front-view monocular camera with a road probabilistic distribution model (RPDM) and fuzzy support vector machines (FSVMs), designed at MIT for autonomous vehicles and other mobile on-road robots.

Autonomous learning, which is a variant of self-supervised learning involving deep learning and unsupervised methods, has also been applied to robot and control tasks. A team at Imperial College in London, collaborating with researchers from University of Cambridge and University of Washington, has created a new method for speeding up learning that incorporates model uncertainty (a probabilistic model) into long-term planning and controller learning, reducing the effect of model errors when learning a new skill.

Assistive and Medical Technologies

An assistive robot (according to Stanford's David L. Jaffe) is a device that can sense, process sensory information, and perform actions that benefit people with disabilities and seniors (though smart assistive technologies also exist for the general population, such as driver assistance tools).

Movement therapy robots provide a diagnostic or therapeutic benefit. Both of these are technologies that are largely (and unfortunately) still confined to the lab, as they're still cost-prohibitive for most hospitals in the U.S. and abroad.

Early examples of assistive technologies included the DeVAR, or desktop vocational assistant robot, developed in the early 1990s by Stanford and the Palo Alto Veterans Affairs Rehabilitation Research and

Development. More recent examples of machine learning-based robotic assistive technologies are being developed that include combining assistive machines with more autonomy, such as the MICO robotic arm (developed at Northwester University) that observes the world through a Kinect Sensor. The implications are more complex yet smarter assistive robots that adapt more readily to user needs but also require partial autonomy (i.e. a sharing of control between the robot and human). In the medical world, advances in machine learning methodologies applied to robotics are fast advancing, even though not readily available in many medical facilities. A collaboration through the Cal-MR: Center for Automation and Learning for Medical Robotics, between researchers at multiple universities and a network of physicians (collaborations with researchers at multiple universities and physicians led to the creation of the Smart Tissue Autonomous Robot (STAR), piloted through the Children's National Health System in DC.

Using innovations in autonomous learning and 3D sensing, STAR is able to stitch together "pig intestines" (used in lieu of human tissue) with better precision and reliability than the best human surgeons.

Researchers and physicians make the statement that STAR is not a replacement for surgeons – who for the foreseeable future would remain nearby to handle emergencies – but offer major benefits in performing similar types of delicate surgeries.

Multi-Agent Learning

Coordination and negotiation are key components of multi-agent learning, which involves machine learning-based robots (or agents – this technique has been widely applied to games) that are able to adapt to a shifting landscape of other robots/agents and find "equilibrium strategies."

Examples of multi-agent learning approaches include no-regret learning tools, which involve weighted algorithms that "boost" learning outcomes in multi-agent planning, and learning in market-based, distributed control systems.

A more concrete example is an algorithm for distributed agents or robots created by researchers from MIT's Lab for Information and Decision Systems in late 2014.

Robots collaborated to build a better and more inclusive learning model than could be done with one robot (smaller chunks of information processed and then combined), based on the concept of exploring a building

and its room layouts and autonomously building a knowledge base. Each robot built its own catalog, and combined with other robots' data sets, the distributed algorithm outperformed the standard algorithm in creating this knowledge base.

While not a perfect system, this type of machine learning approach allows robots to compare catalogs or data sets, reinforce mutual observations and correct omissions or over-generalizations, and will undoubtedly play a near-future role in several robotic applications, including multiple autonomous land and airborne vehicles.

Machine Learning in Robotics: Future Outlook – A Long Term Priority

The above,brief outline of machine-learning based approaches in robotics, combined with contracts and challenges put out by powerful military sponsors (e.g. DARPA, ARL); innovations by major robotics manufacturers (e.g. Silicon Valley Robotics) and start-up manufacturers (Mayfield Robotics); and increased investments by a barrage of auto-manufacturers (from Toyota to BMW) on a next generation of autonomous vehicles (to name a few influential domains), point to the trend of machine learning as a long-term priority.

ROBOTICS: ETHICS OF ARTIFICIAL INTELLIGENCE

The artificial intelligence (AI) and robotics communities face an important ethical decision: whether to support or oppose the development of lethal autonomous weapons systems (LAWS). Technologies have reached a point at which the deployment of such systems is — practically if not legally — feasible within years, not decades. The stakes are high: LAWS have been described as the third revolution in warfare, after gunpowder and nuclear arms.

Autonomous weapons systems select and engage targets without human intervention; they become lethal when those targets include humans. LAWS might include, for example, armed quadcopters that can search for and eliminate enemy combatants in a city, but do not include cruise missiles or remotely piloted drones for which humans make all targeting decisions.

Existing AI and robotics components can provide physical platforms, perception, motor control, navigation, mapping, tactical decision-making and long-term planning. They just need to be combined. For example, the

technology already demonstrated for self-driving cars, together with the human-like tactical control learned by DeepMind's DQN system, could support urban search-and-destroy missions.

Two US Defense Advanced Research Projects Agency (DARPA) programmes foreshadow planned uses of LAWS: Fast Lightweight Autonomy (FLA) and Collaborative Operations in Denied Environment (CODE). The FLA project will program tiny rotorcraft to manoeuvre unaided at high speed in urban areas and inside buildings. CODE aims to develop teams of autonomous aerial vehicles carrying out "all steps of a strike mission — find, fix, track, target, engage, assess" in situations in which enemy signal-jamming makes communication with a human commander impossible. Other countries may be pursuing clandestine programmes with similar goals.

International humanitarian law — which governs attacks on humans in times of war — has no specific provisions for such autonomy, but may still be applicable. The 1949 Geneva Convention on humane conduct in war requires any attack to satisfy three criteria: military necessity; discrimination between combatants and non-combatants; and proportionality between the value of the military objective and the potential for collateral damage. (Also relevant is the Martens Clause, added in 1977, which bans weapons that violate the "principles of humanity and the dictates of public conscience.") These are subjective judgments that are difficult or impossible for current AI systems to satisfy.

The United Nations has held a series of meetings on LAWS under the auspices of the Convention on Certain Conventional Weapons (CCW) in Geneva, Switzerland. Within a few years, the process could result in an international treaty limiting or banning autonomous weapons, as happened with blinding laser weapons in 1995; or it could leave in place the status quo, leading inevitably to an arms race.

As an AI specialist, I was asked to provide expert testimony for the third major meeting under the CCW, held in April, and heard the statements made by nations and non-governmental organizations. Several countries pressed for an immediate ban. Germany said that it "will not accept that the decision over life and death is taken solely by an autonomous system"; Japan stated that it "has no plan to develop robots with humans out of the loop, which may be capable of committing murder".

The United States, the United Kingdom and Israel — the three countries leading the development of LAWS technology — suggested that a treaty

is unnecessary because they already have internal weapons review processes that ensure compliance with international law. Almost all states who are party to the CCW agree with the need for 'meaningful human control' over the targeting and engagement decisions made by robotic weapons. Unfortunately, the meaning of 'meaningful' is still to be determined.

The debate has many facets. Some argue that the superior effectiveness and selectivity of autonomous weapons can minimize civilian casualties by targeting only combatants. Others insist that LAWS will lower the threshold for going to war by making it possible to attack an enemy while incurring no immediate risk; or that they will enable terrorists and non-state-aligned combatants to inflict catastrophic damage on civilian populations.

LAWS could violate fundamental principles of human dignity by allowing machines to choose whom to kill — for example, they might be tasked to eliminate anyone exhibiting 'threatening behaviour'. The potential for LAWS technologies to bleed over into peacetime policing functions is evident to human-rights organizations and drone manufacturers.

In my view, the overriding concern should be the probable endpoint of this technological trajectory. The capabilities of autonomous weapons will be limited more by the laws of physics — for example, by constraints on range, speed and payload — than by any deficiencies in the AI systems that control them.

For instance, as flying robots become smaller, their manoeuvrability increases and their ability to be targeted decreases.

They have a shorter range, yet they must be large enough to carry a lethal payload — perhaps a one-gram shaped charge to puncture the human cranium. Despite the limits imposed by physics, one can expect platforms deployed in the millions, the agility and lethality of which will leave humans utterly defenceless. This is not a desirable future.

The AI and robotics science communities, represented by their professional societies, are obliged to take a position, just as physicists have done on the use of nuclear weapons, chemists on the use of chemical agents and biologists on the use of disease agents in warfare. Debates should be organized at scientific meetings; arguments studied by ethics committees; position papers written for society publications; and votes taken by society members. Doing nothing is a vote in favour of continued development and deployment.

Sabine Hauert: Shape the Debate

Irked by hyped headlines that foster fear or overinflate expectations of robotics and artificial intelligence (AI), some researchers have stopped communicating with the media or the public altogether. But we must not disengage. The public includes taxpayers, policy-makers, investors and those who could benefit from the technology. They hear a mostly one-sided discussion that leaves them worried that robots will take their jobs, fearful that AI poses an existential threat, and wondering whether laws should be passed to keep hypothetical technology 'under control'. My colleagues and I spend dinner parties explaining that we are not evil but instead have been working for years to develop systems that could help the elderly, improve health care, make jobs safer and more efficient, and allow us to explore space or beneath the oceans.

Experts need to become the messengers. Through social media, researchers have a public platform that they should use to drive a balanced discussion. We can talk about the latest developments and limitations, provide the big picture and demystify the technology. I have used social media to crowd-source designs for swarming nanobots to treat cancer. And I found my first PhD student through his nanomedicine blog.

The AI and robotics community needs thought leaders who can engage with prominent commentators such as physicist Stephen Hawking and entrepreneur–inventor Elon Musk and set the agenda at international meetings such as the World Economic Forum in Davos, Switzerland. Public engagement also drives funding. Crowdfunding for JIBO, a personal robot for the home developed by Cynthia Breazeal, at the Massachusetts Institute of Technology (MIT) in Cambridge, raised more than US$2.2 million.

There are hurdles. First, many researchers have never tweeted, blogged or made a YouTube video. Second, outreach is 'yet another thing to do', and time is limited. Third, it can take years to build a social-media following that makes the effort worthwhile. And fourth, engagement work is rarely valued in research assessments, or regarded seriously by tenure committees.

Training, support and incentives are needed. All three are provided by Robohub.org, of which I am co-founder and president. Launched in 2012, Robohub is dedicated to connecting the robotics community to the public. We provide crash courses in science communication at major AI and robotics conferences on how to use social media efficiently and effectively. We invite professional science communicators and journalists to help researchers to prepare an article about their work. The

communicators explain how to shape messages to make them clear and concise and avoid pitfalls, but we make sure the researcher drives the story and controls the end result. We also bring video cameras and ask researchers who are presenting at conferences to pitch their work to the public in five minutes. The results are uploaded to YouTube. We have built a portal for disseminating blogs and tweets, amplifying their reach to tens of thousands of followers.

I can list all the benefits of science communication, but the incentive must come from funding agencies and institutes. Citations cannot be the only measure of success for grants and academic progression; we must also value shares, views, comments or likes. MIT robotics researcher Rodney Brooks's classic 1986 paper on the 'subsumption architecture', a bio-inspired way to program robots to react to their environment, gathered nearly 10,000 citations in 30 years. A video of Sawyer, a robot developed by Brooks's company Rethink Robotics, received more than 60,000 views in one month. Which has had more impact on today's public discourse?

Governments, research institutes, business-development agencies, and research and industry associations do welcome and fund outreach and science-communication efforts. But each project develops its own strategy, resulting in pockets of communication that have little reach.

In my view, AI and robotics stakeholders worldwide should pool a small portion of their budgets (say 0.1%) to bring together these disjointed communications and enable the field to speak more loudly. Special-interest groups, such as the Small Unmanned Aerial Vehicles Coalition that is promoting a US market for commercial drones, are pushing the interests of major corporations to regulators. There are few concerted efforts to promote robotics and AI research in the public sphere. This balance is badly needed.

A common communications strategy will empower a new generation of roboticists that is deeply connected to the public and able to hold its own in discussions. This is essential if we are to counter media hype and prevent misconceptions from driving perception, policy and funding decisions.

ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

The ever-expanding field of Artificial Intelligence stands upon the precipice of a mainstream breakthrough. Whether AI-enhanced smartphones whip up the public frenzy or driverless cars get there first,

it's clear that we're officially in the AI era. Naysayers will point out AI isn't new; researchers were diving into the idea of autonomous computing back in the 1950s. Today's developers aren't so different either, as what they're doing is essentially what experts in the field have been working on for decades.

What's changed is the raw computing power we have available now. Fifty years ago, scientists would have needed computers the size of Nevada to do what we today can do on chips the size of pennies. Perhaps clever architecture could have gotten it down to the size of a shopping mall, but you get the point.

As far as hardware is concerned we've arrived, and so have the robots.

But what does it all mean? Defining the nature of what AI is, and what it's going to do for Joe Public, is difficult. Advances that will affect the entire world are often complex and take awhile before everyone understands what's happening. Remember trying to explain the internet to people in the 90s? There was a time, not all that long ago, when words like "bandwidth" and "router" weren't common in the lexicon of your average person. In the next few years everyone is going to want to understand some basic terms about AI, because you'll be seeing it all over the place as every gadget made in the near-future is going to have some form of artificial intelligence baked in.

Artificial intelligence

The first thing we need to do is understand what an AI actually is. The term "artificial intelligence" refers to a specific field of computer engineering that focuses on creating systems capable of gathering data and making decisions and/or solving problems.

An example of basic AI is a computer that can take 1000 photos of cats for input, determine what makes them similar, and then find photos of cats on the internet. The computer has learned, as best as it can, what a photo of a cat looks like and uses this new intelligence to find things that are similar.

Autonomous

Simply put, autonomy means that an AI construct doesn't need help from people. Driverless cars illustrate the term "autonomous" in varying degrees. Level four autonomy represents a vehicle that doesn't need a steering wheel or pedals: it doesn't need a human inside of it to operate

at full capacity. If we ever have a vehicle that can operate without a driver, and also doesn't need to connect to any grid, server, GPS, or other external source in order to function it'll have reached level five autonomy.

Anything beyond that would be called sentient, and despite the leaps that have been made recently in the field of AI, the singularity (an event representing an AI that becomes self-aware) is purely theoretical at this point.

Algorithm

The most important part of AI is the algorithm. These are math formulas and/or programming commands that inform a regular non-intelligent computer on how to solve problems with artificial intelligence. Algorithms are rules that teach computers how to figure things out on their own. It may be a nerdy construct of numbers and commands, but what algorithms lack in sex appeal they more than make up for in usefulness.

MACHINE LEARNING AND APPLICATIONS

It's likely that you've interacted with some form of AI in your day-to-day activities. If you use Gmail, for example, you may enjoy the automatic e-mail filtering feature. If you own a smartphone, you probably fill out a calendar with the help of Siri, Cortana, or Bixby. If you own a newer vehicle, perhaps you've benefited from a driver-assist feature while driving. As helpful as these software products are, they lack the ability to learn independently. They cannot think outside their code. Machine learning is a branch of AI that aims to give machines the ability to learn a task without pre-existing code.

In the simplest terms, machines are given a large amount of trial examples for a certain task. As they go through these trials, machines learn and adapt their strategy to achieve those goals.

For example, an image-recognition machine may be given millions of pictures to analyze. After going through endless permutations, the machine acquires the ability to recognize patterns, shapes, faces, and more.

A well-known example of this AI concept is Quick, Draw!, a Google-hosted game that lets humans draw simple pictures in under 20 seconds, with the machine-learning algorithm trying to guess the drawing. More than 15 million people have contributed more than 50 million drawings to the app.

Deep learning gets ready to play

How do we get machines to learn more than just a specific task? What if we want it to be able to take what it has learned from analyzing photographs and use that knowledge to analyze different data sets? This requires computer scientists to formulate general-purpose learning algorithms that help machines learn more than just one task.

One famous example of deep learning in action is Google's AlphaGo project written in Lua, C++, and Python code. The AlphaGo AI was able to beat professional Go players, a feat that was thought impossible given the game's incredible complexity and reliance on focused practice and human intuition to master.

How was a program able to master a game that calls for human intuition? Practice, practice, practice — and a little help from an artificial neural network.

Neural networks follow natural model

Deep learning is often made possible by artificial neural networks, which imitate neurons, or brain cells. Artificial neural networks were inspired by things we find in our own biology. The neural net models use math and computer science principles to mimic the processes of the human brain, allowing for more general learning.

An artificial neural network tries to simulate the processes of densely interconnected brain cells, but instead of being built from biology, these neurons, or nodes, are built from code.

Neural networks contain three layers: an input layer, a hidden layer and an output layer. These layers contain thousands, sometimes millions, of nodes. Information is fed into the input layer. Inputs are given a certain weight, and interconnected nodes multiply the weight of the connection as they travel.

Essentially, if the unit of information reaches a certain threshold, then it is able to pass to the next layer. In order to learn from experience, machines compare outputs from a neural network, then modify connections, weights, and thresholds based on the differences among them.

MACHINE LEARNING IN ROBOTICS IN MODERN APPLICATIONS

As the term "machine learning" has heated up, interest in "robotics" (as expressed in Google Trends) has not altered much over the last three

years. So how much of a place is there for machine learning in robotics?

While only a portion of recent developments in robotics can be credited to developments and uses of machine learning, I've aimed to collect some of the more prominent applications together in this article, along with links and references.

Before I delve into machine learning in robotics, go ahead and define "robot". Though at first this might seem simple, it's no easy task to come to an agreement on just what a robot *is* and what it *is not*, even amongst roboticists. For the sake of this article, I'll borrow an abbreviated definition of "robot" from this chapter on the Carnegie Mellon CS Department website:

"Force through intelligence."

or

"Where AI meets the real world."

Some researchers might even argue against a set definition for robot, or debate whether a definition can be relative or dependent upon the context of a situation, such as the concept of "privacy"; this might be a better approach as more and more rules and regulations are created around their use in varying contexts. There's also some debate as to whether the term robot includes innovations such as autonomous vehicles, drones, and other similar machines. For the purposes of this chapter, and considering the definition above, I argue that these types of machines are a class of mobile robot. Most robots are not, and will likely not, be humanoids 10 years from now; as robots are designed for a range of behaviors in a plethora of environments, their bodies and physical abilities will reflect a best fit for those characteristics.

An exception will likely be robots that provide medical or other care or companionship for humans, and perhaps service robots that are meant to establish a more personal and 'humanized' relationship.

Like many innovative technological fields today, robotics has and is being influenced and in some directions steered by machine learning technologies. According to a recent survey published by the Evans Data Corporation Global Development, machine learning and robotics is at the top of developers' priorities for 2016, with 56.4 percent of participants stating that they're building robotics apps and 24.7 percent of all developers indicating the use of machine learning in their projects.

The following overview of machine learning applications in robotics highlights five key areas where machine learning has had a significant

impact on robotic technologies, both at present and in the development stages for future uses. Though by no means inclusive, the purpose of the summary is to give readers a taste for the types of machine learning applications that exist in robotics and stimulate the desire for further research in these and other areas.

ARTIFICIAL INTELLIGENCE (AI) IN ROBOTICS: MACHINE LEARNING

My area focuses on the specific applications of robotics to extreme and challenging environments. These include robots that handle nuclear waste, climb tall towers in the middle of the ocean or survive thousands of meters underwater so not vacuuming the floor or serving you a coffee.

AI is cross cutting

AI forms a part of this programme, not because of the current hype around the technology, but simply that some of the latest developments in machine learning are so well suited to robotic challenges in unstructured environments.

THE INTELLIGENCE COMMUNITY'S NEGLECT OF STRATEGIC INTELLIGENCE

Commonly misunderstood, we neglect it at our peril. The architects of the National Security Act of 1947 would be greatly surprised by today's neglect of strategic intelligence in the Intelligence Community.

This year marks the 60th anniversary of the National Security Act of 1947. So many of our most prominent government institutions were created by this act—the National Security Council (NSC), the Armed Forces as a joint establishment, the US Air Force, and, of course, the Central Intelligence Agency (CIA). As a "living" document, the act has outlasted the Cold War, for which it was devised, and much more.

By the 1980s the act's architects had passed away. Their thoroughness was such, however, that amendments have not radically altered what they essentially put in place. One relatively recent change, the Goldwater-Nichols Act of 1986, in addition to its impact on the interrelationships of the service arms, notably also mandated the creation of an annual National Security Strategy, a document produced by the president and reported annually to the Congress.

The original architects, with World War II in recent memory, knew very well the importance of giving commanders enough authority, and they likewise knew the importance of strategy. By 1947 George Kennan had wired his now famous Long Telegram. In March 1947, President Harry Truman announced what we now call the Truman Doctrine, and so initiated America's national (grand) strategy of Communist Containment. Today, decades later, a national strategy is not only advisable for the republic but legally required. One can almost hear the original architects asking themselves, *Why didn't we think of that*? But as much as the security act's architects would have approved of a published national strategy, they would, I believe, be greatly surprised, perhaps even incensed, by today's neglect of strategic intelligence in the Intelligence Community. Strategic intelligence collection and analysis is a capability they took pains to preserve; we are perilously close to losing it. The reasons are complicated, but they deserve our examination and discussion in this anniversary year.

Does Anyone Know What Strategic Intelligence Is?

Readers can easily get a sense of the problem by conducting a small, admittedly unscientific, survey. Hand someone a report on a foreign-related topic and describe it as "strategic intelligence." Then ask the recipient to explain the term "strategic intelligence" and how the report qualifies. In my own surveys, a typical reply, after an awkward pause, has been that strategic intelligence is information about countries, or about strategic nuclear forces, or perhaps a long-range forecast. Another common reply, commendable in its honesty, has been "I don't know."

Substantively, none of these answers is adequate—and they are downright odd when compared to the straightforward answers many of us would give when asked to define tactical intelligence. These might include something like "intelligence information for the tactical battlefield." Logically enough, the official definition the Pentagon uses is equally straightforward: "Intelligence that is required for planning and conducting tactical operations."

This is the Pentagon's official definition of strategic intelligence:

Intelligence that is required for the formulation of strategy, policy, and military plans and operations at national and theater levels.

Or, in fewer words, strategic intelligence is that intelligence necessary to create and implement a strategy, typically a grand strategy, what officialdom calls a national strategy. A strategy is not really a plan but the logic driving a plan.

A strategy furthers one's advance towards goals by suggesting ways to accommodate and/or orchestrate a variety of variables—sometimes too many for the strategist alone to anticipate and understand. When foreign areas are involved, in-depth expertise is required, which is what strategic intelligence provides. Without the insights of deep expertise—insights based on detailed knowledge of obstacles and opportunities and enemies and friends in a foreign area—a strategy is not much more than an abstract theory, potentially even a flight of fancy. The better the strategic intelligence, the better the strategy, which is why the definition of strategic intelligence should not be so mysterious. Nevertheless, in official circles and beyond, too many people attribute meanings to "strategic" and "strategic intelligence" that no dictionary supports. Ignorance of the meaning of these words has bred ignorance of the strategic product, with, in my view, enormous consequences.

During the past decade and a half, since the Cold War, the production and use of strategic intelligence by the United States government has plunged to egregiously low levels. This decline is badly out of sync with the broader needs of the republic, fails to meet the nation's foreign policy requirements, ill-serves the country's many national security officials, and retards the developing prowess of its intelligence analysts.

This neglect is not only perilous, it is tragic. American ingenuity has made great contributions to the ancient craft of intelligence, contributions worthy of national pride. The most famous is the American spy satellite, a Cold War invention. Less famous but just as ingenious is multi-departmental strategic intelligence, invented during World War II by the Office of Strategic Services (OSS).

Yet, within the government that created it and that was once its master artisan, this analytical invention is now largely neglected. As my informal surveys suggest, very few employees of the Intelligence Community would say they are working to advance the implementation of the official National Security Strategy—or indeed, any strategy. Instead, much of today's intelligence is tactical, tangential, or tied to national strategy only by formal references to high-level strategic planning or guidance documents in forewords, prefaces, or other such administrative front-matter.

Who's Thinking About Tomorrow?

From my perspective, it's not clear anyone is, or will be—at least not as long as the analyst's primary product is current intelligence, which in essence is only the daily news compiled with secret information.

This type of intelligence must be desirable since so many consumers do consume it, but, like journalism without investigative reporting, it is not strategic intelligence and cannot replace it. As a percentage of the community's workload, however, it nearly has. In a survey of hundreds of community analysts performed by a fellow at CIA's Center for the Study of Intelligence (CSI) about two years ago, these complaints were heard:

Our products have become so specific, so tactical even, that our thinking has become tactical. We're losing our strategic edge because we're so focused on today's issues.

About 15 years ago, I used to have 60 percent of my time available for long-term products. Now, it's between 20 and 25 percent.

[V]elocity isn't a substitute for quality. We've gotten rid of the real analytic products that we use to make, and now we just report on current events.

Many of the community's elders likewise lament the consequences of a national intelligence effort now so focused upon the immediate:

The Intelligence Community really focused on current intelligence, on policy support. It does very little research. It has very little understanding below the level of the policymaker and, in my view, on many issues. I think that, in some ways, these two groups are reinforcing each other's worst habits.

A lot of strategic intelligence is not secret. It's out there. You'd better have some people who understand history. Instead, they've gotten sucked into the current intelligence business, which is death. It's death to knowing what's going on.

Is American...strategic intelligence up to the demands of the global environment and our national policies and strategies? I think there is a prima facie case that the answer is no.

Summarizing their concern is this excerpt from the CSI-published conference report from which the preceding comment was drawn:

A major [community] weakness...is its difficulty in providing strategic intelligence—the comprehensive overviews that put disparate events and the fragmentary snapshots provided by different intelligence sources into a contextual framework that makes it meaningful for the intelligence consumer. This criticism applies to intelligence prepared both for a national policy audience and for more specialized audiences, such as battlefield commanders.

Some supervisors argue that the community is doing more strategic intelligence work than is generally reported. Perhaps. But the excerpt above hints at a deeper, more insidious problem: It describes strategic

intelligence as the provision of context. Context is nice, sometimes even helpful, but it does not compellingly excite the average consumer, especially the military one, because it is not strategic support. Yet "context" is what most analysts and consumers assume strategic intelligence is.

Another common assumption is that strategic intelligence is merely a longer range perspective. Officialdom even promotes this, if unwittingly. For example, in the National Defense Intelligence College, a component of the Defense Intelligence Agency, is the Center for Strategic Intelligence Research (CSIR). The center describes itself as "the Intelligence Community's research and publication center devoted to an impartial exploration of medium- and long-range issues of concern to intelligence directors "

Where in that description, however, is there any allusion to national strategy? Or does strategic intelligence exist in a realm without strategy? Should it? At the risk of waxing nostalgic about the Cold War, in that era many policymakers were voracious consumers of strategic intelligence because it did provide strategic support. Used to tailor the grand strategy of communist containment, it deeply assessed the threats the United States and its allies faced, articulated their strengths and weaknesses, and noted exploitable opportunities. It was "current" in that it was timely, but it was also strategic. Directly applicable to the national strategy, it was, in today's terminology, "actionable" intelligence.

At present, about one half of the community's analysts possess less than five years of experience.8 Strategic intelligence is not their forte; few would have learned it in college and most have not had enough practice to gain sufficient understanding and expertise to produce strategic intelligence. As intelligence agencies swell their ranks with more and more new analysts, this situation is unlikely to improve anytime soon.

At CIA in particular, General Michael Hayden told Congress last year that for every 10 CIA analysts with less than four years of experience, only one analyst has more than 10 years of experience. "This is the least experienced analytic workforce in the history of the Central Intelligence Agency," he said. One result, warned Carl W. Ford Jr., a former assistant secretary of state for intelligence and research, is that "we haven't done strategic intelligence for so long that most of our analysts don't know how to do it anymore."

Another reason strategic intelligence "isn't done" is that among today's intelligence consumers, urgency is pushing tactical thinking. *To stop terrorists, I need this specific piece of tactical intelligence—right now.* Consequently, by

default, those analytical topics that feel somehow too grand, or too distant in time and place to matter immediately, tend to get ignored.

In fairness to intelligence analysts and their managers, they are merely following standard procedure, performing compartmentalized, narrowly focused routines. But reality is not entirely amenable to compartmentalization. Reality is inter-related and messy, involving deadly diseases from AIDS to avian flu; politically disruptive environmental changes; demographic dislocation; endemic corruption; trafficking in everything from people to weapons of mass destruction (WMD); intolerant belief-systems; genocide; shifting centers of economic power; global energy competition; and engineering breakthroughs from bio-manipulation to nano technology. These challenges are so profoundly complex, they cannot be well explained only in current or tactical intelligence. Even if analysts are doing the reporting, reporting the facts de jour is not analysis. At the other extreme, analysis should not exist for its own sake, as though any interpretation of facts is better than none at all. Producing token interpretations, day after day, may keep an analyst employed, but as analytical practice this is only "make work" activity. More often than not it just dulls an analyst's proficiency while the consumer gets a flow of pseudo-analytic drivel. Effective analysis ought to enhance a product until it empowers a consumer with the maximum advantage an expert's insight can provide. That is actionable intelligence.

At the Creation

Many a reader of *Studies in Intelligence* knows the contributions of Sherman Kent, including his book *Strategic Intelligence and American World Policy*, published in 1949. But to understand from whence modern strategic intelligence originated and where we stand today, we need to look back to World War II, to the work of the Research and Analysis (R&A) branch of the OSS.

At the time, the R&A products that most impressed the US military were infrastructure studies. In 1942, as American forces prepared to invade North Africa, a young Kent at R&A supervised the creation of several studies of that region's ports and railways. Showing vast detail, those studies amazed their military consumers. R&A found most of the raw information quite openly in books, trade journals, statistical abstracts and almanacs, even in the archived project files of cooperative private companies. Kent and his colleagues—all practiced scholars supported by

the full resources of the Library of Congress—knew where to find good information.

Today, by contrast, the typical intelligence analyst rarely exploits open sources as well. Working in environments dominated by secrecy and security concerns, most analysts work in relative seclusion. As a result, compared to an experienced professor or a seasoned business researcher—both proficient at exploiting open sources deeply—most entry-level analysts are novices.

Accurate, detailed information is not necessarily available via the Internet, nor is it always free. Far more exists off the Internet, but the daily deadlines of current intelligence discourage its deep exploitation. So, for reasons of ease, speed, and perhaps a little arrogance, most community analysts confine their raw material to secret information. Secret information may be very good, but information need not be secret to be accurate. And, as we know from the experience of Iraqi WMD, secret information is not necessarily always accurate.

Back to R&A. In 1943, it subjected its famed infrastructure studies to military-economic analysis and, in so doing, invented multi-departmental strategic intelligence. This excerpt from a CIA-published history of the OSS summarizes that phenomenal achievement:

Analyses by the Enemy Objectives Unit (EOU), a team of R&A economists posted to the U.S. Embassy in London, sent Allied bombers toward German fighter aircraft factories in 1943 and early 1944. After the Luftwaffe's interceptor force was weakened, Allied bombers could strike German oil production, which EOU identified as the choke-point in the Nazi war effort. The idea was not original with OSS, but R&A's well-documented support gave it credibility and helped convince Allied commanders to try it.... The resulting scarcity of aviation fuel all but grounded Hitler's Luftwaffe and, by the end of [1944], diesel and gasoline production had also plummeted, immobilizing thousands of German tanks and trucks.

A great success. Imagine if R&A's infrastructure studies had not existed or were produced in haste by amateurs ignorant of the best sources, the results either inaccurate or incomplete. The actual studies were good, of course, but they might have remained strictly tactical intelligence tools, as tactical as a sergeant's field map, nothing strategic. Imagine if nobody had bothered to think any harder, too cautious or too busy to consider, let alone attempt, a thoroughly multi-disciplinary analysis in the hope of creating a decisive advantage. Good information abounded, but information on paper is not necessarily knowledge in an analyst's mind, and therefore

not necessarily incorporated into that analyst's impressions and analyses.

Which brings us to the EOU's economists. Quite young, they could have been derided as "a bunch of silly economists ignorant of real war." They did have advanced university degrees and did represent the OSS, but what made them insightful, persuasive, and ultimately successful is what they knew as individuals. They knew what they were talking about, and it showed. Their thorough study of the multi-disciplinary material they accumulated made them true subject-matter experts. In the process, they created a new intelligence discipline whose tradecraft transforms vast amounts of scattered information into an individual's comprehensive knowledge and, ultimately, into exceptional insight. The respect they received, they earned.

The OSS did not survive the postwar demobilization of late 1945, but R&A did. Initially transferred to the State Department, it went to CIA because the strategic intelligence capability it embodied was understood to be essential to the national security, whether in war or peace.

Preserving that capability was one of the objectives the architects of the National Security Act of 1947 had in mind. Although the term "strategic intelligence" does not appear, for that term was not yet commonly used among civilians, the act did call for the continuous production of "national intelligence," a category the act treats as distinctly different from tactical intelligence.

National intelligence, according to the act, was to be produced by the Intelligence Community under the Director of Central Intelligence (DCI), now the DNI. Tactical intelligence was to be the job of the military services, perhaps not without Intelligence Community help, but that help was not to be the community's main effort. The original architects of the act knew the mission of producing national (strategic) intelligence would be daunting, which is why they created a central agency, CIA, to not only receive and coordinate the government's intelligence information but, crucially, undertake multi-disciplinary analysis (an endeavor more comprehensive than "all-source analysis") to achieve the great successes R&A had achieved in World War II.

Little wonder, then, that so many veterans of the old R&A, like Sherman Kent, were recruited into the new CIA.

Informative or Ivory Tower?

"Let things be such," Kent advised during the Cold War, "that if our

policymaking master is to disregard our knowledge and wisdom, he will never do so because our work was inaccurate, incomplete, or patently biased."

Every good analyst knows the importance of objectivity. By following evidence-based logic, an objective analysis holds the potential to debunk a policymaker's preconceptions, even reveal how his preferred policy actually fails. What keeps the policymaker receptive to such analysis, despite the bad news it may contain, is its claim to objectivity.

The analyst's need to be objective and his need to know which topics most interest a policymaker (or other consumer) have posed a dilemma that has been much discussed in these pages and in the literature of intelligence in general. Kent himself rated the risk that analysts would be contaminated by consumers a greater danger than the risk posed by self-imposed isolation.

As a result, the CIA's analytical components tended to be isolated and at times seemed out of touch with their consumers. Because so much intelligence work is secretive anyway, the isolation would have felt normal. The Cold War itself reinforced the isolation by requiring little daily interaction between analysts and consumers, the Cuban Missile Crisis being a rare exception.

More typically, the president and other senior officials received daily intelligence briefings, delivered by a briefer (not an analyst) or as a document. Thereafter, those officials would seldom see or speak with an intelligence officer until the next morning's briefing.

That arrangement worked throughout the Cold War because most policymakers knew which countries mattered and knew a lot about them. Every US president from Kennedy to George H. W. Bush witnessed the opening of the Cold War as adults and learned the dynamics of the containment strategy and the key countries in the game. The Cold War dominated current events, university discussions, and, of course, military planning. With decades of experience, each president would find the Intelligence Community effort to be additional to their own efforts and thus only supplemental, albeit crucially so.

In the military as well, limited interaction prevailed. Behind their salutes and outward camaraderie, many intelligence and operations personnel were actually a little suspicious of each other, mutually afraid of security leaks. Contingency war planning was considered so sensitive that intelligence people, ostensibly supporting the operators, were told

remarkably few specifics by those very operators devising the plans. This left many analysts with time to hone their craft. Consider what they had to learn: In strategic intelligence especially, though not exclusively, every issue involves multiple disciplines: politics, economics, organizational behavior, infrastructure studies (terrain, transportation, telecommunications), engineering, and military science (ground, naval, air, space, nuclear, unconventional).

Cultural awareness is imperative, which means knowing more than just some stereotypes. Every ethnicity, religion, and organization has a culture, usually several, their diversity and dynamics revealed only through study. Another analytical skill is to see events in true proportion, using historical experience to investigate across time and distance. An obscure event may possess more lasting significance than today's headline story—the former brewing as a future crisis, the latter likely to be forgotten within days. Intertwined with analysis is communicating it.

This can be remarkably difficult because many habits of conversation tend to be remarkably sloppy. *Well, everybody knows what I really mean!* Little better are many habits of writing.

In 1953, decades before instant e-mail rendered a quick spurt of typing preferable to a carefully crafted essay, Kent expressed his "sense of outrage at the infantile imprecision of the language" being used even then. To craft language which is literal, concise, and not misleading requires editing, editing, and more editing.

Analysts are thus encouraged, though less so these days, to write strategic studies on their own initiative: typically a few pages long, including an executive summary. The luckiest studies somehow avoid a consumer's immediate toss into a burn bag of classified trash, instead gaining a temporary but honored place on his desk, ready for a spare moment's reading because the content remains relevant for at least six months, in some cases for years. Yet, even if the only readers are the analyst's colleagues, every study results from practice.

What's In A Name? Sometimes Some Misunderstanding.

The efforts of Kent and his fellows to promote semantic precision could not, alas, counteract decades of Cold War routine. Misconceptions were spread, now all too common, of what "strategic" means and hence what strategic intelligence supposedly is.

One misconception is that strategic intelligence must pertain to a long

period of time. In truth, strategic intelligence pertains to strategy, whereas the particular strategy of containment lasted a long period of time. Containment emphasized patience: hold back the Communist bloc states until their internal troubles compel either their reform or their implosion. Since a long wait was expected, many strategic intelligence studies produced then were trend analyses, forecasts, and multiyear estimates. If the timeframe of a strategic issue is short, however, as several are, the strategic intelligence should mirror that.

If that seems obvious now, it was not so obvious then. Even less obvious was a Cold War routine which encouraged the idea that "strategic" means long range. In 1947, the new US Air Force saw in nuclear weapons a means to inflict so-called strategic bombing—defeat an enemy by bombing his national assets, particularly his industrial cities.

The US Army wanted nuclear weapons, too, for so-called battlefield use—to destroy Soviet Army formations in eastern Germany if they tried to invade the West. Two nuclear roles, strategic bombing and battlefield use, thus created two categories of nuclear weapons, strategic and tactical. Hence the assumption, still prevalent throughout the military today, that strategic means long range while tactical means short.

That assumption is false. Would a thermonuclear blast on a "tactical" battlefield have strategic ramifications? Of course. Consequently, today's experts in nuclear arms control cannot easily define, in precise legalistic treaty language, what makes a nuclear warhead exclusively "tactical" or "strategic." Not even the Strategic Arms Reduction Treaty (START), some 800 pages long, attempts to define the word "strategic." START defines delivery systems, such as heavy bombers and inter-continental ballistic missiles. Its negotiators could have defined "strategic" as merely some agreed number of kilometers. Yet they did not, indeed quite sensibly.

Beware of What You Wish For...

By the time Bill Clinton assumed the presidency in 1993, the Cold War was over and the world had changed. Subsequent globalization has not homogenized it. What globalization has done is link more localities than ever before—via television, e-mail, phone calls, postal packages, and airplane flights. Usually the results are beneficial, a worthy trade in goods, services, and ideas. But whenever the "locals" somewhere grow restless, the response time left to "outsiders" (actually distant participants) is now acutely short. Since a Soviet affairs expert is no longer "qualified" to speak

intelligently about Africa, the Far East, Latin America, or even about today's Russia, specialized expertise in that foreign area is now indispensable. Since terrorist networks can thrive in even the most anarchic and impoverished places, every country, indeed every province, now merits at least some intelligence attention.

In other words, today should be a golden age for strategic intelligence. Instead, what began in the 1990s as a needed intelligence reform—an attempt to reduce the analyst's isolation from the policymaker—has overcompensated, the bureaucratic pendulum pushed from one extreme to another.

Some critics accuse the reform itself of having "politicized" intelligence, for it encourages more analyst-consumer interaction than was preferred during the Cold War. More interaction does raise some risks, of course, but there were risks, too, when the analysts were isolated. Kent himself realized this late in his career. Though he remained concerned about the potential for "group think," he taught that analysts and consumers must communicate well enough that when an analyst warns of a coming international crisis, the consumer breaks away from his busy schedule and does respond, quickly—for he trusts in that analyst's competence. Otherwise, without that trust and easy access, without that professional bond, warnings are ignored too often. "Warning is like love," Kent quipped. "It takes two to make it."

The reform was initiated by Robert Gates when he was the DCI (1991-93). Drawing upon his experience as an analyst and an NSC consumer, he observed and proclaimed:

Unless intelligence officers are down in the trenches with the policymakers—understand the issues and know what US objectives are, how the process works, and who the people are—they cannot possibly provide either relevant or timely intelligence that will contribute to better-informed decisions. Others agreed, including an important advisory body in 1996, the Clinton administration's Commission on the Roles and Capabilities of the U.S. Intelligence Community. Among its recommendations was this advice:

Intelligence must be closer to those it serves The Commission believes [that the objectivity] problem is real but manageable. The need to present the "unvarnished truth" to policymakers is at the core of every analyst's training and ethos [At the same time, as one expert testified,] "if an intelligence analyst is not in some danger of being politicized, he is probably not doing his job." The Commission agrees.

Hence the phenomenal change, one which the "Long War" on terrorism has since intensified. Whatever consumers ask, analysts now endeavor to answer with unprecedented single-mindedness. Likewise in the military, operations in the Balkans, Afghanistan, and Iraq have encouraged a much closer interaction between intelligence and operations personnel. Close intelligence support has enabled successes as spectacular as the capture of Saddam Hussein. And it tracks down terrorists.

...You May Get Your Wish—But Nothing Else.

Unfortunately, when the consumers' obvious preference is for current and tactical intelligence, strategic intelligence faces neglect. Those analysts who grew up in the period when attention to strategic intelligence permitted them to deepen their skills and become genuine subject-matter experts have been dwindling away. Many have retired from government service for private sector jobs or left the field entirely.

Meanwhile, a decade's worth of younger (albeit very bright) analysts are being promoted with much less experience in that past crucible of analytical development. It is lacking because the skills necessary for strategic intelligence do not thrive in the equivalent of a crisis center, rushing from task to task, fact-sheet to fact-sheet, and blurb to blurb. "It's like cramming for finals, except we do it every day." If current trends continue, the high analytical standards of the past will go from standard procedure to "old school" to possibly a dead art.

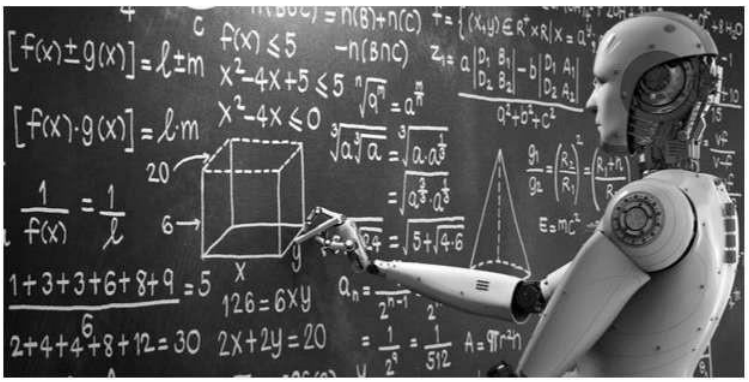
Both the 9/11 Commission and the WMD Commission have noted this strategic intelligence deficiency, the latter's report adding:

Managers and analysts throughout the Intelligence Community have repeatedly expressed frustration with their inability to carve out time for long-term research and thinking. This problem is reinforced by the current system of incentives for analysts, in which analysts are often rewarded for the number of pieces they produce, rather than the substantive depth or quality of their production.

Under the tutelage of the National Intelligence Council (NIC) there is now a unit of analysts, on rotation, officially devoted to strategic intelligence work. As beneficial as their work can be, however, the NIC itself has only 18 members. How many of the community's thousands of analysts can they mentor personally? Not the mediocre, presumably. A former chairman of the NIC, Robert Hutchings, has even expressed concern that the NIC staff, the chosen few, has gotten too involved in doing current intelligence work in order to help produce the DNI's daily morning briefings for the president.

Simply ordering the community's analysts to produce more strategic intelligence may seem the obvious solution, but decrees alone cannot change an analyst's opinion of which product types would best advance his career. As long as any "strategic intelligence" products provide only "context" and not actionable strategic support, how can the tradecraft not actually languish? Whenever a crisis grabs the headlines, a bellicose Iran or North Korea for example, analyses are published of the "strategic ramifications." But if those reports fall within the domain of strategic intelligence, they hardly fill it.

Garnering less attention are the less interesting issues and countries, presumably resulting in less expertise. There is some renewed interest in doing longer forecasts, but those particular analysts are generally separated from the rest, their experience confined mostly to themselves. Rotational assignments might help, but many years will pass before that specialized experience pervades the larger community. Furthermore, strategic intelligence work is something a young analyst should begin with, develop with, not "graduate" into after years of ignorance of it.



Let's have a look at examples of these latest techniques and the problems they are solving.

Perception: where am I and what am I looking at?

One of the fields of computer vision that has in recent years been disrupted by AI is image classification. The main advance in recent years is deep learning and particularly convolutional neural networks that are trained to recognise objects in images.

Learning techniques and algorithms for robots

Like many current machine learning techniques, the algorithms were

actually first proposed and used on a smaller scale decades ago. The reason for the recent surge in their performance and use is threefold:

- There are much larger data-sets now that engineers use to train algorithms
- Cheap access to the powerful computer hardware particularly suitable to the task
- Open access to coding techniques and tools; engineers now don't need to re-invent the wheel every time they approach a new problem

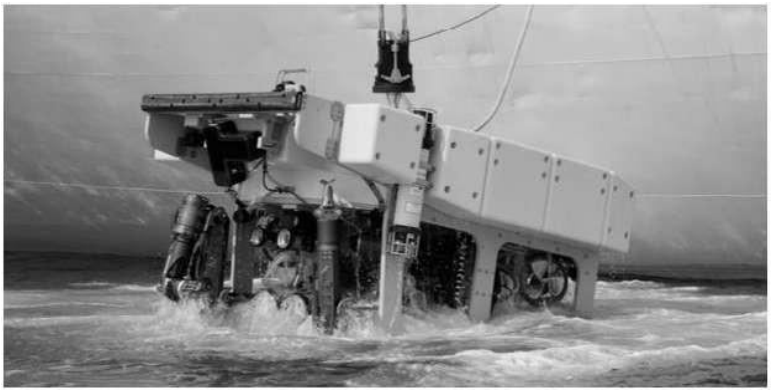


Fig. A remotely operated underwater vehicle (ROV)

So what? In robotics, this is an important capability which can reduce direct or remote human involvement in hazardous environments.

A robot capable of identifying objects in images, from an onboard camera, in real time, can perceive more about its surroundings.

These algorithms may be trained to recognise specific defects in structures being inspected, potentially using the same transfer learning technique as in some of the recent examples of neural networks for diagnosis from medical images.

Planning and Control: how can I complete my task?

Reinforcement learning is one of the most exciting developments in machine learning in recent years.

The most commonly seen examples are those of AI playing various retro computer games.

These examples may appear to be just for fun however, what they demonstrate is that very general tasks can be solved using only using what the machine can see.

The game's score acts as the machine's reward.

It means that the algorithms being developed are highly general so they can be retrained for many different tasks and environments.

In much the same way as a baby will experiment with the world and learn over time what works and what does not, eventually complex behaviours can be learnt in order to achieve a simple goal.

Real life imitating games

So what? If reinforcement learning can be applied to games, it can be applied to simulations of many real-life environments.

If a simulation of a robot and its environment is close enough to real life, then this technique can explore and optimise different solutions to tasks.

In some cases, simulations can be sped up so years of learning can be compressed into just hours.

Training a robot workforce

For many robots that work in extreme environments, explicitly programmed instructions are either not feasible, or so time-consuming for a human that the robot becomes an unproductive burden.

Reinforcement learning means that robots can be given simple goals to achieve, and they will use their learnt techniques and their internal knowledge of the world to plan how to achieve a task.

At Innovate UK we are funding projects that use machine learning algorithms across all sectors, and into all of the Industrial Strategy Grand Challenges, from medicine to the digital economy to space, transportation and agriculture.

AI points of view

Artificial Intelligence (AI). Two words, which together conjure up an extraordinarily wide range of meanings, and with them, opinions and emotions. Ask one person, and their view is that AI is an existential threat to humanity; potentially taking all our jobs or even turning against us like

the sci-fi like visions of killer robots. Ask another, and you might be met with an eye roll, that the entire subject can be dismissed as simply algorithms, and anything more is just hype.

Both extreme viewpoints have their truths and their fallacies. In reality, the machine learning techniques that make the modern AI magic tricks happen are progressing at an exciting rate, transforming some industries, but simultaneously the various AI apocalypse scenarios are a comforting distance away.

AI AND OUR FUTURE WORKFORCE

The World Economic Forum estimates that, by 2022, a large proportion of companies will have adopted technologies such as machine learning, and therefore strongly encourages governments and education to focus on rapidly raising education and skills, with a focus on both STEM (science, technology, engineering and mathematics) and non-cognitive soft skills, in order to meet this impending need.

Microsoft's recent study shows that, by 2030, students will need to have mastered two facets of this new world by the time they graduate:

- Know how to utilise ever-changing technology, such as AI, to their advantage
- Understand how to work with other people in a team to problem solve effectively

Preparing students to work alongside AI in the future can start early. As most children are comfortable with digital technology by the time they are of school age, teaching them the skills they'll need to thrive in a digital workplace is important. Add the inclusion of AI in education, and the workforce of the future will be better prepared to face the unknown challenges of the workplace of tomorrow.

MACHINE LEARNING

The meat and potatoes of AI is machine learning — in fact it's typically acceptable to substitute the terms artificial intelligence and machine learning for one another. They aren't quite the same, however, but connected.

Machine learning is the process by which an AI uses algorithms to perform artificial intelligence functions. It's the result of applying rules to create outcomes through an AI.

Black box

When the rules are applied an AI does a lot of complex math. This math, often, can't even be understood by humans (and sometimes it just wouldn't be worth the time it would take for us to figure it all out) yet the system outputs useful information. When this happens it's called black box learning. The real work happens in such a way that we don't really care how the computer arrived at the decisions it's made, because we know what rules it used to get there. Black box learning is how we can ethically skip "showing our work" like we had to in high school algebra.

Neural network

When we want an AI to get better at something we create a neural network. These networks are designed to be very similar to the human nervous system and brain. It uses stages of learning to give AI the ability to solve complex problems by breaking them down into levels of data. The first level of the network may only worry about a few pixels in an image file and check for similarities in other files. Once the initial stage is done, the neural network will pass its findings to the next level which will try to understand a few more pixels, and perhaps some metadata. This process continues at every level of a neural network.

Deep learning

Deep learning is what happens when a neural network gets to work. As the layers process data the AI gains a basic understanding. You might be teaching your AI to understand cats, but once it learns what paws are that AI can apply that knowledge to a different task. Deep learning means that instead of understanding what something is, the AI begins to learn "why."

Natural language processing

It takes an advanced neural network to parse human language. When an AI is trained to interpret human communication it's called natural language processing. This is useful for chat bots and translation services, but it's also represented at the cutting edge by AI assistants like Alexa and Siri.

Reinforcement learning

AI is a lot more like humans than we might be comfortable believing. We learn in almost the exact same way. One method of teaching a machine,

just like a person, is to use reinforcement learning. This involves giving the AI a goal that isn't defined with a specific metric, such as telling it to "improve efficiency" or "find solutions." Instead of finding one specific answer the AI will run scenarios and report results, which are then evaluated by humans and judged. The AI takes the feedback and adjusts the next scenario to achieve better results.

Supervised learning

This is the very serious business of proving things. When you train an AI model using a supervised learning method you provide the machine with the correct answer ahead of time. Basically the AI knows the answer and it knows the question. This is the most common method of training because it yields the most data: it defines patterns between the question and answer. If you want to know why something happens, or how something happens, an AI can look at the data and determine connections using the supervised learning method.

Unsupervised learning

In many ways the spookiest part of AI research is realizing that the machines are really capable of learning, and they're using layers upon layers of data and processing capability to do so. With unsupervised learning we don't give the AI an answer. Rather than finding patterns that are predefined like, "why people choose one brand over another," we simply feed a machine a bunch of data so that it can find whatever patterns it is able to.

Transfer learning

Another spooky way machines can learn is through transfer learning. Once an AI has successfully learned something, like how to determine if an image is a cat or not, it can continue to build on it's knowledge even if you aren't asking it to learn anything about cats. You could take an AI that can determine if an image is a cat with 90-percent accuracy, hypothetically, and after it spent a week training on identifying shoes it could then return to its work on cats with a noticeable improvement in accuracy.

Turing Test

If you're like most AI experts you're cautiously optimistic about the future and you have reservations about our safety as we draw closer to

robots that are indistinguishable from people. Alan Turing shared your concerns. Though he died in 1954 his legacy lives on in two ways. Primarily he's credited with cracking Nazi codes and helping the Allies win World War 2. He's also the father of modern computing and the creator of the Turing Test.

While the test was originally conceived as a way of determining if a human could be fooled by a conversation, in text display only, between a human and an artificial intelligence, it has since become short hand for any AI that can fool a person into believing they're seeing or interacting with a real person.

6

Artificial Intelligence in Software Metrics for Algorithmic Trading

SOFTWARE METRICS IN ALGORITHMIC

In computer science, efficiency is used to describe properties of an algorithm relating to how much of various types of resources it consumes. Algorithmic efficiency can be thought of as analogous to engineering productivity for a repeating or continuous process, where the goal is to reduce resource consumption, including time to completion, to some acceptable, optimal level.

Software Metrics

The two most frequently encountered and measurable metrics of an algorithm are:-

- speed or running time - the time it takes for an algorithm to complete, and
- 'space' - the memory or 'non-volatile storage' used by the algorithm during its operation.

but also might apply to

- transmission size - such as required bandwidth during normal operation or
- size of external memory- such as temporary disk space used to accomplish its task

and perhaps even

- the size of required 'longterm' disk space required after its operation to record its output or maintain its required function during its

required useful lifetime (examples: a data table, archive or a computer log) and also

- the performance per watt and the total energy, consumed by the chosen hardware implementation (with its System requirements, necessary auxiliary support systems including interfaces, cabling, switching, cooling and security), during its required useful lifetime. See Total cost of ownership for other potential costs that might be associated with any particular implementation.

(An extreme example of these metrics might be to consider their values in relation to a repeated simple algorithm for calculating and storing (ð+n) to 50 decimal places running for say, 24 hours, either on a "pocket calculator" sized processor such as an ipod or an early mainframe operating in its own purpose-built heated or air conditioned unit.) The process of making code more efficient is known as optimization and in the case of automatic optimization (i.e. compiler optimization - performed by compilers on request or by default), usually focus on space at the cost of speed, or vice versa.

There are also quite simple programming techniques and 'avoidance strategies' that can actually improve both at the same time, usually irrespective of hardware, software or language. Even the re-ordering of nested conditional statements - to put the least frequently occurring condition first (example: test patients for blood type ='AB-', before testing age > 18, since this type of blood occurs in only about 1 in 100 of the population - thereby eliminating the second test at runtime in 99% of instances), can reduce actual instruction path length, something an optimizing compiler would almost certainly not be aware of - but which a programmer can research relatively easily even without specialist medical knowledge.

History

The first machines that were capable of computation were severely limited by purely mechanical considerations. As later electronic machines were developed they were, in turn, limited by the speed of their electronic counterparts. As software replaced hard-wired circuits, the efficiency of algorithms also became important. It has long been recognized that the precise 'arrangement of processes' is critical in reducing elapse time.

- "In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations

must influence the selections amongst them for the purposes of a calculating engine. One essential object is to choose that arrangement which shall tend to reduce to a minimum the time necessary for completing the calculation"

Ada Lovelace 1815-1852, generally considered as 'the first programmer' who worked on Charles Babbage's early mechanical general-purpose computer

- "In established engineering disciplines a 12% improvement, easily obtained, is never considered marginal and I believe the same viewpoint should prevail in software engineering"

Extract from "Structured Programming with go to Statements" by Donald Knuth, renowned computer scientist and Professor Emeritus of the Art of Computer Programming at Stanford University.

- "The key to performance is elegance, not battalions of special cases" attributed to Jon Bentley and (Malcolm) Douglas McIlroy

Speed

The absolute speed of an algorithm for a given input can simply be measured as the duration of execution (or clock time) and the results can be averaged over several executions to eliminate possible random effects. Most modern processors operate in a multi-processing & multi-programming environment so consideration must be made for parallel processes occurring on the same physical machine, eliminating these as far as possible.

For superscalar processors, the speed of a given algorithm can sometimes be improved through instruction-level parallelism within a single processor (but, for optimal results, the algorithm may require some adaptation to this environment to gain significant advantage ('speedup'), becoming, in effect, an entirely different algorithm). A relative measure of an algorithms performance can sometimes be gained from the total instruction path length which can be determined by a run time Instruction Set Simulator (where available). An estimate of the speed of an algorithm can be determined in various ways. The most common method uses time complexity to determine the Big-O of an algorithm. See Run-time analysis for estimating how fast a particular algorithm may be according to its type (example: lookup unsorted list, lookup sorted list etc.) and in terms of scalability - its dependence on 'size of input', processor power and other factors.

Memory

Often, it is possible to make an algorithm faster at the expense of memory. This might be the case whenever the result of an 'expensive' calculation is cached rather than recalculating it afresh each time. The additional memory requirement would, in this case, be considered additional overhead although, in many situations, the stored result occupies very little extra space and can often be held in pre-compiled static storage, reducing not just processing time but also allocation & deallocation of working memory. This is a very common method of improving speed, so much so that some programming languages often add special features to support it, such as C++'s 'mutable' keyword. The memory requirement of an algorithm is actually two separate but related things:-

- The memory taken up by the compiled executable code (the object code or binary file) itself (on disk or equivalent, depending on the hardware and language). This can often be reduced by preferring run-time decision making mechanisms (such as virtual functions and run-time type information) over certain compile-time decision making mechanisms (such as macro substitution and templates). This, however, comes at the cost of speed.
- Amount of temporary "dynamic memory" allocated during processing. For example, dynamically pre-caching results, as mentioned earlier, improves speed at the cost of this attribute. Even the depth of sub-routine calls can impact heavily on this cost and increase path length too, especially if there are 'heavy' dynamic memory requirements for the particular functions invoked. The use of copied function parameters (rather than simply using pointers to earlier, already defined, and sometimes static values) actually doubles the memory requirement for this particular memory metric (as well as carrying its own processing overhead for the copying itself. This can be particularly relevant for quite 'lengthy' parameters such as html script, JavaScript source programmes or extensive freeform text such as letters or emails.

Rematerialization

It has been argued that Rematerialization (re-calculating) may occasionally be more efficient than holding results in cache. This is the somewhat non-intuitive belief that it can be faster to re-calculate from the input - even if the answer is already known - when it can be shown, in some special cases, to decrease "register pressure". Some optimizing

compilers have the ability to decide when this is considered worthwhile based on a number of criteria such as complexity and no side effects, and works by keeping track of the expression used to compute each variable, using the concept of available expressions. This is most likely to be true when a calculation is very fast (such as addition or bitwise operations), while the amount of data which must be cached would be very large, resulting in inefficient storage. Small amounts of data can be stored very efficiently in registers or fast cache, while in most contemporary computers large amounts of data must be stored in slower memory or even slower hard drive storage, and thus the efficiency of storing data which can be computed quickly drops significantly.

Precomputation

Precomputing a complete range of results prior to compiling, or at the beginning of an algorithm's execution, can often increase algorithmic efficiency substantially. This becomes advantageous when one or more inputs is constrained to a small enough range that the results can be stored in a reasonably sized block of memory. Because memory access is essentially constant in time complexity (except for caching delays), any algorithm with a component which has worse than constant efficiency over a small input range can be improved by precomputing values. In some cases efficient approximation algorithms can be obtained by computing a discrete subset of values and interpolating for intermediate input values, since interpolation is also a linear operation.

Transmission Size

Data compression algorithms can be useful because they help reduce the consumption of expensive resources, such as hard disk space or transmission bandwidth. This however also comes at a cost - which is additional processing time to compress and subsequently decompress. Depending upon the speed of the data transfer, compression may reduce overall response times which, ultimately, equates to speed - even though processing within the computer itself takes longer. For audio, MP3 is a compression method used extensively in portable sound systems. The efficiency of a data compression algorithm relates to the compression factor and speed of achieving both compression and decompression. For the purpose of archiving an extensive database, it might be considered worthwhile to achieve a very high compression ratio, since decompression is less likely to occur on the majority of the data.

Data Presentation

Output data can be presented to the end user in many ways - such as via punched tape or card, digital displays, local display monitors, remote computer monitors or printed. Each of these has its own inherent initial cost and, in some cases, an ongoing cost (e.g. refreshing an image from memory). As an example, the web site "Google" recently showed, as its logo, an image of the Vancouver olympics that is around 8K of gif image. The normally displayed Google image is a PNG image of 28K (or 48K), yet the raw text string for "Google" occupies only 6 octets or 48 bits (4,778 or 8192 *times* less).

This graphically illustrates that how data is presented can significantly effect the overall efficiency of transmission (and also the complete algorithm - since both GIF and PNG images require yet more processing). It is estimated by "Internet World Stats" that there were 1,733,993,741 internet users in 2009 and, to transmit this new image to each one of them, would require around 136,000 *billion* (10^9)octets of data to be transmitted - at least once - into their personal web cache. In "Computational Energy Cost of TCP", co-authors Bokyung Wang and Suresh Singh examine the energy costs for TCP and calculated, for their chosen example, a cost of 0.022 Joules per packet (of approx 1489 octets). On this basis, a total of around 2,000,000,000 joules (2 GJ) of energy might be expended by TCP elements alone to display the new logo for all users for the first time. To maintain or re-display this image requires still more processing and consequential energy cost (in contrast to printed output for instance).

Encoding and Decoding Methods (Compared and Contrasted)

When data is encoded for any 'external' use, it is possible to do so in an almost unlimited variety of different formats that are sometimes conflicting. This content encoding (of the raw data) may be designed for:

- optimal readability – by humans
- optimal decoding speed – by other computer programmes
- optimal compression – for archiving or data transmission
- optimal compatibility – with "legacy" or other existing formats or programming languages
- optimal security – using encryption

(For character level encoding, see the various encoding techniques such as EBCDIC or ASCII)

It is unlikely that all of these goals could be met with a single 'generic' encoding scheme and so a compromise will often be the desired goal and will often be compromised by the need for standardization and/or legacy and compatibility issues.

Encoding Efficiently

For data encoding whose destination is to be input for further computer processing, readability is not an issue – since the receiving processors algorithm can decode the data to any other desirable form including human readable. From the perspective of algorithmic efficiency, minimizing subsequent decoding (with zero or minimal parsing) should take highest priority.

The general rule of thumb is that any encoding system that 'understands' the underlying data structure - sufficiently to encode it in the first place - should be equally capable of easily encoding it in such a way that makes decoding it highly efficient. For variable length data with possibly omitted data values, for instance, this almost certainly means the utilization of declarative notation (i.e. including the length of the data item as a prefix to the data so that a de-limiter is not required and parsing completely eliminated). For keyword data items, tokenizing the key to an index (integer) after its first occurrence not only reduces subsequent data size but, furthermore, reduces future decoding overhead for the same items that are repeated. For more 'generic' encoding for efficient data compression see Arithmetic encoding and entropy encoding articles.

Historically, optimal encoding was not only worthwhile from an efficiency standpoint but was also common practise to conserve valuable memory, external storage and processor resources. Once validated a country name for example could be held as a shorter sequential country code which could then also act as an index for subsequent 'decoding', using this code as an entry number within a table or record number within a file. If the table or file contained fixed length entries, the code could easily be converted to an absolute memory address or disk address for fast retrieval. The ISBN system for identifying books is a good example of a practical encoding method which also contains a built-in hierarchy. According to recent articles in New Scientist and Scientific American; "TODAY'S telecommunications networks could use one ten-thousandth of the power they presently consume if smarter data-coding techniques were used", according to Bell Labs, based in Murray Hill, New Jersey It recognizes that this is only a theoretical limit but nevertheless sets itself a more realistic,

practical goal of a 1,000 fold reduction within 5 years with future, as yet unidentified, technological changes.

Examples of Several Common Encoding Methods

- Comma separated values (CSV - a list of data values separated by commas)
- Tab separated values (TSV) - a list of data values separated by 'tab' characters
- HyperText Markup Language (HTML) - the predominant markup language for web pages
- Extensible Markup Language (XML) - a generic framework for storing any amount of text or any data whose structure can be represented as a tree with at least one element - the root element.
- JavaScript Object Notation (JSON) - human-readable format for representing simple data structures

The last of these, (JSON) is apparently widely used for internet data transmission, primarily it seems because the data can be uploaded by a single JavaScript 'eval' statement - without the need to produce what otherwise would likely have been a more efficient purpose built encoder/ decoder. There are in fact quite large amounts of repeated (and therefore redundant data) in this particular format, and also in HTML and XML source, that could quite easily be eliminated. XML is recognized as a verbose form of encoding. Binary XML has been put forward as one method of reducing transfer and processing times for XML and, while there are several competing formats, none has been widely adopted by a standards organization or accepted as a de facto standard. It has also been criticized by Jimmy Zhang for essentially trying to solve the wrong problem There are a number of freely available products on the market that partially compress HTML files and perform some or all of the following:

- merge lines
- remove unnecessary whitespace characters
- remove unnecessary quotation marks. For example, BORDER="0" will be replaced with BORDER=0)
- replace some tags with shorter ones (e.g. replace STRIKE tags with S, STRONG with B and EM with I)
- remove HTML comments (comments within scripts and styles are not removed)

- remove <!DOCTYPE..> tags
- remove named meta tags

The effect of this limited form of compression is to make the HTML code smaller and faster to load, but more difficult to read manually (so the original HTML code is usually retained for updating), but since it is predominantly meant to be processed only by a browser, this causes no problems. Despite these small improvements, HTML, which is the predominant language for the web still remains a predominantly *source* distributed, interpreted markup language, with high redundancy.

Kolmogorov Complexity

The study of encoding techniques has been examined in depth in an area of computer science characterized by a method known as Kolmogorov complexity where a value known as ('K') is accepted as 'not a computable function'. The Kolmogorov complexity of any computable object is the length of the shortest programme that computes it and then halts. The invariance theorem shows that it is not really important which computer is used. Essentially this implies that there is no automated method that can produce an optimum result and is therefore characterized by a requirement for human ingenuity or Innovation. See also Algorithmic probability.

Effect of Programming Paradigms

The effect that different programming paradigms have on algorithmic efficiency is fiercely contested, with both supporters and antagonists for each new paradigm.

Strong supporters of structured programming, such as Dijkstra for instance, who favour entirely goto-less programmes are met with conflicting evidence that appears to nullify its supposed benefits. The truth is, even if the structured code itself contains no gotos, the optimizing compiler that creates the binary code almost certainly generates them (and not necessarily in the most efficient way). Similarly, OOP protagonists who claim their paradigm is superior are met with opposition from strong sceptics such as Alexander Stepanov who suggested that OOP provides a mathematically limited viewpoint and called it, "almost as much of a hoax as Artificial Intelligence" In the long term, benchmarks, using real-life examples, provide the only real hope of resolving such conflicts - at least in terms of run-time efficiency.

Optimization Techniques

The word optimize is normally used in relation to an existing algorithm/ computer programme (i.e. to improve upon completed code). In this section it is used both in the context of existing programmes and also in the design and implementation of new algorithms, thereby avoiding the most common performance pitfalls. It is clearly wasteful to produce a working programme - at first using an algorithm that ignores all efficiency issues - only to then have to redesign or rewrite sections of it if found to offer poor performance. Optimization can be broadly categorized into two domains:-

- Environment specific - that are essentially worthwhile only on certain platforms or particular computer languages
- General techniques - that apply irrespective of platform

Environment Specific

Optimization of algorithms frequently depends on the properties of the machine the algorithm will be executed on as well as the language the algorithm is written in and chosen data types. For example, a programmer might optimize code for time efficiency in an application for home computers (with sizable amounts of memory), but for code destined to be embedded in small, "memory-tight" devices, the programmer may have to accept that it will run more slowly, simply because of the restricted memory available for any potential software optimization. For a discussion of hardware performance, see article on Computer performance which covers such things as CPU clock speed, cycles per instruction and other relevant metrics. For a discussion on how the choice of particular instructions available on a specific machine effect efficiency, see later section 'Choice of instruction and data type'.

General Techniques

- Linear search such as unsorted table look-ups in particular can be very expensive in terms of execution time but can be reduced significantly through use of efficient techniques such as indexed arrays and binary searches. Using a simple linear search on first occurrence and using a cached result thereafter is an obvious compromise.
- Use of indexed programme branching, utilizing branch tables or "threaded code" to control programme flow, (rather than using multiple conditional IF statements or unoptimized CASE/SWITCH)

can drastically reduce instruction path length, simultaneously reduce programme size and even also make a programme easier to read and more easily maintainable (in effect it becomes a 'decision table' rather than repetitive spaghetti code).

- Loop unrolling performed manually, or more usually by an optimizing compiler, can provide significant savings in some instances. By processing 'blocks' of several array elements at a time, individually addressed, (for example, within a While loop), much pointer arithmetic and end of loop testing can be eliminated, resulting in decreased instruction path lengths. Other Loop optimizations are also possible.

Tunnel Vision

There are many techniques for improving algorithms, but focusing on a single favourite technique can lead to a "tunnel vision" mentality. For example, in this X86 assembly example, the author offers loop unrolling as a reasonable technique that provides some 40% improvements to his chosen example. However, the same example would benefit significantly from both inlining and use of a trivial hash function. If they were implemented, either as alternative or complementary techniques, an even greater percentage gain might be expected. A combination of optimizations may provide ever increasing speed, but selection of the most easily implemented and most effective technique, from a large repertoire of such techniques, is desirable as a starting point.

Dependency Trees and Spreadsheets

Spreadsheets are a 'special case' of algorithms that self-optimize by virtue of their dependency trees that are inherent in the design of spreadsheets to reduce re-calculations when a cell changes. The results of earlier calculations are effectively cached within the workbook and only updated if another cells changed value effects it directly.

Table Lookup

Table lookups can make many algorithms more efficient, particularly when used to bypass computations with a high time complexity. However, if a wide input range is required, they can consume significant storage resources. In cases with a sparse valid input set, hash functions can be used to provide more efficient lookup access than a full table.

Hash Function Algorithms

A hash function is any well-defined procedure or mathematical function which converts a large, possibly variable-sized amount of data into a small datum, usually a single integer that may serve as an index to an array. The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes. Hash functions are frequently used to speed up table lookups. The choice of a hashing function (to avoid a linear or brute force search) depends critically on the nature of the input data, and their probability distribution in the intended application.

Trivial Hash Function

Sometimes if the datum is small enough, a "trivial hash function" can be used to effectively provide constant time searches at almost zero cost. This is particularly relevant for single byte lookups (e.g. ASCII or EBCDIC characters)

Searching Strings

Searching for particular text strings (for instance "tags" or keywords) in long sequences of characters potentially generates lengthy instruction paths. This includes searching for delimiters in comma separated files or similar processing which can be very simply and effectively eliminated (using declarative notation for instance). Several methods of reducing the cost for general searching have been examined and the "Boyer–Moore string search algorithm" (or Boyer–Moore–Horspool algorithm, a similar but modified version) is one solution that has been proven to give superior results to repetitive comparisons of the entire search string along the sequence.

Hot Spot Analyzers

Special system software products known as "performance analyzers" are often available from suppliers to help diagnose "hot spots" - during actual execution of computer programmes - using real or test data - they perform a Performance analysis under generally repeatable conditions. They can pinpoint sections of the programme that might benefit from specifically targeted programmer optimization without necessarily spending time optimizing the rest of the code. Using programme re-runs, a measure of relative improvement can then be determined to decide if the optimization was successful and by what amount. Instruction Set Simulators can be used as an alternative to measure the instruction path

length at the machine code level between selected execution paths, or on the entire execution. Regardless of the type of tool used, the quantitative values obtained can be used in combination with anticipated reductions (for the targeted code) to estimate a relative or absolute overall saving. For example if 50% of the total execution time (or path length) is absorbed in a subroutine whose speed can be doubled by programmer optimization, an overall saving of around 25% might be expected (Amdahl law). Efforts have been made at the University of California, Irvine to produce dynamic executable code using a combination of hot spot analysis and run-time programme trace tree. A JIT like dynamic compiler was built by Andreas Gal and others, "in which relevant (i.e., frequently executed) control flows are ...discovered lazily during execution"

Benchmarking & Competitive Algorithms

For new versions of software or to provide comparisons with competitive systems, benchmarks are sometimes used which assist with gauging an algorithms relative performance. If a new sort algorithm is produced for example it can be compared with its predecessors to ensure that at least it is efficient as before with known data - taking into consideration any functional improvements. Benchmarks can be used by customers when comparing various products from alternative suppliers to estimate which product will best suit their specific requirements in terms of functionality and performance. For example in the mainframe world certain proprietary sort products from independent software companies such as Syncsort compete with products from the major suppliers such as IBM for speed. Some benchmarks provide opportunities for producing an analysis comparing the relative speed of various compiled and interpreted languages for example and *The Computer Language Benchmarks Game* compares the performance of implementations of typical programming problems in several programming languages. (Even creating "do it yourself" benchmarks to get at least some appreciation of the relative performance of different programming languages, using a variety of user specified criteria, is quite simple to produce as this "Nine language Performance roundup" by Christopher W. Cowell-Shah demonstrates by example)

Compiled Versus Interpreted Languages

A compiled algorithm will, in general, execute faster than the equivalent interpreted algorithm simply because some processing is required even

at run time to 'understand' (i.e. interpret) the instructions to effect an execution. A compiled programme will normally output an object or machine code equivalent of the algorithm that has already been processed by the compiler into a form more readily executed by microcode or the hardware directly. The popular Perl language is an example of an interpreted language and benchmarks indicate that it executes approximately 24 times more slowly than compiled C.

Optimizing Compilers

Many compilers have features that attempt to optimize the code they generate, utilizing some of the techniques outlined in this study and others specific to the compilation itself. Loop optimization is often the focus of optimizing compilers because significant time is spent in programme loops and parallel processing opportunities can often be facilitated by automatic code re-structuring such as loop unrolling. Optimizing compilers are by no means perfect. There is no way that a compiler can guarantee that, for all programme source code, the fastest (or smallest) possible equivalent compiled programme is output; such a compiler is fundamentally impossible because it would solve the halting problem. Additionally, even optimizing compilers generally have no access to runtime metrics to enable them to improve optimization through 'learning'.

Just-in-Time Compilers

'On-the-fly' processors known today as just-in-time or 'JIT' compilers combine features of interpreted languages with compiled languages and may also incorporate elements of optimization to a greater or lesser extent. Essentially the JIT compiler can compile small sections of source code statements (or bytecode) as they are newly encountered and (usually) retain the result for the next time the same source is processed. In addition, pre-compiled segments of code can be in-lined or called as dynamic functions that themselves perform equally fast as the equivalent 'custom' compiled function. Because the JIT processor also has access to run-time information (that a normal compiler can't have) it is also possible for it to optimize further executions depending upon the input and also perform other run-time introspective optimization as execution proceeds. A JIT processor may, or may not, incorporate self modifying code or its equivalent by creating 'fast path' routes through an algorithm. It may also use such techniques as dynamic Fractional cascading or any other similar runtime device based on collected actual runtime metrics. It is therefore entirely

possible that a JIT compiler might (counter intuitively) execute even faster than an optimally 'optimized' compiled programme.

Self-Modifying Code

As mentioned above, just-in-time compilers often make extensive use of self-modifying code to generate the actual machine instructions required to be executed.

The technique can also be used to reduce instruction path lengths in application programmes where otherwise repetitive conditional tests might otherwise be required within the main programme flow. This can be particularly useful where a sub routine may have embedded debugging code that is either active (testing mode) or inactive (production mode) depending upon some input parameter. A simple solution using a form of dynamic dispatching is where the sub routine entry point is dynamically 'swapped' at initialization, depending upon the input parameter. Entry point A) includes the debugging code prologue and entry point B) excludes the prologue; thus eliminating all overhead except the initial 'test and swap' (even when test/debugging is selected, when the overhead is simply the test/debugging code itself).

Genetic Algorithm

In the world of performance related algorithms it is worth mentioning the role of genetic algorithms which compete using similar methods to the natural world in eliminating inferior algorithms in favour of more efficient versions.

Object Code Optimizers

Some proprietary programme optimizers such as the "COBOL Optimizer" developed by Capex Corporation in the mid 1970's for COBOL, actually took the unusual step of optimizing the Object code (or binary file) after normal compilation. This type of optimizer, recently sometimes referred to as a "post pass" optimizer or peephole optimizer, depended, in this case, upon knowledge of 'weaknesses' in the standard IBM COBOL compiler and actually replaced (or patched) sections of the object code with more efficient code. A number of other suppliers have recently adopted the same approach.

Alignment of Data

Most processors execute faster if certain data values are aligned on

word, doubleword or page boundaries. If possible design/specify structures to satisfy appropriate alignments. This avoids exceptions.

Locality of Reference

To reduce Cache miss exceptions by providing good spatial locality of reference, specify 'high frequency'/volative working storage data within defined structure(s) so that they are also allocated from contiguous sections of memory (rather than possibly scattered over many pages). Group closely related data values also 'physically' close together within these structures. Consider the possibility of creating an 'artificial' structure to group some otherwise unrelated, but nevertheless frequently referenced, items together.

Choice of Instruction or Data Type

Particularly in an Assembly language (although also applicable to HLL statements), the choice of a particular 'instruction' or data type, can have a large impact on execution efficiency.

In general, instructions that process variables such as signed or unsigned 16-bit or 32-bit integers are faster than those that process floating point or packed decimal. Modern processors are even capable of executing multiple 'fixed point' instructions in parallel with the simultaneous execution of a floating point instruction.

If the largest integer to be encountered can be accommodated by the 'faster' data type, defining the variables as that type will result in faster execution - since even a non-optimizing compiler will, in-effect, be 'forced' to choose appropriate instructions that will execute faster than would have been the case with data types associated with 'slower' instructions. Assembler programmers (and optimizing compiler writers) can then also benefit from the ability to perform certain common types of arithmetic for instance - division by 2, 4, 8 etc. by performing the very much faster binary shift right operations (in this case by 1, 2 or 3 bits). If the choice of input data type is not under the control of the programmer, although prior conversion (outside of a loop for instance) to a faster data type carries some overhead, it can often be worthwhile if the variable is then to be used as a loop counter, especially if the count could be quite a high value or there are many input values to process.

As mentioned above, choice of individual assembler instructions (or even sometimes just their order of execution) on particular machines can effect the efficiency of an algorithm.

See Assembly Optimization Tips for one quite numerous arcane list of various technical (and sometimes non-intuitive) considerations for choice of assembly instructions on different processors that also discusses the merits of each case. Sometimes microcode or hardware quirks can result in unexpected performance differences between processors that assembler programmers can actively code for - or else specifically avoid if penalties result - something even the best optimizing compiler may not be designed to handle.

Data Granularity

The greater the granularity of data definitions (such as splitting a geographic address into separate street/city/postal code fields) can have performance overhead implications during processing. Higher granularity in this example implies more procedure calls in Object-oriented programming and parallel computing environments since the additional objects are accessed via multiple method calls rather than perhaps one.

Subroutine Granularity

For structured programming and procedural programming in general, great emphasis is placed on designing programmes as a hierarchy of (or at least a set of) subroutines.

For object oriented programming, the method call (a subroutine call) is the standard method of testing and setting all values in objects and so increasing data granularity consequently causes increased use of subroutines.

The greater the granularity of subroutine usage, the larger the proportion of processing time devoted to the mechanism of the subroutine linkages themselves.The presence of a (called) subroutine in a programme contributes nothing extra to the functionality of the programme. The extent to which subroutines (and their consequent memory requirements) influences the overall performance of the complete algorithm depends to a large extent on the actual implementation.

In assembly language programmes, the invocation of a subroutine need not involve much overhead, typically adding just a couple of machine instructions to the normal instruction path length, each one altering the control flow either *to* the subroutine or returning *from* it (saving the state on a stack being optional, depending on the complexity of the subroutine and its requirement to reuse general purpose registers). In many cases, small subroutines that perform frequently used data transformations using

'general purpose' work areas can be accomplished without the need to save or restore any registers, including the return register.

By contrast, HLL programmes typically always invoke a 'standard' procedure call (the *calling convention*), which involves saving the programme state by default and usually allocating additional memory on the stack to save all registers and other relevant state data (the prologue and epilogue code). Recursion in a HLL programme can consequently consume significant overhead in both memory and execution time managing the stack to the required depth.

Guy Steele pointed out in a 1977 paper that a *well-designed* programming language implementation *can* have very low overheads for procedural abstraction (but laments, in most implementations, that they seldom achieve this in practice - being "rather thoughtless or careless in this regard"). Steele concludes that "we should have a healthy respect for procedure calls" (because they are powerful) but he also cautioned "use them sparingly" See section Avoiding costs for discussion of how inlining subroutines can be used to improve performance.

For the Java language, use of the "final" keyword can be used to force method inlining (resulting in elimination of the method call, no dynamic dispatch and the possibility to constant-fold the value - with no code executed at runtime)

Choice of Language / Mixed Languages

Some computer languages can execute algorithms more efficiently than others. As stated already, interpreted languages often perform less efficiently than compiled languages. In addition, where possible, 'high-use', and time-dependent sections of code may be written in a language such as assembler that can usually execute faster and make better use of resources on a particular platform than the equivalent HLL code on the same platform. This section of code can either be statically called or dynamically invoked (external function) or embedded within the higher level code (e.g. Assembler instructions embedded in a 'C' language program).

The effect of higher levels of abstraction when using a HLL has been described as the *Abstraction penalty* Programmers who are familiar with assembler language (in addition to their chosen HLL) and are also familiar with the code that will be generated by the HLL, under known conditions, can sometimes use HLL language primitives of that language to generate

code almost identical to assembler language. This is most likely to be possible only in languages that support pointers such as PL/1 or C. This is facilitated if the chosen HLL compiler provides an optional assembler listing as part of its printout so that various alternatives can be explored without also needing specialist knowledge of the compiler internals.

Software Validation Versus Hardware Validation

An optimization technique that was frequently taken advantage of on 'legacy' platforms was that of allowing the hardware (or microcode) to perform validation on numeric data fields such as those coded in (or converted to) packed decimal (or packed BCD). The choice was to either spend processing time checking each field for a valid numeric content in the particular internal representation chosen or simply assume the data was correct and let the hardware detect the error upon execution. The choice was highly significant because to check for validity on multiple fields (for sometimes many millions of input records), it could occupy valuable computer resources. Since input data fields were in any case frequently built from the output of earlier computer processing, the actual probability of a field containing invalid data was exceedingly low and usually the result of some 'corruption'. The solution was to incorporate an 'event handler' for the hardware detected condition ('data exception)' that would intercept the occasional errant data field and either 'report, correct and continue' or, more usually, abort the run with a core dump to try to determine the reason for the bad data.

Similar event handlers are frequently utilized in today's web based applications to handle other exceptional conditions but repeatedly parsing data input, to ensure its validity before execution, has nevertheless become much more commonplace - partly because processors have become faster (and the perceived need for efficiency in this area less significant) but, predominantly - because data structures have become less 'formalized' (e.g. .csv and .tsv files) or uniquely identifiable (e.g. packed decimal). The potential savings using this type of technique may have therefore fallen into general dis-use as a consequence and therefore repeated data validations and repeated data conversions have become an accepted overhead. Ironically, one consequence of this move to less formalized data structures is that a corruption of say, a numeric binary integer value, will not be detected at all by the hardware upon execution (for instance: is an ASCII hexadecimal value '20202020' a valid signed or unsigned binary value - or simply a string of blanks that has corrupted it?)

Algorithms for Vector & Superscalar Processors

Algorithms for vector processors are usually different than those for scalar processors since they can process multiple instructions and/or multiple data elements in parallel. The process of converting an algorithm from a scalar to a vector process is known as vectorization and methods for automatically performing this transformation as efficiently as possible are constantly sought. There are intrinsic limitations for implementing instruction level parallelism in Superscalar processors but, in essence, the overhead in deciding for certain if particular instruction sequences can be processed in parallel can sometimes exceed the efficiency gain in so doing. The achievable reduction is governed primarily by the (somewhat obvious) law known as Amdahl's law, that essentially states that the improvement from parallel processing is determined by its slowest sequential component. Algorithms designed for this class of processor therefore require more care if they are not to unwittingly disrupt the potential gains.

Avoiding Costs

- Defining variables as integers for indexed arrays instead of floating point will result in faster execution.
- Defining structures whose structure length is a multiple of a power of 2 (2,4,8,16 etc.), will allow the compiler to calculate array indexes by shifting a binary index by 1, 2 or more bits to the left, instead of using a multiply instruction will result in faster execution. Adding an otherwise redundant short filler variable to 'pad out' the length of a structure element to say 8 bytes when otherwise it would have been 6 or 7 bytes may reduce overall processing time by a worthwhile amount for very large arrays. See for generated code differences for C as for example.
- Storage defined in terms of bits, when bytes would suffice, may inadvertently involve extremely long path lengths involving bitwise operations instead of more efficient single instruction 'multiple byte' copy instructions. (This does not apply to 'genuine' intentional bitwise operations - used for example instead of multiplication or division by powers of 2 or for TRUE/FALSE flags.)
- Unnecessary use of allocated dynamic storage when static storage would suffice, can increase the processing overhead substantially - both increasing memory requirements and the associated allocation/ deallocation path length overheads for each function call.

- Excessive use of function calls for very simple functions, rather than in-line statements, can also add substantially to instruction path lengths and stack/unstack overheads. For particularly time critical systems that are not also code size sensitive, automatic or manual inline expansion can reduce path length by eliminating all the instructions that call the function and return from it. (A conceptually similar method, loop unrolling, eliminates the instructions required to set up and terminate a loop by, instead; repeating the instructions inside the loop multiple times. This of course eliminates the branch back instruction but may also increase the size of the binary file or, in the case of JIT built code, dynamic memory. Also, care must be taken with this method, that re-calculating addresses for each statement within an unwound indexed loop is not more expensive than incrementing pointers within the former loop would have been. If absolute indexes are used in the generated (or manually created) unwound code, rather than variables, the code created may actually be able to avoid generated pointer arithmetic instructions altogether, using offsets instead).

Memory Management

Whenever memory is *automatically* allocated (for example in HLL programmes, when calling a procedure or when issuing a system call), it is normally released (or 'freed'/ 'deallocated'/ 'deleted') automatically when it is no longer required - thus allowing it to be re-used for another purpose *immediately*. Some memory management can easily be accomplished by the compiler, as in this example. However, when memory is *explicitly* allocated (for example in OOP when "new" is specified for an object), releasing the memory is often left to an asynchronous 'garbage collector' which does not necessarily release the memory at the earliest opportunity (as well as consuming some additional CPU resources deciding if it can be). The current trend nevertheless appears to be towards taking full advantage of this fully automated method, despite the tradeoff in efficiency - because it is claimed that it makes programming easier. Some functional languages are known as 'lazy functional languages' because of the significant use of garbage collection and can consume much more memory as a result.

- Array processing may simplify programming but use of separate statements to sum different elements of the same array(s) may produce code that is not easily optimized and that requires multiple

passes of the arrays that might otherwise have been processed in a single pass. It may also duplicate data if array slicing is used, leading to increased memory usage and copying overhead.

- In OOP, if an object is known to be immutable, it can be copied simply by making a copy of a reference to it instead of copying the entire object. Because a reference (typically only the size of a pointer) is usually much smaller than the object itself, this results in memory savings and a boost in execution speed.

Readability, Trade Offs and Trends

One must be careful, in the pursuit of good coding style, not to over-emphasize efficiency. Frequently, a clean, readable and 'usable' design is much more important than a fast, efficient design that is hard to understand. There are exceptions to this 'rule' (such as embedded systems, where space is tight, and processing power minimal) but these are rarer than one might expect. However, increasingly, for many 'time critical' applications such as air line reservation systems, point-of-sale applications, ATMs (cash-point machines), Airline Guidance systems, Collision avoidance systems and numerous modern web based applications - operating in a real-time environment where speed of response is fundamental - there is little alternative.

Determining if Optimization is Worthwhile

The essential criteria for using optimized code are of course dependent upon the expected use of the algorithm. If it is a new algorithm and is going to be in use for many years and speed is relevant, it is worth spending some time designing the code to be as efficient as possible from the outset.

If an existing algorithm is proving to be too slow or memory is becoming an issue, clearly something must be done to improve it. For the average application, or for one-off applications, avoiding inefficient coding techniques and encouraging the compiler to optimize where possible may be sufficient.

One simple way (at least for mathematicians) to determine whether an optimization is worthwhile is as follows: Let the original time and space requirements (generally in Big-O notation) of the algorithm be O_1 and O_2. Let the new code require N_1 and N_2 time and space respectively. If $N_1N_2 < O_1O_2$, the optimization should be carried out. However, as mentioned above, this may not always be true.

Implications for Algorithmic Efficiency

A recent report, published in December 2007, from Global Action Plan, a UK-based environmental organization found that computer servers are "at least as great a threat to the climate as SUVs or the global aviation industry" drawing attention to the carbon footprint of the IT industry in the UK. According to an Environmental Research Letters report published in September 2008, "Total power used by information technology equipment in data centers represented about 0.5% of world electricity consumption in 2005. When cooling and auxiliary infrastructure are included, that figure is about 1%. The total data center power demand in 2005 is equivalent (in capacity terms) to about seventeen 1000 MW power plants for the world." Some media reports claim that performing two Google searches from a desktop computer can generate about the same amount of carbon dioxide as boiling a kettle for a cup of tea, according to new research; however, the factual accuracy of this comparison is disputed, and the author of the study in question asserts that the two-searches-tea-kettle statistic is a misreading of his work.

Greentouch, a recently established consortium of leading Information and Communications Technology (ICT) industry, academic and non-governmental research experts, has set itself the mission of reducing reduce energy consumption per user by a factor of 1000 from current levels. "A thousand-fold reduction is roughly equivalent to being able to power the world's communications networks, including the Internet, for three years using the same amount of energy that it currently takes to run them for a single day".

The first meeting in February 2010 will establish the organization's five-year plan, first year deliverables and member roles and responsibilities. Intellectual property issues will be addressed and defined in the forum's initial planning meetings.

The conditions for research and the results of that research will be high priority for discussion in the initial phase of the research forum's development. Computers having become increasingly more powerful over the past few decades, emphasis was on a 'brute force' mentality. This may have to be reconsidered in the light of these reports and more effort placed in future on reducing carbon footprints through optimization. It is a timely reminder that algorithmic efficiency is just another aspect of the more general thermodynamic efficiency. The genuine economic benefits of an optimized algorithm are, in any case, that more processing can be done

for the same cost or that useful results can be shown in a more timely manner and ultimately, acted upon sooner.

ALGORITHMIC TRADING

In electronic financial markets, algorithmic trading or automated trading, also known as algo trading, black-box trading or robo trading, is the use of computer programmes for entering trading orders with the computer algorithm deciding on aspects of the order such as the timing, price, or quantity of the order, or in many cases initiating the order without human intervention. Algorithmic Trading is widely used by pension funds, mutual funds, and other buy side (investor driven) institutional traders, to divide large trades into several smaller trades in order to manage market impact, and risk. Sell side traders, such as market makers and some hedge funds, provide liquidity to the market, generating and executing orders automatically. A special class of algorithmic trading is "high-frequency trading" (HFT), in which computers make elaborate decisions to initiate orders based on information that is received electronically, before human traders are capable of processing the information they observe. This has resulted in a dramatic change of the market microstructure, particularly in the way liquidity is provided.

Algorithmic trading may be used in any investment strategy, including market making, inter-market spreading, arbitrage, or pure speculation (including trend following). The investment decision and implementation may be augmented at any stage with algorithmic support or may operate completely automatically ("on auto-pilot"). A third of all EU and US stock trades in 2006 were driven by automatic programmes, or algorithms, according to Boston-based financial services industry research and consulting firm Aite Group. As of 2009, HFT firms account for 73% of all US equity trading volume.

In 2006 at the London Stock Exchange, over 40% of all orders were entered by algo traders, with 60% predicted for 2007. American markets and European markets generally have a higher proportion of algo trades than other markets, and estimates for 2008 range as high as an 80% proportion in some markets.

Foreign exchange markets also have active algo trading (about 25% of orders in 2006). Futures and options markets are considered to be fairly easily integrated into algorithmic trading, with about 20% of options volume expected to be computer generated by 2010. Bond markets are moving

toward more access to algorithmic traders. One of the main issues regarding HFT is the difficulty in determining just how profitable it is. A report released in August 2009 by the TABB Group, a financial services industry research firm, estimated that the 300 securities firms and hedge funds that specialize in this type of trading took in roughly US$21 billion in profits in 2008. Algorithmic and HFT have been the subject of much public debate since the U.S. Securities and Exchange Commission and the Commodity Futures Trading Commission implicated them in the May 6, 2010 Flash Crash, when the Dow Jones Industrial Average suffered its largest intraday point loss ever to that date, though prices quickly recovered.

History

Computerization of the order flow in financial markets began in the early 1970s with some landmarks being the introduction of the New York Stock Exchange's "designated order turnaround" system (DOT, and later SuperDOT) which routed orders electronically to the proper trading post to be executed manually, and the "opening automated reporting system" (OARS) which aided the specialist in determining the market clearing opening price (SOR; Smart Order Routing). Programme trading is defined by the New York Stock Exchange as an order to buy or sell 15 or more stocks valued at over US$1 million total. In practice this means that all programme trades are entered with the aid of a computer. In the 1980s programme trading became widely used in trading between the S&P500 equity and futures markets. In stock index arbitrage a trader buys (or sells) a stock index futures contract such as the S&P 500 futures and sells (or buys) a portfolio of up to 500 stocks (can be a much smaller representative subset) at the NYSE matched against the futures trade. The programme trade at the NYSE would be pre-programmed into a computer to enter the order automatically into the NYSE's electronic order routing system at a time when the futures price and the stock index were far enough apart to make a profit.

At about the same time portfolio insurance was designed to create a synthetic put option on a stock portfolio by dynamically trading stock index futures according to a computer model based on the Black-Scholes option pricing model. Both strategies, often simply lumped together as "programme trading", were blamed by many people (for example by the Brady report) for exacerbating or even starting the 1987 stock market crash. Yet the impact of computer driven trading on stock market crashes is unclear and widely discussed in the academic community. Financial

markets with fully electronic execution and similar electronic communication networks developed in the late 1980s and 1990s. In the U.S., decimalization, which changed the minimum tick size from 1/16 of a dollar (US$0.0625) to US$0.01 per share, may have encouraged algorithmic trading as it changed the market microstructure by permitting smaller differences between the bid and offer prices, decreasing the market-makers' trading advantage, thus increasing market liquidity.

This increased market liquidity led to institutional traders splitting up orders according to computer algorithms in order to execute their orders at a better average price. These average price benchmarks are measured and calculated by computers by applying the time weighted (i.e. unweighted) average price TWAP or more usually by the volume weighted average price VWAP. As more electronic markets opened, other algorithmic trading strategies were introduced.

These strategies are more easily implemented by computers because machines can react more rapidly to temporary mispricing and examine prices from several markets simultaneously. For example Stealth (developed by the *Deutsche Bank*), Sniper and Guerilla (developed by *Credit Suisse*), arbitrage, statistical arbitrage, trend following, and mean reversion. This type of trading is what is driving the new demand for Low Latency Proximity Hosting and Global Exchange Connectivity. It is imperative to understand what is latency when putting together a strategy for electronic trading. Latency refers to the delay between the transmission of information from a source and the reception of the information at a destination. Latency has as a lower bound the speed of light; this corresponds to a few microseconds per 1,000 kilometers of optical fibre. Any signal regenerating or routing equipment will introduce greater latency than this speed-of-light baseline.

Strategies

Trend Following

Trend following is an investment strategy that tries to take advantage of long-term moves that seem to play out in various markets. The system aims to work on the market trend mechanism and take benefit from both sides of the market enjoying the profits from the *ups* and *downs* of the stock or futures markets. Traders who use this approach can use current market price calculation, moving averages and channel breakouts to determine the general direction of the market and to generate trade signals. Traders

who subscribe to a trend following strategy do not aim to forecast or predict specific price levels; they simply jump on the trend and ride it.

Pair Trading

The pairs trade or pair trading is a market neutral trading strategy enabling traders to profit from virtually any market conditions: uptrend, downtrend, or sidewise movement. This trading strategy is categorized as a statistical arbitrage and convergence trading strategy.

Delta Neutral Strategies

In finance, delta neutral describes a portfolio of related financial securities, in which the portfolio value remains unchanged due to small changes in the value of the underlying security.

Such a portfolio typically contains options and their corresponding underlying securities such that positive and negative delta components offset, resulting in the portfolio's value being relatively insensitive to changes in the value of the underlying security.

Arbitrage

In economics and finance, arbitrage is the practice of taking advantage of a price difference between two or more markets: striking a combination of matching deals that capitalize upon the imbalance, the profit being the difference between the market prices. When used by academics, an arbitrage is a transaction that involves no negative cash flow at any probabilistic or temporal state and a positive cash flow in at least one state; in simple terms, it is the possibility of a risk-free profit at zero cost.

Conditions for Arbitrage

Arbitrage is possible when one of three conditions is met:

1. The same asset does not trade at the same price on all markets (the "law of one price").
2. Two assets with identical cash flows do not trade at the same price.
3. An asset with a known price in the future does not today trade at its future price discounted at the risk-free interest rate (or, the asset does not have negligible costs of storage; as such, for example, this condition holds for grain but not for securities).

Arbitrage is not simply the act of buying a product in one market and selling it in another for a higher price at some later time. The transactions

must occur *simultaneously* to avoid exposure to market risk, or the risk that prices may change on one market before both transactions are complete. In practical terms, this is generally only possible with securities and financial products which can be traded electronically, and even then, when each leg of the trade is executed the prices in the market may have moved. Missing one of the legs of the trade (and subsequently having to trade it soon after at a worse price) is called 'execution risk' or more specifically 'leg risk'. In the simplest example, any good sold in one market should sell for the same price in another. Traders may, for example, find that the price of wheat is lower in agricultural regions than in cities, purchase the good, and transport it to another region to sell at a higher price. This type of price arbitrage is the most common, but this simple example ignores the cost of transport, storage, risk, and other factors. "True" arbitrage requires that there be no market risk involved. Where securities are traded on more than one exchange, arbitrage occurs by simultaneously buying in one and selling on the other.

Mean Reversion

Mean reversion is a mathematical methodology sometimes used for stock investing, but it can be applied to other processes. In general terms the idea is that both a stock's high and low prices are temporary, and that a stock's price will tend to have an average price over time. Mean reversion involves first identifying the trading range for a stock, and then computing the average price using analytical techniques as it relates to assets, earnings, etc. When the current market price is less than the average price, the stock is considered attractive for purchase, with the expectation that the price will rise. When the current market price is above the average price, the market price is expected to fall. In other words, deviations from the average price are expected to revert to the average. The Standard deviation of the most recent prices (e.g., the last 20) is often used as a buy or sell indicator. Stock reporting services (such as Yahoo! Finance, MS Investor, Morningstar, etc.), commonly offer moving averages for periods such as 50 and 100 days. While reporting services provide the averages, identifying the high and low prices for the study period is still necessary. Mean reversion has the appearance of a more scientific method of choosing stock buy and sell points than charting, because precise numerical values are derived from historical data to identify the buy/sell values, rather than trying to interpret price movements using charts (charting, also known as technical analysis).

Scalping

Scalping (trading) is a method of arbitrage of small price gaps created by the bid-ask spread. Scalpers attempt to act like traditional market makers or specialists. To *make the spread* means to buy at the Bid price and sell at the Ask price, to gain the bid/ask difference. This procedure allows for profit even when the bid and ask do not move at all, as long as there are traders who are willing to take market prices. It normally involves establishing and liquidating a position quickly, usually within minutes or even seconds. The role of a scalper is actually the role of market makers or specialists who are to maintain the liquidity and order flow of a product of a market. A market maker is basically a specialized scalper. The volume a market maker trades are many times more than the average individual scalpers. A market maker has a sophisticated trading system to monitor trading activity. However, a market maker is bound by strict exchange rules while the individual trader is not. For instance, NASDAQ requires each market maker to post at least one bid and one ask at some price level, so as to maintain a two-sided market for each stock represented.

Transaction Cost Reduction

Most strategies referred to as Algorithmic Trading (as well as algorithmic liquidity seeking) fall into the cost-reduction category. Large orders are broken down into several smaller orders and entered into the market over time. This basic strategy is called "iceberging". The success of this strategy may be measured by the average purchase price against the VWAP for the market over that time period. One algorithm designed to find hidden orders or icebergs is called "Stealth". Most of these strategies were first documented in 'Optimal Trading Strategies' by Robert Kissell.

Strategies that Only Pertain to Dark Pools

Recently, HFT, which comprises a broad set of buy-side as well as market making sell side traders, has become more prominent and controversial. These algorithms or techniques are commonly given names such as "Stealth" (developed by the Deutsche Bank), "Iceberg", "Dagger", "Guerrilla", "Sniper", "BASOR" (developed by Quod Financial) and "Sniffer". Yet are at their core quite simple mathematical constructs. Dark pools are alternative electronic stock exchanges where trading takes place anonymously, with most orders hidden or "iceberged." Gamers or "sharks" sniff out large orders by "pinging" small market orders to buy and sell.

When several small orders are filled the sharks may have discovered the presence of a large iceberged order. "Now it's an arms race," said Andrew Lo, director of the Massachusetts Institute of Technology's Laboratory for Financial Engineering. "Everyone is building more sophisticated algorithms, and the more competition exists, the smaller the profits." One of the unintended adverse effects of algorithmic trading, has been the dramatic increase in the volume of trade allocations and settlements, as well as the transaction settlement costs associated with them. Since 2004, there have been a number of technological advances and service providers by individuals like Scott Kurland, who have built solutions for aggregating trades executed across algorithms, in order to counter these rising settlement costs.

High-Frequency Trading

In the U.S., high-frequency trading (HFT) firms represent 2% of the approximately 20,000 firms operating today, but account for 73% of all equity trading volume. As of the first quarter in 2009, total assets under management for hedge funds with HFT strategies were US$141 billion, down about 21% from their high. The HFT strategy was first made successful by Renaissance Technologies. High-frequency funds started to become especially popular in 2007 and 2008. Many HFT firms are market makers and provide liquidity to the market which has lowered volatility and helped narrow Bid-offer spreads making trading and investing cheaper for other market participants. HFT has been a subject of intense public focus since the U.S. Securities and Exchange Commission and the Commodity Futures Trading Commission implicated both algorithmic and HFT in the May 6, 2010 Flash Crash. High-frequency trading is quantitative trading that is characterized by short portfolio holding periods. There are four key categories of HFT strategies: market-making based on order flow, market-making based on tick data information, event arbitrage and statistical arbitrage. All portfolio-allocation decisions are made by computerized quantitative models. The success of HFT strategies is largely driven by their ability to simultaneously process volumes of information, something ordinary human traders cannot do.

Market Making

Market making is a set of HFT strategies that involves placing a limit order to sell (or offer) above the current market price or a buy limit order (or bid) below the current price in order to benefit from the bid-ask spread.

Automated Trading Desk, which was bought by Citigroup in July 2007, has been an active market maker, accounting for about 6% of total volume on both NASDAQ and the New York Stock Exchange.

Statistical Arbitrage

Another set of HFT strategies is classical arbitrage strategy might involve several securities such as covered interest rate parity in the foreign exchange market which gives a relation between the prices of a domestic bond, a bond denominated in a foreign currency, the spot price of the currency, and the price of a forward contract on the currency. If the market prices are sufficiently different from those implied in the model to cover transactions cost then four transactions can be made to guarantee a risk-free profit. HFT allows similar arbitrages using models of greater complexity involving many more than 4 securities. The TABB Group estimates that annual aggregate profits of low latency arbitrage strategies currently exceed US$21 billion. A wide range of statistical arbitrage strategies have been developed whereby trading decisions are made on the basis of deviations from statistically significant relationships. Like market-making strategies, statistical arbitrage can be applied in all asset classes.

Event Arbitrage

A subset of risk, merger, convertible, or distressed securities arbitrage that counts on a specific event, such as a contract signing, regulatory approval, judicial decision, etc., to change the price or rate relationship of two or more financial instruments and permit the arbitrageur to earn a profit. Merger arbitrage also called risk arbitrage would be an example of this. Merger arbitrage generally consists of buying the stock of a company that is the target of a takeover while shorting the stock of the acquiring company. Usually the market price of the target company is less than the price offered by the acquiring company. The spread between these two prices depends mainly on the probability and the timing of the takeover being completed as well as the prevailing level of interest rates. The bet in a merger arbitrage is that such a spread will eventually be zero, if and when the takeover is completed. The risk is that the deal "breaks" and the spread massively widens.

Low-Latency Trading

HFT is often confused with low-latency trading that uses computers that execute trades within milliseconds, or "with extremely low latency"

in the jargon of the trade. Low-latency trading is highly dependent on ultra-low latency networks. They profit by providing information, such as competing bids and offers, to their algorithms microseconds faster than their competitors. The revolutionary advance in speed has led to the need for firms to have a real-time, colocated trading platform in order to benefit from implementing high-frequency strategies. Strategies are constantly altered to reflect the subtle changes in the market as well as to combat the threat of the strategy being reverse engineered by competitors. There is also a very strong pressure to continuously add features or improvements to a particular algorithm, such as client specific modifications and various performance enhancing changes (regarding benchmark trading performance, cost reduction for the trading firm or a range of other implementations). This is due to the evolutionary nature of algorithmic trading strategies - they must be able to adapt and trade intelligently, regardless of market conditions, which involves being flexible enough to withstand a vast array of market scenarios. As a result, a significant proportion of net revenue from firms is spent on the R&D of these autonomous trading systems.

Strategy Implementation

Most of the algorithmic strategies are implemented using modern programming languages, although some still implement strategies designed in spreadsheets. Basic models can rely on as little as a linear regression, while more complex game-theoretic and pattern recognition or predictive models can also be used to initiate trading. Neural networks and genetic programming have been used to create these models.

Issues and Developments

Algorithmic trading has been shown to substantially improve market liquidity among other benefits. However, improvements in productivity brought by algorithmic trading have been opposed by human brokers and traders facing stiff competition from computers.

Concerns

"The downside with these systems is their black box-ness," Mr. Williams said. "Traders have intuitive senses of how the world works. But with these systems you pour in a bunch of numbers, and something comes out the other end, and it's not always intuitive or clear why the black box latched onto certain data or relationships." "The Financial Services

Authority has been keeping a watchful eye on the development of black box trading. In its annual report the regulator remarked on the great benefits of efficiency that new technology is bringing to the market. But it also pointed out that 'greater reliance on sophisticated technology and modelling brings with it a greater risk that systems failure can result in business interruption'." UK Treasury minister Lord Myners has warned that companies could become the "playthings" of speculators because of automatic high-frequency trading. Lord Myners said the process risked destroying the relationship between an investor and a company. Other issues include the technical problem of latency or the delay in getting quotes to traders, security and the possibility of a complete system breakdown leading to a market crash. "Goldman spends tens of millions of dollars on this stuff. They have more people working in their technology area than people on the trading desk...The nature of the markets has changed dramatically." Algorithmic and HTF were implicated in the May 6, 2010 Flash Crash, when the Dow Jones Industrial Average plunged about 600 points only to recover those losses within minutes. At the time, it was the second largest point swing, 1,010.14 points, and the biggest one-day point decline, 998.5 points, on an intraday basis in Dow Jones Industrial Average history.

Recent Developments

Financial market news is now being formatted by firms such as Need To Know News, Thomson Reuters, Dow Jones, and Bloomberg, to be read and traded on via algorithms. "Computers are now being used to generate news stories about company earnings results or economic statistics as they are released. And this almost instantaneous information forms a direct feed into other computers which trade on the news." The algorithms do not simply trade on simple news stories but also interpret more difficult to understand news. Some firms are also attempting to automatically assign *sentiment* (deciding if the news is good or bad) to news stories so that automated trading can work directly on the news story. "Increasingly, people are looking at all forms of news and building their own indicators around it in a semi-structured way," as they constantly seek out new trading advantages said Rob Passarella, global director of strategy at Dow Jones Enterprise Media Group. His firm provides both a low latency news feed and news analytics for traders.

Passarella also pointed to new academic research being conducted on the degree to which frequent Google searches on various stocks can serve

as trading indicators, the potential impact of various phrases and words that may appear in Securities and Exchange Commission statements and the latest wave of online communities devoted to stock trading topics. "Markets are by their very nature conversations, having grown out of coffee houses and taverns", he said.

So the way conversations get created in a digital society will be used to convert news into trades, as well, Passarella said. "There is a real interest in moving the process of interpreting news from the humans to the machines" says Kirsti Suutari, global business manager of algorithmic trading at Reuters. "More of our customers are finding ways to use news content to make money."

An example of the importance of news reporting speed to algorithmic traders was an advertising campaign by Dow Jones (appearances included page W15 of the Wall Street Journal, on March 1, 2008) claiming that their service had beaten other news services by 2 seconds in reporting an interest rate cut by the Bank of England. In July 2007, Citigroup, which had already developed its own trading algorithms, paid $680 million for Automated Trading Desk, a 19-year-old firm that trades about 200 million shares a day. Citigroup had previously bought Lava Trading and OnTrade Inc.

Technical Design

The technical designs of such systems are not standardized. Conceptually, the design can be divided into logical units:

1. The data stream unit (the part of the systems that receives data (e.g. quotes, news) from external sources).
2. The decision or strategy unit
3. The execution unit.

With the wide use of social networks, some systems implement scanning or screening technologies to read posts of users extracting human sentiment and influence the trading strategies.

Effects

Though its development may have been prompted by decreasing trade sizes caused by decimalization, algorithmic trading has reduced trade sizes further. Jobs once done by human traders are being switched to computers. The speeds of computer connections, measured in milliseconds and even microseconds, have become very important. More

fully automated markets such as NASDAQ, Direct Edge and BATS, in the US, have gained market share from less automated markets such as the NYSE. Economies of scale in electronic trading have contributed to lowering commissions and trade processing fees, and contributed to international mergers and consolidation of financial exchanges. Competition is developing among exchanges for the fastest processing times for completing trades. For example, in June 2007, the London Stock Exchange launched a new system called TradElect that promises an average 10 millisecond turnaround time from placing an order to final confirmation and can process 3,000 orders per second. Since then, competitive exchanges have continued to reduce latency with turnaround times of 3 milliseconds available. This is of great importance to high-frequency traders because they have to attempt to pinpoint the consistent and probable performance ranges of given financial instruments. These professionals are often dealing in versions of stock index funds like the E-mini S&Ps because they seek consistency and risk-mitigation along with top performance. They must filter market data to work into their software programming so that there is the lowest latency and highest liquidity at the time for placing stop-losses and/or taking profits. With high volatility in these markets, this becomes a complex and potentially nerve-wracking endeavor, where a small mistake can lead to a large loss. Absolute frequency data play into the development of the trader's pre-programmed instructions. Spending on computers and software in the financial industry increased to $26.4 billion in 2005.

Communication Standards

Algorithmic trades require communicating considerably more parameters than traditional market and limit orders. A trader on one end (the "buy side") must enable their trading system (often called an "Order Management System" or "Execution Management System") to understand a constantly proliferating flow of new algorithmic order types. The R&D and other costs to construct complex new algorithmic orders types, along with the execution infrastructure, and marketing costs to distribute them, are fairly substantial. What was needed was a way that marketers (the "sell side") could express algo orders electronically such that buy-side traders could just drop the new order types into their system and be ready to trade them without constant coding custom new order entry screens each time. FIX Protocol LTD http://www.fixprotocol.org is a trade association that publishes free, open standards in the securities trading area. The FIX

language was originally created by Fidelity Investments, and the association Members include virtually all large and many midsized and smaller broker dealers, money center banks, institutional investors, mutual funds, etc. This institution dominates standard setting in the pretrade and trade areas of security transactions. In 2006-2007 several members got together and published a draft XML standard for expressing algorithmic order types. The standard is called FIX Algorithmic Trading Definition Language (FIXatdl). The first version of this standard, 1.0 was not widely adopted due to limitations in the specification, but the second version, 1.1 (released in March 2010) is expected to achieve broad adoption and in the process dramatically reduce time-to-market and costs associated with distributing new algorithms.

7

Algorithms in Informed Search and Hill Climbing in AI

AI - POPULAR SEARCH ALGORITHMS

Searching is the universal technique of problem solving in AI. There are some single-player games such as tile games, Sudoku, crossword, etc. The search algorithms help you to search for a particular position in such games.

Single Agent Pathfinding Problems

The games such as 3X3 eight-tile, 4X4 fifteen-tile, and 5X5 twenty four tile puzzles are single-agent-path-finding challenges. They consist of a matrix of tiles with a blank tile. The player is required to arrange the tiles by sliding a tile either vertically or horizontally into a blank space with the aim of accomplishing some objective.

The other examples of single agent pathfinding problems are Travelling Salesman Problem, Rubik's Cube, and Theorem Proving.

Search Terminology

- Problem Space " It is the environment in which the search takes place. (A set of states and set of operators to change those states)
- Problem Instance " It is Initial state + Goal state.
- Problem Space Graph " It represents problem state. States are shown by nodes and operators are shown by edges.
- Depth of a problem " Length of a shortest path or shortest sequence of operators from Initial State to goal state.
- Space Complexity " The maximum number of nodes that are stored in memory.

- Time Complexity " The maximum number of nodes that are created.
- Admissibility " A property of an algorithm to always find an optimal solution.
- Branching Factor " The average number of child nodes in the problem space graph.
- Depth " Length of the shortest path from initial state to goal state.

Brute-Force Search Strategies

They are most simple, as they do not need any domain-specific knowledge. They work fine with small number of possible states.

Requirements "

- State description
- A set of valid operators
- Initial state
- Goal state description

Breadth-First Search

It starts from the root node, explores the neighboring nodes first and moves towards the next level neighbors. It generates one tree at a time until the solution is found. It can be implemented using FIFO queue data structure. This method provides shortest path to the solution.

If branching factor (average number of child nodes for a given node) = b and depth = d, then number of nodes at level d = b^d.

The total no of nodes created in worst case is $b + b^2 + b^3 + \ldots + b^d$.

Disadvantage " Since each level of nodes is saved for creating next one, it consumes a lot of memory space. Space requirement to store nodes is exponential.

Its complexity depends on the number of nodes. It can check duplicate nodes.

Depth-First Search

It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order. As the nodes on the single path are stored in each iteration from root to leaf node, the space requirement to store nodes is linear. With branching factor b and depth as m, the storage space is bm.

Disadvantage " This algorithm may not terminate and go on infinitely on one path. The solution to this issue is to choose a cut-off depth. If the ideal cut-off is *d*, and if chosen cut-off is lesser than *d*, then this algorithm may fail. If chosen cut-off is more than *d*, then execution time increases.

Its complexity depends on the number of paths. It cannot check duplicate nodes.

Bidirectional Search

It searches forward from initial state and backward from goal state till both meet to identify a common state.

The path from initial state is concatenated with the inverse path from the goal state. Each search is done only up to half of the total path.

Uniform Cost Search

Sorting is done in increasing cost of the path to a node. It always expands the least cost node. It is identical to Breadth First search if each transition has the same cost.

It explores paths in the increasing order of cost.

Disadvantage " There can be multiple long paths with the cost d" C*. Uniform Cost search must explore them all.

Iterative Deepening Depth-First Search

It performs depth-first search to level 1, starts over, executes a complete depth-first search to level 2, and continues in such way till the solution is found.

It never creates a node until all lower nodes are generated. It only saves a stack of nodes. The algorithm ends when it finds a solution at depth *d*. The number of nodes created at depth *d* is b^d and at depth *d-1* is b^{d-1}.

Informed (Heuristic) Search Strategies

To solve large problems with large number of possible states, problem-specific knowledge needs to be added to increase the efficiency of search algorithms.

Heuristic Evaluation Functions

They calculate the cost of optimal path between two states. A heuristic function for sliding-tiles games is computed by counting number of moves

that each tile makes from its goal state and adding these number of moves for all tiles.

Pure Heuristic Search

It expands nodes in the order of their heuristic values. It creates two lists, a closed list for the already expanded nodes and an open list for the created but unexpanded nodes.

In each iteration, a node with a minimum heuristic value is expanded, all its child nodes are created and placed in the closed list. Then, the heuristic function is applied to the child nodes and they are placed in the open list according to their heuristic value. The shorter paths are saved and the longer ones are disposed.

*A * Search*

It is best-known form of Best First search. It avoids expanding paths that are already expensive, but expands most promising paths first.

f(n) = g(n) + h(n), where

- g(n) the cost (so far) to reach the node
- h(n) estimated cost to get from the node to the goal
- f(n) estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

Greedy Best First Search

It expands the node that is estimated to be closest to goal. It expands nodes based on f(n) = h(n). It is implemented using priority queue.

Disadvantage " It can get stuck in loops. It is not optimal.

Local Search Algorithms

They start from a prospective solution and then move to a neighboring solution. They can return a valid solution even if it is interrupted at any time before they end.

Hill-Climbing Search

It is an iterative algorithm that starts with an arbitrary solution to a problem and attempts to find a better solution by changing a single element of the solution incrementally. If the change produces a better solution, an incremental change is taken as a new solution. This process is repeated until there are no further improvements.

function Hill-Climbing (problem), returns a state that is a local maximum.

```
inputs: problem, a problem
local variables: current, a node
 neighbor, a node
current <-Make_Node(Initial-State[problem])
loop
do neighbor <- a highest_valued successor of current
if Value[neighbor] d" Value[current] then
return State[current]
current <- neighbor
end
```

Disadvantage " This algorithm is neither complete, nor optimal.

Local Beam Search

In this algorithm, it holds k number of states at any given time. At the start, these states are generated randomly. The successors of these k states are computed with the help of objective function. If any of these successors is the maximum value of the objective function, then the algorithm stops.

Otherwise the (initial k states and k number of successors of the states = 2k) states are placed in a pool. The pool is then sorted numerically. The highest k states are selected as new initial states. This process continues until a maximum value is reached.

function BeamSearch(*problem, k*), returns a solution state.

```
start with k randomly generated states
loop
generate all successors of all k states
if any of the states = solution, then return the
state
else select the k best successors
end
```

Simulated Annealing

Annealing is the process of heating and cooling a metal to change its internal structure for modifying its physical properties. When the metal cools, its new structure is seized, and the metal retains its newly obtained properties. In simulated annealing process, the temperature is kept variable.

BEST FIRST SEARCH ALGORITHM IN AI | CONCEPT, IMPLEMENTATION, ADVANTAGES, DISADVANTAGES

Most of the AI advancements that have caught our attention in the past have been the ability of the machine to beat humans at playing games. Be it 'Deep Blue' defeating the legendary Gary Kasparov in Chess in 1997 or 'Alpha Go' defeating Lee Sudol in 2016, the potential of AI to mimic and surpass human mental capabilities has exponentially increased over time.

Search algorithms form the core of such Artificial Intelligence programs. And while we may be inclined to think that this has limited applicability only in areas of gaming and puzzle-solving, such algorithms are in fact used in many more AI areas like route and cost optimizations, action planning, knowledge mining, robotics, autonomous driving, computational biology, software and hardware verification, theorem proving etc. In a way, many AI problems can be modelled as a search problem where the task is to reach the goal from the initial state via state transformation rules. So the search space is defined as a graph (or a tree) and the aim is to reach the goal from the initial state via the shortest path, in terms of cost, length, a combination of both etc.

All search methods can be broadly classified into two categories:

1. Uninformed (or Exhaustive or Blind) methods, where the search is carried out without any additional information that is already provided in the problem statement. Some examples include Breadth-First Search, Depth First Search etc.
2. Informed (or Heuristic) methods, where the search is carried out by using additional information to determine the next step towards finding the solution. BFS is an example of such algorithms

Informed search methods are more efficient, low in cost and high in performance as compared to uninformed search methods.

What is Best First Search?

If we consider searching as a form of traversal in a graph, an uninformed search algorithm would blindly traverse to the next node in a given manner without considering the cost associated with that step. An informed search, like BFS, on the other hand, would use an evaluation function to decide which among the various available nodes is the most promising (or 'BEST') before traversing to that node.

BFS uses the concept of a Priority queue and heuristic search. To search the graph space, the BFS method uses two lists for tracking the traversal. An 'Open' list that keeps track of the current 'immediate' nodes available for traversal and a 'CLOSED' list that keeps track of the nodes already traversed.

Best First Search Algorithm

1. Create 2 empty lists: OPEN and CLOSED
2. Start from the initial node (say N) and put it in the 'ordered' OPEN list
3. Repeat the next steps until the GOAL node is reached

1. If the OPEN list is empty, then EXIT the loop returning 'False'
2. Select the first/top node (say N) in the OPEN list and move it to the CLOSED list. Also, capture the information of the parent node
3. If N is a GOAL node, then move the node to the Closed list and exit the loop returning 'True'. The solution can be found by backtracking the path
4. If N is not the GOAL node, expand node N to generate the 'immediate' next nodes linked to node N and add all those to the OPEN list
5. Reorder the nodes in the OPEN list in ascending order according to an evaluation function f(n)

This algorithm will traverse the shortest path first in the queue. The time complexity of the algorithm is given by O(n*logn).

Variants of Best First Search

The two variants of BFS are Greedy Best First Search and A* Best First Search. Greedy BFS makes use of the Heuristic function and search and allows us to take advantage of both algorithms.

There are various ways to identify the 'BEST' node for traversal and accordingly there are various flavours of BFS algorithm with different heuristic evaluation functions f(n). We will cover the two most popular versions of the algorithm in this blog, namely Greedy Best First Search and A* Best First Search.

Let's say we want to drive from city S to city E in the shortest possible road distance, and we want to do it in the fastest way, by exploring the least number of cities along the way, i.e. the least number of steps.

Whenever we arrive at an intermediate city, we get to know the air/ flight distance from that city to our goal city E.

This distance is an approximation of how close we are to the goal from a given node and is denoted by the heuristic function h(n). This heuristic value is mentioned within each node. However, note that this is not always equal to the actual road distance, as the road may have many curves while moving up a hill, and more.

Also, when we travel from one node to the other, we get to know the actual road distance between the current city and the immediate next city on the way which is mentioned over the paths in the given figure. The sum of the distance from the start city to each of these immediate next cities is denoted by the function g(n).

At any point, the decision on which city to go to next is governed by our evaluation function. The city which gives the least value for this evaluation function will be explored first.

The only difference between Greedy BFS and A* BFS is in the evaluation function. For Greedy BFS the evaluation function is f(n) = h(n) while for A* the evaluation function is f(n) = g(n) + h(n).

Essentially, since A* is more optimal of the two approaches as it also takes into consideration the total distance travelled so far i.e. g(n).

Advantages and Disadvantages of Best First Search

Advantages:

1. Can switch between BFS and DFS, thus gaining the advantages of both.
2. More efficient when compared to DFS.

Disadvantages:

1. Chances of getting stuck in a loop are higher.

Try changing the graph and see how the algorithms perform on them. Leave your comments below for any doubts. Don't forget to check out popular free Artificial Intelligence courses to upskill in the domain.

AO* SEARCH(GRAPH): CONCEPT, ALGORITHM, IMPLEMENTATION, ADVANTAGES, DISADVANTAGES

The Depth first search and Breadth first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph.

AO* Search: (And-Or) Graph

The Depth first search and Breadth first search given earlier for OR trees or graphs can be easily adopted by AND-OR graph. The main difference lies in the way termination conditions are determined, since all goals following an AND nodes must be realized; where as a single goal node following an OR node will do. So for this purpose we are using AO* algorithm.

Like A* algorithm here we will use two arrays and one heuristic function.

OPEN: It contains the nodes that has been traversed but yet not been marked solvable or unsolvable.

CLOSE: It contains the nodes that have already been processed.

6 7:The distance from current node to goal node.

Algorithm:

Step 1: Place the starting node into OPEN.

Step 2: Compute the most promising solution tree say T0.

Step 3: Select a node n that is both on OPEN and a member of T0. Remove it from OPEN and place it in

CLOSE

Step 4: If n is the terminal goal node then leveled n as solved and leveled all the ancestors of n as solved. If the starting node is marked as solved then success and exit.

Step 5: If n is not a solvable node, then mark n as unsolvable. If starting node is marked as unsolvable, then return failure and exit.

Step 6: Expand n. Find all its successors and find their h (n) value, push them into OPEN.

Step 7: Return to Step 2.

Step 8: Exit.

Implementation

Let us take the following example to implement the AO* algorithm.

Step 1:

In the above graph, the solvable nodes are A, B, C, D, E, F and the unsolvable nodes are G, H. Take A as the starting node. So place A into OPEN.

Advantages: It is an optimal algorithm. If traverse according to the ordering of nodes. It can be used for both OR and AND graph.

Disadvantages: Sometimes for unsolvable nodes, it can't find the optimal path. Its complexity is than other algorithms.

DEFINE BEAM SEARCH

Beam search is a heuristic search algorithm that explores a graph by expanding the most optimistic node in a limited set. Beam search is an optimization of *best-first search* that reduces its memory requirements.

Best-first search is a graph search that orders all partial solutions according to some heuristic. But in beam search, only a predetermined number of best partial solutions are kept as candidates. Therefore, it is a *greedy* algorithm.

Beam search uses *breadth-first search* to build its search tree. At each level of the tree, it generates all successors of the states at the current level, sorting them in increasing order of heuristic cost. However, it only stores a predetermined number (â), of best states at each level called the *beamwidth*. Only those states are expanded next.

The greater the beam width, the fewer states are pruned. No states are pruned with infinite beam width, and beam search is identical to breadth-first search. The beamwidth bounds the memory required to perform the search. Since a goal state could potentially be pruned, beam search sacrifices completeness (the guarantee that an algorithm will terminate with a solution if one exists). Beam search is not optimal, which means there is no guarantee that it will find the best solution.

In general, beam search returns the first solution found. Once reaching the configured maximum search depth (i.e., translation length), the algorithm will evaluate the solutions found during a search at various depths and return the best one that has the highest probability.

The beam width can either be *fixed* or *variable.* One approach that uses a variable beam width starts with the width at a minimum. If no solution is found, the beam is widened, and the procedure is repeated.

Components of Beam Search

A beam search takes three components as its input:

1. A problem to be solved,
2. A set of heuristic rules for pruning,

3. And a memory with a limited available capacity.

The problem is the problem to be solved, usually represented as a graph, and contains a set of nodes in which one or more of the nodes represents a goal. The set of heuristic rules are rules specific to the problem domain and prune unfavorable nodes from memory regarding the problem domain.

The memory is where the *"beam"* is stored, memory is full, and a node is to be added to the beam, the most costly node will be deleted, such that the memory limit is not exceeded.

Beam Search Algorithm

The following algorithm for a beam search, as a modified best-first search, is adapted from Zhang's 1999:

```
1. beamSearch(problemSet, ruleSet, memorySize)
2.  openMemory = new memory of size memorySize
3.  nodeList = problemSet.listOfNodes
4.  node = root or initial search node
5.  Add node to openMemory;
6.  while (node is not a goal node)
7.  Delete node from openMemory;
8.  Expand node and obtain its children, evaluate those children;
9.  If a child node is pruned according to a rule in ruleSet, delete it;
10.  Place remaining, non-pruned children into openMemory;
11. If memory is full and has no room for new nodes, remove the worst
12.   node, determined by ruleSet, in openMemory;
13.  node = the least costly node in openMemory;
```

Uses of Beam Search

A beam search is most often used to maintain tractability in large systems with insufficient memory to store the entire search tree. For example,

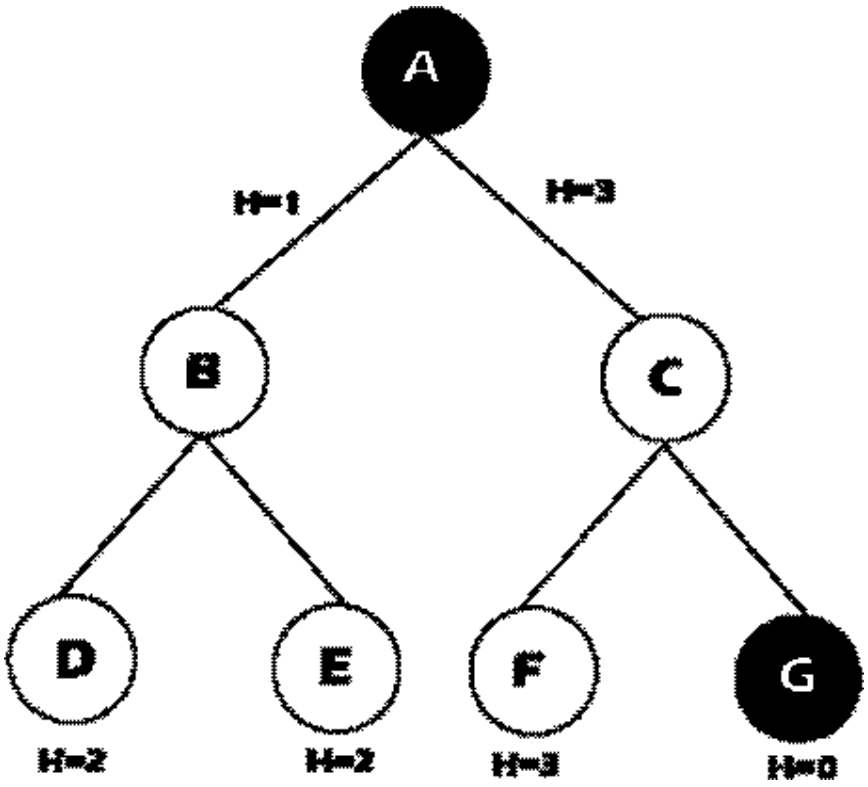
- It has been used in many machine translation systems.
- Each part is processed to select the best translation, and many different ways of translating the words appear.
- According to their sentence structures, the top best translations are kept, and the rest are discarded. The translator then evaluates the

translations according to a given criterion, choosing the translation which best keeps the goals.

- The first use of a beam search was in the *Harpy Speech Recognition System*, CMU 1976.

Drawbacks of Beam Search

Here is a drawback of the Beam Search with an example:



- In general, the Beam Search Algorithm is not *complete*. Despite these disadvantages, beam search has found success in the practical areas of *speech recognition, vision, planning,* and *machine learning.*
- The main disadvantages of a beam search are that the search may not result in an optimal goal and may not even reach a goal at all after given unlimited time and memory when there is a path from the start node to the goal node.
- The beam search algorithm terminates for two cases: a required goal node is reached, or a goal node is not reached, and there are no nodes left to be explored.

- A more accurate heuristic function and a larger beam width can improve Beam Search's chances of finding the goal.

For example, let's take the value of ß = 2 for the tree shown below. So, follow the following steps to find the goal node.

Step 1: OPEN= {A}

Step 2: OPEN= {B, C}

Step 3: OPEN= {D, E}

Step 4: OPEN= {E}

Step 5: OPEN= { }

The open set becomes empty without finding the goal node.

Beam Search Optimality

The Beam Search algorithm is not complete in some cases. Therefore it is also not guaranteed to be optimal. It can happen because of these reasons:

- The beam width and an inaccurate heuristic function may cause the algorithm to miss expanding the shortest path.
- A more precise heuristic function and a larger beam width can make Beam Search more likely to find the optimal path to the goal.

For example, we have a tree with heuristic values shown below:

Follow the following steps to find the path for the goal node.

Step 1: OPEN= {A}

Step 2: OPEN= {B, C}

Step 3: OPEN= {C, E}

Step 4: OPEN= {F, E}

Step 5: OPEN= {G, E}

Step 6: Found the goal node {G}, now stop.

Path: A-> C-> F-> G

But the *Optimal Path* is A-> D-> G

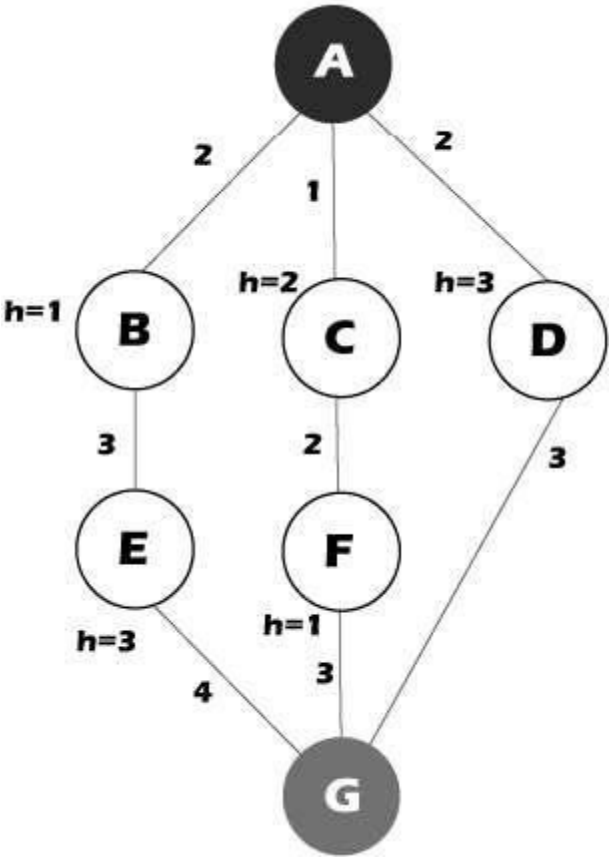
Time Complexity of Beam Search

The time complexity of the Beam Search algorithm depends on the following things, such as:

- The accuracy of the heuristic function.
- In the worst case, the heuristic function leads Beam Search to the deepest level in the search tree.

- The worst-case time = *O(B*m)*

B is the beam width, and *m* is the maximum depth of any path in the search tree.



Space Complexity of Beam Search

The space complexity of the Beam Search algorithm depends on the following things, such as:

- Beam Search's memory consumption is its most desirable trait.
- Since the algorithm only stores B nodes at each level in the search tree.
- The worst-case space complexity = *O(B*m)*

B is the beam width, and m is the maximum depth of any path in the search tree.

- This linear memory consumption allows Beam Search to probe very deeply into large search spaces and potentially find solutions that other algorithms cannot reach.

Variants of Beam Search

Beam search has been made complete by combining it with *depth-first search*, resulting in *beam stack search* and *depth-first beam search.* With limited discrepancy search, *beam search results in limited discrepancy backtracking* (BULB).

The resulting search algorithms are anytime algorithms that find reasonable but likely sub-optimal solutions quickly, like beam search, then backtrack and continue to find improved solutions until convergence to an optimal solution.

In the context of a local search, we call *local beam search* a specific algorithm that begins selecting â generated states. Then, for each level of the search tree, it always considers â new states among all the possible successors of the current ones until it reaches a goal.

Since local beam search often ends up on local maxima, a standard solution is to choose the next â states in a random way, with a probability dependent on the heuristic evaluation of the states. This kind of search is called *stochastic beam search.*

Beam search

In computer science, beam search is a heuristic search algorithm that explores a graph by expanding the most promising node in a limited set. Beam search is an optimization of best-first search that reduces its memory requirements. Best-first search is a graph search which orders all partial solutions (states) according to some heuristic. But in beam search, only a predetermined number of best partial solutions are kept as candidates. It is thus a greedy algorithm.

The term "beam search" was coined by Raj Reddy of Carnegie Mellon University in 1977.

Details

Beam search uses breadth-first search to build its search tree. At each level of the tree, it generates all successors of the states at the current level, sorting them in increasing order of heuristic cost. However, it only stores a predetermined number of best states at each level (called the beam

width). Only those states are expanded next. The greater the beam width, the fewer states are pruned. With an infinite beam width, no states are pruned and beam search is identical to breadth-first search. The beam width bounds the memory required to perform the search. Since a goal state could potentially be pruned, beam search sacrifices completeness (the guarantee that an algorithm will terminate with a solution, if one exists). Beam search is not optimal (that is, there is no guarantee that it will find the best solution).

Uses

A beam search is most often used to maintain tractability in large systems with insufficient amount of memory to store the entire search tree. For example, it has been used in many machine translation systems. (The state of the art now primarily uses neural machine translation based methods.) To select the best translation, each part is processed, and many different ways of translating the words appear. The top best translations according to their sentence structures are kept, and the rest are discarded. The translator then evaluates the translations according to a given criterion, choosing the translation which best keeps the goals. The first use of a beam search was in the Harpy Speech Recognition System, CMU 1976.

Variants

Beam search has been made complete by combining it with depth-first search, resulting in *beam stack search* and *depth-first beam search*, and with limited discrepancy search, resulting in *beam search using limited discrepancy backtracking* (BULB). The resulting search algorithms are anytime algorithms that find good but likely sub-optimal solutions quickly, like beam search, then backtrack and continue to find improved solutions until convergence to an optimal solution.

INFORMED SEARCH ALGORITHMS

So far we have talked about the uninformed search algorithms which looked through search space for all possible solutions of the problem without having any additional knowledge about search space. But informed search algorithm contains an array of knowledge such as how far we are from the goal, path cost, how to reach to goal node, etc. This knowledge help agents to explore less to the search space and find more efficiently the goal node.

The informed search algorithm is more useful for large search space. Informed search algorithm uses the idea of heuristic, so it is also called Heuristic search.

Heuristics function: Heuristic is a function which is used in Informed Search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by h(n), and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive.

Admissibility of the heuristic function is given as:

1. h(n) <= h*(n)

Here h(n) is heuristic cost, and h*(n) is the estimated cost. Hence heuristic cost should be less than or equal to the estimated cost.

Pure Heuristic Search:

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues unit a goal state is found.

In the informed search we will discuss two main algorithms which are given below:

- Best First Search Algorithm(Greedy search)
- A* Search Algorithm

Best-first Search Algorithm (Greedy Search)

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node

which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

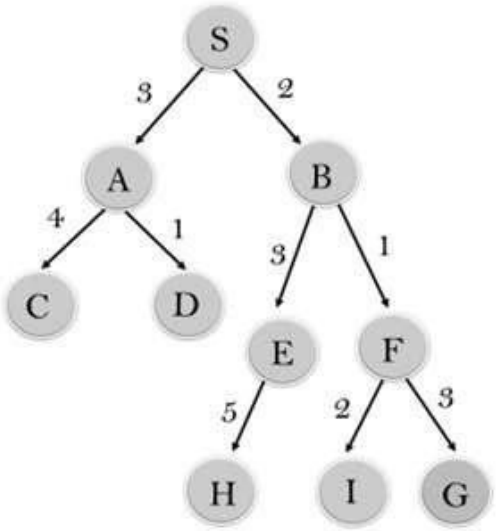
1. f(n)= g(n).

Were, h(n)= estimated cost from node n to the goal.

The greedy best first algorithm is implemented by the priority queue.

Best first search algorithm:

- Step 1: Place the starting node into the OPEN list.
- Step 2: If the OPEN list is empty, Stop and return failure.
- Step 3: Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
- Step 4: Expand the node n, and generate the successors of node n.
- Step 5: Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- Step 6: For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- Step 7: Return to Step 2.



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

Advantages:

- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

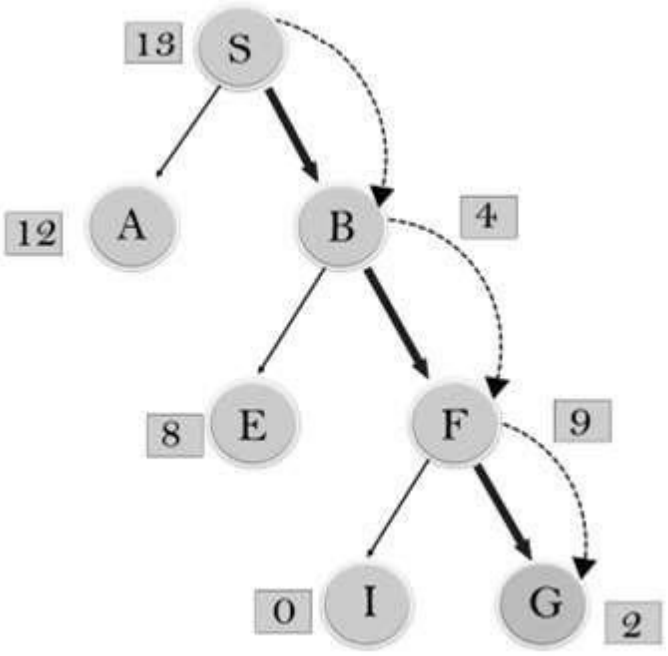
- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

Example:

Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.

In this search example, we are using two lists which are OPEN and CLOSED Lists. Following are the iteration for traversing the above example.

Expand the nodes of S and put in the CLOSED list



Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]

Iteration 2: Open [E, F, A], Closed [S, B]

: Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F]

: Open [I, E, A], Closed [S, B, F, G]

Hence the final solution path will be: S——> B——>F——> G

Time Complexity: The worst case time complexity of Greedy best first search is $O(b^m)$.

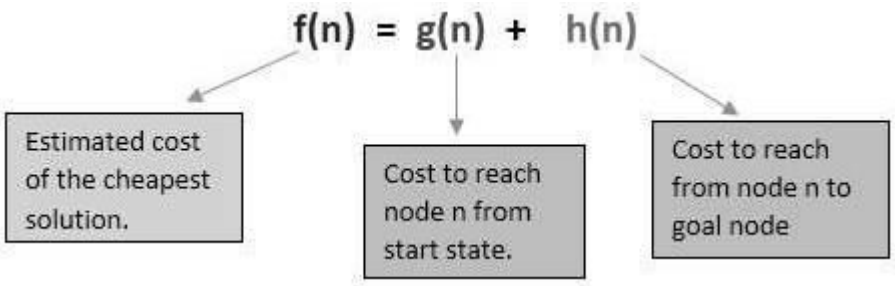
Space Complexity: The worst case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.

Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

Optimal: Greedy best first search algorithm is not optimal.

A Search Algorithm:*

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n). In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a fitness number.



At each point in the search space, only those node is expanded which have the lowest value of f(n), and the algorithm terminates when the goal node is found.

Algorithm of A* search:

The A* search algorithm is a popular pathfinding and graph traversal method used in various applications like game development and robotics. It combines the strengths of Dijkstra's algorithm and the Greedy Best-First-Search by considering both the cost to reach the node (g) and the estimated cost to reach the goal from that node (h). The algorithm uses a priority queue to explore nodes with the lowest f(x) = g(x) + h(x) value first. A* ensures that the shortest path is found efficiently by balancing exploration of new paths and exploitation of known paths, making it both complete and optimal.

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to Step 2.

Advantages:

- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

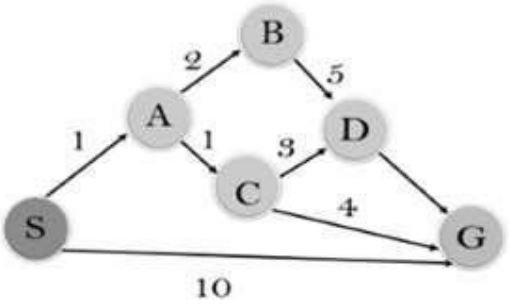
Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example: In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table

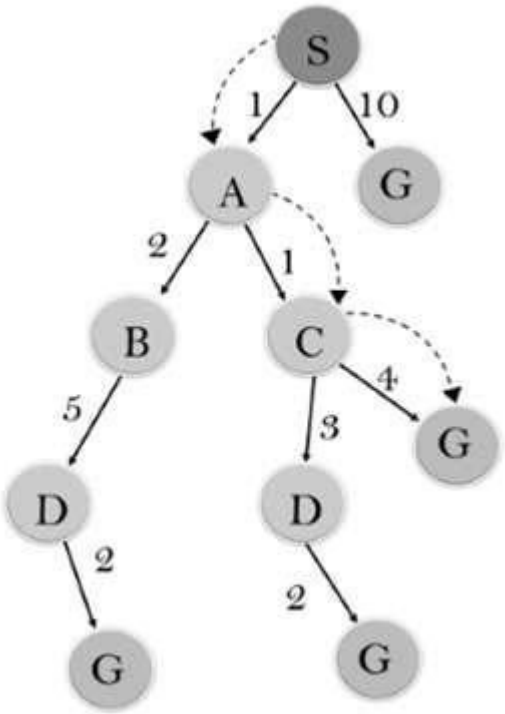
so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.

Here we will use OPEN and CLOSED list.



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution:



Initialization: {(S, 5)}

Iteration1: {(S—> A, 4), (S—>G, 10)}

Iteration2: {(S—> A—>C, 4), (S—> A—>B, 7), (S—>G, 10)}

Iteration3: {(S—> A—>C—>G, 6), (S—> A—>C—>D, 11), (S—> A—>B, 7), (S—>G, 10)}

Iteration 4 will give the final result, as S—>A—>C—>G it provides the optimal path with cost 6.

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n) <="" li="">

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- Admissible: the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- Consistency: Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is O(b^d)

HILL CLIMBING ALGORITHM

Hill Climbing is a heuristic search used for mathematical optimisation problems in the field of Artificial Intelligence.

So, given a large set of inputs and a good heuristic function, the algorithm tries to find the best possible solution to the problem in the most reasonable time period. This solution may not be the absolute best(global optimal maximum) but it is sufficiently good considering the time allotted.

The definition above implies that hill-climbing solves the problems where we need to maximise or minimise a given real function by selecting values from the given inputs.

A great example of this is the *Travelling Salesman Problem* where we need to minimise the distance travelled by the salesman.

Features of Hill Climbing

It carries out a Heuristic search.

A heuristic function is one that ranks all the potential alternatives in a search algorithm based on the information available. It helps the algorithm to select the best route to its solution. This basically means that this search *algorithm* may not find the optimal solution to the problem but it will give the best possible solution in a reasonable amount of time.

It is a variant of the generate-and-test algorithm.

The algorithm is as follows :

Step1: Generate possible solutions.

Step2: Evaluate to see if this is the expected solution.

Step3: If the solution has been found quit else go back to step 1.

Hill climbing takes the feedback from the test procedure and the generator uses it in deciding the next move in the search space. Hence, we call it as a *variant of the generate-and-test algorithm*.

It uses the Greedy approach.

At any point in state space, the search moves in that direction only which optimises the cost of function with the hope of finding the most optimum solution at the end.

State Space diagram for Hill Climbing

The State-space diagram is a graphical representation of the *set of states(input)* our search algorithm can reach vs the value of our *objective function(function we intend to maximise/minimise)*. Here;

1. The X-axis denotes the state space ie states or configuration our algorithm may reach.

2. The Y-axis denotes the values of objective function corresponding to a particular state.

The best solution will be that state space where objective function has maximum value or global maxima.

Following are the different regions in the State Space Diagram;

- Local maxima: It is a state which is better than its neighbouring state however there exists a state which is better than it (global maximum). This state is better because here the value of the objective function is higher than its neighbours.
- Global maxima: It is the best possible state in the state space diagram. This because at this state, objective function has the highest value.
- Plateau/flat local maxima: It is a flat region of state space where neighbouring states have the same value.
- Ridge: It is a region which is higher than its neighbour's but itself has a slope. It is a special kind of local maximum.
- Current state: The region of state space diagram where we are currently present during the search.

Working of Hill Climbing Algorithm

Hill Climbing is the simplest implementation of a *Genetic Algorithm*. Instead of focusing on the ease of implementation, it completely rids itself of concepts like population and crossover. It has faster iterations compared to more traditional genetic algorithms, but in return, it is less thorough than the traditional ones.

Types of Hill Climbing

Simple Hill Climbing

Simple hill climbing is the simplest way to implement a hill-climbing algorithm. It only evaluates the neighbour node state at a time and selects the first one which optimizes current cost and set it as a current state. It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state. This algorithm has the following features:

- Less time consuming
- Less optimal solution
- The solution is not guaranteed

Algorithm for Simple Hill Climbing

- Step 1: Evaluate the initial state, if it is goal state then return success and Stop.
- Step 2: Loop Until a solution is found or there is no new operator left to apply.
- Step 3: Select and apply an operator to the current state.
- Step 4: Check new state:
  i. If it is goal state, then return success and quit.
  ii. else if it is better than the current state then assign new state as a current state.
  iii. else if not better than the current state, then return to step 2.
- Step 5: Exit.

Steepest-Ascent hill climbing

The steepest-Ascent algorithm is a variation of the simple hill-climbing algorithm. This algorithm examines all the neighbouring nodes of the current state and selects one neighbour node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbours.

Algorithm for Steepest-Ascent hill climbing

- Step 1: Evaluate the initial state, if it is goal state then return success and stop, else make the current state as your initial state.
- Step 2: Loop until a solution is found or the current state does not change.
  i. Let S be a state such that any successor of the current state will be better than it.
  ii. For each operator that applies to the current state;
    — Apply the new operator and generate a new state.
    — Evaluate the new state.
    — If it is goal state, then return it and quit, else compare it to the S.
    — If it is better than S, then set new state as S.
    — If the S is better than the current state, then set the current state to S.
- Step 5: Exit.

Stochastic hill climbing

Stochastic hill climbing does not examine for all its neighbours before moving. Rather, this search algorithm selects one neighbour node at random and evaluate it as a current state or examine another state.

Problems in different regions in Hill climbing

Hill climbing cannot reach the best possible state if it enters any of the following regions :

Local maximum: At a local maximum all neighbouring states have values which are worse than the current state. Since hill-climbing uses a greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.

To overcome the local maximum problem: Utilise the *backtracking technique*. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.

Plateau: On the plateau, all neighbours have the same value. Hence, it is not possible to select the best direction.

To overcome plateaus: Make a big jump. Randomly select a state far away from the current state. Chances are that we will land at a non-plateau region

Ridge: Any point on a ridge can look like a peak because the movement in all possible directions is downward. Hence, the algorithm stops when it reaches such a state.

To overcome Ridge: You could use two or more rules before testing. It implies moving in several directions at once.

Simulated Annealing

A hill-climbing algorithm which never makes a move towards a lower value guaranteed to be incomplete because it can get stuck on a local maximum. And if algorithm applies a random walk, by moving a successor, then it may complete but not efficient.

Mechanically, the term annealing is a process of hardening a metal or glass to a high temperature then cooling gradually, so this allows the metal to reach a low-energy crystalline state. The same process is used in simulated annealing in which the algorithm picks a random move, instead of picking the best move. If the random move improves the state, then it

follows the same path. Otherwise, the algorithm follows the path which has a probability of less than 1 or it moves downhill and chooses another path.

INFORMED SEARCH

Informed Search refers to search algorithms which help in navigating large databases with certain available information about the end goal in search and most widely used in large databases where uninformed search algorithms can't accurately curate precise results.

For example, when searching on google maps you give the search algorithm information like a place you plan to visit from your current location for it to accurately navigate the distance, the time taken and real-time traffic updates on that particular route. This is all driven by complex Informed search algorithms powering google maps search functionality.

Types of Informed Search Algorithms

Before getting started with different types of Informed search Algorithms, it's important to understand some basic concepts like Search Space which refers to space or the database in which the search is to be performed, the Initial State or Start State from where the search begins and the goal state which is the result of the search like our destination in the earlier example of google maps and goal test to check whether the current state is the destination or goal state. Path cost is a numerical term assigned to measure the numeric cost of the path taken to achieve the goal. Heuristic function, which is a function used to measure how close our current state is to the goal state and uses heuristic properties to find out the best possible path with respect to path cost to achieve the goal state.

In informed search algorithms as discussed, we have information on the goal state which narrows down our results precisely. There may be many possible ways to get to the goal state, but we need to get the best possible outcome or path for our search; this is where informed search shines.

Pure Heuristic search

A pure heuristic search algorithm is a simple search performed on the basis of heuristic value denoted y h(n) to a node.

In a heuristic search, there are two lost created, open for new but unexpanded nodes and closed for expanded nodes, where for every iteration

the node with smallest heuristic value is expanded, and all its 'child' nodes are expanded and added to close it. A Heuristic function is then applied to that closed list, and the node with the shortest path is saved, and rest are dispersed.

Best First or 'Greedy' Search

In order to understand the Greedy search, we need to briefly understand the concept of DFS and FS. They're both vertex-based techniques to find the shortest path. While DFS uses a stack data structure, BFS uses a queue data structure to find out the shortest route.

Greedy search at its core uses the best path from the current state using a combination of both DFS and FS techniques to find the shortest path. The node that is closest to the goal is expanded in that current state and the closest cost to that point.

It's called the greedy search because it may not find the best solution at a given point of time, but it surely does give you an optimal one within a reasonable amount of time. It can be best explained by the example of the Travelling Salesman problem.

A salesman is given a list of places to visit in a city and has to find his optimum route to travel in order to be productive to reduce his travelling time as much as possible. Here he'll be given a choice of choosing between 2 or more places from his current position to go to, and he'll choose the one with least distance thereby reducing his travel time albeit not being his best destination but that is a roader optimal solution.

A Tree Search*

Simply put, A* search combines elements and strengths from a greedy search and a Uniform cost search. The heuristic of A*search is the summation of Heuristic cost in greedy search and that in uniform cost search denoted here,

F(x)= h(x)+g(x)

Where,

- h(x): Forward cost referring cost of a node from the current state to the goal state.
- g(x): Backward cost which is the cost of a node from the root state or initial state.

Here, as represented, the idea is to select a node with the shortest f(x).

One of the biggest advantages of A* search algorithm is that it's a complete search algorithm as when compared to a simple heuristic approach which only gives the shortest paths, it also takes into account the optimality of the operation overall and therefore is the most widely used search algorithm and can solve complex functions with complex search space. However, as discussed, it keeps into its memory all of the expansion and generation of nodes, is resource constraint and can't be used for very large scale operations.

*A*Graph Search*

In Tree search, the branches or nodes are expanded again to newer iterations and thus wasting time in the process whereas in graph search same nodes which are expanded before aren't expanded. Here the heuristic is represented by consistency, where the graph search is optimal when the forward cost as represented in A*tree search is equal to or less than backward cost between the two nodes.

One of the advantages of graph search over A* search is that it doesn't store all the nodes and therefore isn't resource-constrained and can be used for very large-scale operations.

INTRODUCTION TO HILL CLIMBING IN ARTIFICIAL INTELLIGENCE

Hill Climbing is a form of heuristic search algorithm which is used in solving optimization related problems in Artificial Intelligence domain. The algorithm starts with a non-optimal state and iteratively improves its state until some predefined condition is met. The condition to be met is based on the heuristic function. The aim of the algorithm is to reach an optimal state which is better than its current state. The starting point which is the non-optimal state is referred to as the base of the hill and it tries to constantly iterate (climb) untill it reaches the peak value, that is why it is called Hill Climbing Algorithm.

Hill Climbing Algorithm is a memory-efficient way of solving large computational problems. It takes into account the current state and immediate neighbouring state. The Hill Climbing Problem is particularly useful when we want to maximize or minimize any particular function based on the input which it is taking. The most commonly used Hill Climbing Algorithm is "Travelling Salesman" Problem" where we have to minimize the distance travelled by the salesman. Hill Climbing Algorithm

may not find the global optimal (best possible) solution but it is good for finding local minima/maxima efficiently.

Features of Hill Climbing in Artificial Intelligence

Following are few of the key features of Hill Climbing Algorithm

- Greedy Approach: The algorithm moves in the direction of optimizing the cost i.e. finding Local Maxima/Minima
- No Backtracking: It cannot remember the previous state of the system so backtracking to the previous state is not possible
- Feedback Mechanism: The feedback from the previous computation helps in deciding the next course of action i.e. whether to move up or down the slope

State Space Diagram – Hill Climbing in Artificial Intelligence

- Local Maxima/Minima: Local Minima is a state which is better than its neighbouring state, however, it is not the best possible state as there exists a state where objective function value is higher
- Global Maxima/Minima: It is the best possible state in the state diagram. Here the value of the objective function is highest
- Current State: Current State is the state where the agent is present currently
- Flat Local Maximum: This region is depicted by a straight line where all neighbouring states have the same value so every node is local maximum over the region

Problems in Hill Climbing Algorithm

Local Maximum

The algorithm terminates when the current node is local maximum as it is better than its neighbours. However, there exists a global maximum where objective function value is higher

Solution: Back Propagation can mitigate the problem of Local maximum as it starts exploring alternate paths when it encounters Local Maximum

Ridge

Ridge occurs when there are multiple peaks and all have the same value or in other words, there are multiple local maxima which are same as global maxima

Plateau

Plateau is the region where all the neighbouring nodes have the same value of objective function so the algorithm finds it hard to select an appropriate direction.

Types of Hill Climbing Algorithm in Artificial Intelligence

Simple Hill Climbing

It is the simplest form of the Hill Climbing Algorithm. It only takes into account the neighboring node for its operation. If the neighboring node is better than the current node then it sets the neighbor node as the current node. The algorithm checks only one neighbor at a time. Following are a few of the key feature of the Simple Hill Climbing Algorithm

Algorithm

1. Examine the current state, Return success if it is a goal state
2. Continue the Loop until a new solution is found or no operators are left to apply
3. Apply the operator to the node in the current state
4. Check for the new state
   - If Current State = Goal State, Return success and exit
   - Else if New state is better than current state then Goto New state
   - Else return to step 2
5. Exit

Steepest-Ascent Hill Climbing

Steepest-Ascent hill climbing is an advanced form of simple Hill Climbing Algorithm. It runs through all the nearest neighbor nodes and selects the node which is nearest to the goal state. The algorithm requires more computation power than Simple Hill Climbing Algorithm as it searches through multiple neighbors at once.

Algorithm

1. Examine the current state, Return success if it is a goal state
2. Continue the Loop until a new solution is found or no operators are left to apply

Let 'Temp' be a state such that any successor of the current state will have a higher value for the objective function. For all operators that can be applied to the current state

- Apply the operator to create a new state
- Examine new state
- If Current State = Goal State, Return success and exit
- Else if New state is better than Temp then set this state as Temp
- If Temp is better than Current State set Current state to Target

Stochastic Hill Climbing

Stochastic Hill Climbing doesn't look at all its neighboring nodes to check if it is better than the current node instead, it randomly selects one neighboring node, and based on the pre-defined criteria it decides whether to go to the neighboring node or select an alternate node.

Advantage of Hill Climbing Algorithm in Artificial Intelligence

Advantage of Hill Climbing Algorithm in Artificial Intelligence is given below:

- Hill Climbing is very useful in routing-related problems like Travelling Salesmen Problem, Job Scheduling, Chip Designing, and Portfolio Management
- It is good in solving the optimization problem while using only limited computation power
- It is more efficient than other search algorithms

Hill Climbing Algorithm is a very widely used algorithm for Optimization related problems as it gives decent solutions to computationally challenging problems. It has certain drawbacks associated with it like its Local Minima, Ridge, and Plateau problem which can be solved by using some advanced algorithm.

Understanding Hill Climbing Algorithm in Artificial Intelligence

A hill-climbing algorithm is an Artificial Intelligence (AI) algorithm that increases in value continuously until it achieves a peak solution. This algorithm is used to optimize mathematical problems and in other real-life applications like marketing and job scheduling.

Introduction to hill climbing algorithm

A hill-climbing algorithm is a local search algorithm that moves continuously upward (increasing) until the best solution is attained. This algorithm comes to an end when the peak is reached.

This algorithm has a node that comprises two parts: state and value. It begins with a non-optimal state (the hill's base) and upgrades this state until a certain precondition is met. The heuristic function is used as the basis for this precondition. The process of continuous improvement of the current state of iteration can be termed as climbing. This explains why the algorithm is termed as a *hill-climbing algorithm*.

A hill-climbing algorithm's objective is to attain an optimal state that is an upgrade of the existing state. When the current state is improved, the algorithm will perform further incremental changes to the improved state. This process will continue until a peak solution is achieved. The peak state cannot undergo further improvements.

Features of a hill climbing algorithm

A hill-climbing algorithm has four main features:

1. It employs a greedy approach: This means that it moves in a direction in which the cost function is optimized. The greedy approach enables the algorithm to establish local maxima or minima.
2. No Backtracking: A hill-climbing algorithm only works on the current state and succeeding states (future). It does not look at the previous states.
3. Feedback mechanism: The algorithm has a feedback mechanism that helps it decide on the direction of movement (whether up or down the hill). The feedback mechanism is enhanced through the generate-and-test technique.
4. Incremental change: The algorithm improves the current solution by incremental changes.

State-space diagram analysis

A state-space diagram provides a graphical representation of states and the optimization function. If the objective function is the y-axis, we aim to establish the local maximum and global maximum.

If the cost function represents this axis, we aim to establish the local minimum and global minimum. More information about local minimum, local maximum, global minimum, and global maximum can be found here.

The following diagram shows a simple state-space diagram. The objective function has been shown on the y-axis, while the state-space represents the x-axis.

A state-space diagram consists of various regions that can be explained as follows;

- Local maximum: A local maximum is a solution that surpasses other neighboring solutions or states but is not the best possible solution.
- Global maximum: This is the best possible solution achieved by the algorithm.
- Current state: This is the existing or present state.
- Flat local maximum: This is a flat region where the neighboring solutions attain the same value.
- Shoulder: This is a plateau whose edge is stretching upwards.

Problems with hill climbing

There are three regions in which a hill-climbing algorithm cannot attain a global maximum or the optimal solution: local maximum, ridge, and plateau.

Local maximum

At this point, the neighboring states have lower values than the current state. The greedy approach feature will not move the algorithm to a worse off state. This will lead to the hill-climbing process's termination, even though this is not the best possible solution.

This problem can be solved using momentum. This technique adds a certain proportion (m) of the initial weight to the current one. m is a value between 0 and 1. Momentum enables the hill-climbing algorithm to take huge steps that will make it move past the local maximum.

Plateau

In this region, the values attained by the neighboring states are the same. This makes it difficult for the algorithm to choose the best direction.

This challenge can be overcome by taking a huge jump that will lead you to a non-plateau space.

Ridge

The hill-climbing algorithm may terminate itself when it reaches a ridge. This is because the peak of the ridge is followed by downward movement rather than upward movement.

This impediment can be solved by going in different directions at once.

Types of hill climbing algorithms

The following are the types of a hill-climbing algorithm:

Simple hill climbing

This is a simple form of hill climbing that evaluates the neighboring solutions. If the next neighbor state has a higher value than the current state, the algorithm will move. The neighboring state will then be set as the current one.

This algorithm consumes less time and requires little computational power. However, the solutions produced by the algorithm are sub-optimal. In some cases, an optimal solution may not be guaranteed.

Steepest – Ascent hill climbing

This algorithm is more advanced than the simple hill-climbing algorithm. It chooses the next node by assessing the neighboring nodes. The algorithm moves to the node that is closest to the optimal or goal state.

Algorithm

- Conduct an assessment of the current state. Stop the process and indicate success if it is a goal state.
- Perform looping on the current state if the assessment in step 1 did not establish a goal state.
- Continue looping to attain a new solution.
- Establish or set a state (X) such that current state successors have higher values than it.
- Run the new operator and produce a new solution.
- Assess this solution to establish whether it is a goal state. If this is the case, exit the program. Otherwise, compare it with the state (X).
- If the new state has a higher value than the state (X), set it as X. The current state should be set to Target if the state (X) has a higher value than the current state.

Stochastic hill climbing

In this algorithm, the neighboring nodes are selected randomly. The selected node is assessed to establish the level of improvement. The algorithm will move to this neighboring node if it has a higher value than the current state.

Applications of hill climbing algorithm

The hill-climbing algorithm can be applied in the following areas:

Marketing

A hill-climbing algorithm can help a marketing manager to develop the best marketing plans. This algorithm is widely used in solving Traveling-Salesman problems. It can help by optimizing the distance covered and improving the travel time of sales team members. The algorithm helps establish the local minima efficiently.

Robotics

Hill climbing is useful in the effective operation of robotics. It enhances the coordination of different systems and components in robots.

Job Scheduling

The hill climbing algorithm has also been applied in job scheduling. This is a process in which system resources are allocated to different tasks within a computer system. Job scheduling is achieved through the migration of jobs from one node to a neighboring node. A hill-climbing technique helps establish the right migration route.

8

System Virtual Machines in Artificial Intelligence

VIRTUAL MACHINE

In computing, a virtual machine (VM) is the virtualization/emulation of a computer system. Virtual machines are based on computer architectures and provide functionality of a physical computer. Their implementations may involve specialized hardware, software, or a combination.

Virtual machines differ and are organized by their function, shown here:

- System virtual machines (also termed full virtualization VMs) provide a substitute for a real machine. They provide functionality needed to execute entire operating systems. A hypervisor uses native execution to share and manage hardware, allowing for multiple environments which are isolated from one another, yet exist on the same physical machine. Modern hypervisors use hardware-assisted virtualization, virtualization-specific hardware, primarily from the host CPUs.
- Process virtual machines are designed to execute computer programs in a platform-independent environment.

Some virtual machine emulators, such as QEMU and video game console emulators, are designed to also emulate (or "virtually imitate") different system architectures thus allowing execution of software applications and operating systems written for another CPU or architecture. Operating-system-level virtualization allows the resources of a computer to be partitioned via the kernel. The terms are not universally interchangeable.

System virtual machines

A "virtual machine" was originally defined by Popek and Goldberg as "an efficient, isolated duplicate of a real computer machine." Current use includes virtual machines that have no direct correspondence to any real hardware. The physical, "real-world" hardware running the VM is generally referred to as the 'host', and the virtual machine emulated on that machine is generally referred to as the 'guest'. A host can emulate several guests, each of which can emulate different operating systems and hardware platforms.

The desire to run multiple operating systems was the initial motive for virtual machines, so as to allow time-sharing among several single-tasking operating systems. In some respects, a system virtual machine can be considered a generalization of the concept of virtual memory that historically preceded it. IBM's CP/CMS, the first systems to allow full virtualization, implemented time sharing by providing each user with a single-user operating system, the Conversational Monitor System (CMS). Unlike virtual memory, a system virtual machine entitled the user to write privileged instructions in their code. This approach had certain advantages, such as adding input/output devices not allowed by the standard system.

As technology evolves virtual memory for purposes of virtualization, new systems of memory overcommitment may be applied to manage memory sharing among multiple virtual machines on one computer operating system. It may be possible to share *memory pages* that have identical contents among multiple virtual machines that run on the same physical machine, what may result in mapping them to the same physical page by a technique termed kernel same-page merging (KSM). This is especially useful for read-only pages, such as those holding code segments, which is the case for multiple virtual machines running the same or similar software, software libraries, web servers, middleware components, etc. The guest operating systems do not need to be compliant with the host hardware, thus making it possible to run different operating systems on the same computer (e.g., Windows, Linux, or prior versions of an operating system) to support future software.

The use of virtual machines to support separate guest operating systems is popular in regard to embedded systems. A typical use would be to run a real-time operating system simultaneously with a preferred complex operating system, such as Linux or Windows. Another use would be for novel and unproven software still in the developmental stage, so it runs

inside a sandbox. Virtual machines have other advantages for operating system development and may include improved debugging access and faster reboots.

Multiple VMs running their own guest operating system are frequently engaged for server consolidation.

Process virtual machines

"Application virtual machine" redirects here. Not to be confused with application virtualization.

A process VM, sometimes called an *application virtual machine*, or *Managed Runtime Environment* (MRE), runs as a normal application inside a host OS and supports a single process. It is created when that process is started and destroyed when it exits. Its purpose is to provide a platform-independent programming environment that abstracts away details of the underlying hardware or operating system and allows a program to execute in the same way on any platform.

A process VM provides a high-level abstraction – that of a high-level programming language (compared to the low-level ISA abstraction of the system VM). Process VMs are implemented using an interpreter; performance comparable to compiled programming languages can be achieved by the use of just-in-time compilation.

This type of VM has become popular with the Java programming language, which is implemented using the Java virtual machine. Other examples include the Parrot virtual machine and the .NET Framework, which runs on a VM called the Common Language Runtime. All of them can serve as an abstraction layer for any computer language.

A special case of process VMs are systems that abstract over the communication mechanisms of a (potentially heterogeneous) computer cluster. Such a VM does not consist of a single process, but one process per physical machine in the cluster. They are designed to ease the task of programming concurrent applications by letting the programmer focus on algorithms rather than the communication mechanisms provided by the interconnect and the OS. They do not hide the fact that communication takes place, and as such do not attempt to present the cluster as a single machine.

Unlike other process VMs, these systems do not provide a specific programming language, but are embedded in an existing language; typically such a system provides bindings for several languages (e.g., C and Fortran).

Examples are Parallel Virtual Machine (PVM) and Message Passing Interface (MPI). They are not strictly virtual machines because the applications running on top still have access to all OS services and are therefore not confined to the system model.

History

Both system virtual machines and process virtual machines date to the 1960s and continue to be areas of active development.

System virtual machines grew out of time-sharing, as notably implemented in the Compatible Time-Sharing System (CTSS). Time-sharing allowed multiple users to use a computer concurrently: each program appeared to have full access to the machine, but only one program was executed at the time, with the system switching between programs in time slices, saving and restoring state each time. This evolved into virtual machines, notably via IBM's research systems: the M44/44X, which used partial virtualization, and the CP-40 and SIMMON, which used full virtualization, and were early examples of hypervisors. The first widely available virtual machine architecture was the CP-67/CMS. An important distinction was between using multiple virtual machines on one host system for time-sharing, as in M44/44X and CP-40, and using one virtual machine on a host system for prototyping, as in SIMMON. Emulators, with hardware emulation of earlier systems for compatibility, date back to the IBM System/360 in 1963, while the software emulation (then-called "simulation") predates it.

Process virtual machines arose originally as abstract platforms for an intermediate language used as the intermediate representation of a program by a compiler; early examples date to around 1966. An early 1966 example was the O-code machine, a virtual machine that executes O-code (object code) emitted by the front end of the BCPL compiler. This abstraction allowed the compiler to be easily ported to a new architecture by implementing a new back end that took the existing O-code and compiled it to machine code for the underlying physical machine. The Euler language used a similar design, with the intermediate language named *P* (portable).

This was popularized around 1970 by Pascal, notably in the Pascal-P system (1973) and Pascal-S compiler (1975), in which it was termed p-code and the resulting machine as a p-code machine. This has been influential, and virtual machines in this sense have been often generally called p-code machines. In addition to being an intermediate language, Pascal p-code was also executed directly by an interpreter implementing

the virtual machine, notably in UCSD Pascal (1978); this influenced later interpreters, notably the Java virtual machine (JVM). Another early example was SNOBOL4 (1967), which was written in the SNOBOL Implementation Language (SIL), an assembly language for a virtual machine, which was then targeted to physical machines by transpiling to their native assembler via a macro assembler. Macros have since fallen out of favor, however, so this approach has been less influential. Process virtual machines were a popular approach to implementing early microcomputer software, including Tiny BASIC and adventure games, from one-off implementations such as Pyramid 2000 to a general-purpose engine like Infocom's z-machine, which Graham Nelson argues is "possibly the most portable virtual machine ever created".

Significant advances occurred in the implementation of Smalltalk-80, particularly the Deutsch/Schiffmann implementation which pushed just-in-time (JIT) compilation forward as an implementation approach that uses process virtual machine.

Later notable Smalltalk VMs were VisualWorks, the Squeak Virtual Machine, and Strongtalk. A related language that produced a lot of virtual machine innovation was the Self programming language, which pioneered adaptive optimization and generational garbage collection. These techniques proved commercially successful in 1999 in the HotSpot Java virtual machine. Other innovations include having a register-based virtual machine, to better match the underlying hardware, rather than a stack-based virtual machine, which is a closer match for the programming language; in 1995, this was pioneered by the Dis virtual machine for the Limbo language. OpenJ9 is an alternative for HotSpot JVM in OpenJDK and is an open source eclipse project claiming better startup and less resource consumption compared to HotSpot.

Full virtualization

In full virtualization, the virtual machine simulates enough hardware to allow an unmodified "guest" OS (one designed for the same instruction set) to be run in isolation. This approach was pioneered in 1966 with the IBM CP-40 and CP-67, predecessors of the VM family.

Examples outside the mainframe field include Parallels Workstation, Parallels Desktop for Mac, VirtualBox, Virtual Iron, Oracle VM, Virtual PC, Virtual Server, Hyper-V, VMware Workstation, VMware Server (discontinued, formerly called GSX Server), VMware ESXi, QEMU, Adeos, Mac-on-Linux, Win4BSD, Win4Lin Pro, and Egenera vBlade technology.

Operating-system-level virtualization

In operating-system-level virtualization, a physical server is virtualized at the operating system level, enabling multiple isolated and secure virtualized servers to run on a single physical server. The "guest" operating system environments share the same running instance of the operating system as the host system. Thus, the same operating system kernel is also used to implement the "guest" environments, and applications running in a given "guest" environment view it as a stand-alone system. The pioneer implementation was FreeBSD jails; other examples include Docker, Solaris Containers, OpenVZ, Linux-VServer, LXC, AIX Workload Partitions, Parallels Virtuozzo Containers, and iCore Virtual Accounts.

MACHINE LEARNING

Machine learning, a branch of artificial intelligence, is a scientific discipline concerned with the design and development of algorithms that allow computers to evolve behaviours based on empirical data, such as from sensor data or databases.

A learner can take advantage of examples (data) to capture characteristics of interest of their unknown underlying probability distribution. Data can be seen as examples that illustrate relations between observed variables.

A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data; the difficulty lies in the fact that the set of all possible behaviours given all possible inputs is too large to be covered by the set of observed examples (training data).

Hence the learner must generalize from the given examples, so as to be able to produce a useful output in new cases. Machine learning, like all subjects in artificial intelligence, requires cross-disciplinary proficiency in several areas, such as probability theory, statistics, pattern recognition, cognitive science, data mining, adaptive control, computational neuroscience and theoretical computer science.

Definition

A computer programme is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Generalization

The core objective of a learner is to generalize from its experience. The training examples from its experience come from some generally unknown probability distribution and the learner has to extract from them something more general, something about that distribution, that allows it to produce useful answers in new cases.

Human Interaction

Some machine learning systems attempt to eliminate the need for human intuition in data analysis, while others adopt a collaborative approach between human and machine. Human intuition cannot, however, be entirely eliminated, since the system's designer must specify how the data is to be represented and what mechanisms will be used to search for a characterization of the data.

Algorithm Types

Machine learning algorithms are organized into a taxonomy, based on the desired outcome of the algorithm.

- Supervised learning generates a function that maps inputs to desired outputs (also called labels, because they are often provided by human experts labeling the training examples). For example, in a classification problem, the learner approximates a function mapping a vector into classes by looking at input-output examples of the function.
- Unsupervised learning models a set of inputs, like clustering.
- Semi-supervised learning combines both labeled and unlabeled examples to generate an appropriate function or classifier.
- Reinforcement learning learns how to act given an observation of the world. Every action has some impact in the environment, and the environment provides feedback in the form of rewards that guides the learning algorithm.
- Transduction tries to predict new outputs based on training inputs, training outputs, and test inputs.
- Learning to learn learns its own inductive bias based on previous experience.

Theory

The computational analysis of machine learning algorithms and their

performance is a branch of theoretical computer science known as computational learning theory. Because training sets are finite and the future is uncertain, learning theory usually does not yield absolute guarantees of the performance of algorithms. Instead, probabilistic bounds on the performance are quite common.

In addition to performance bounds, computational learning theorists study the time complexity and feasibility of learning. In computational learning theory, a computation is considered feasible if it can be done in polynomial time. There are two kinds of time complexity results. Positive results show that a certain class of functions can be learned in polynomial time. Negative results show that certain classes cannot be learned in polynomial time. There are many similarities between machine learning theory and statistics, although they use different terms.

Approaches

Decision Tree Learning

Decision tree learning uses a decision tree as a predictive model which maps observations about an item to conclusions about the item's target value.

Association Rule Learning

Association rule learning is a method for discovering interesting relations between variables in large databases.

Artificial Neural Networks

An artificial neural network (ANN) learning algorithm, usually called "neural network" (NN), is a learning algorithm that is inspired by the structure and/or functional aspects of biological neural networks. Computations are structured in terms of an interconnected group of artificial neurons, processing information using a connectionist approach to computation. Modern neural networks are non-linear statistical data modeling tools. They are usually used to model complex relationships between inputs and outputs, to find patterns in data, or to capture the statistical structure in an unknown joint probability distribution between observed variables.

Genetic Programming

Genetic programming (GP) is an evolutionary algorithm-based methodology inspired by biological evolution to find computer programmes

that perform a user-defined task. It is a specialization of genetic algorithms (GA) where each individual is a computer programme. It is a machine learning technique used to optimize a population of computer programmes according to a fitness landscape determined by a program's ability to perform a given computational task.

Inductive Logic Programming

Inductive logic programming (ILP) is an approach to rule learning using logic programming as a uniform representation for examples, background knowledge, and hypotheses. Given an encoding of the known background knowledge and a set of examples represented as a logical database of facts, an ILP system will derive a hypothesized logic programme which entails all the positive and none of the negative examples.

Support Vector Machines

Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. Given a set of training examples, each marked as belonging to one of two categories, an SVM training algorithm builds a model that predicts whether a new example falls into one category or the other.

Clustering

Cluster analysis or clustering is the assignment of a set of observations into subsets (called *clusters*) so that observations in the same cluster are similar in some sense. Clustering is a method of unsupervised learning, and a common technique for statistical data analysis.

Bayesian Networks

A Bayesian network, belief network or directed acyclic graphical model is a probabilistic graphical model that represents a set of random variables and their conditional independencies via a directed acyclic graph (DAG). For example, a Bayesian network could represent the probabilistic relationships between diseases and symptoms. Given symptoms, the network can be used to compute the probabilities of the presence of various diseases. Efficient algorithms exist that perform inference and learning.

Reinforcement Learning

Reinforcement learning is concerned with how an *agent* ought to take

actions in an *environment* so as to maximize some notion of long-term *reward*. Reinforcement learning algorithms attempt to find a *policy* that maps *states* of the world to the actions the agent ought to take in those states. Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected.

Representation Learning

Several learning algorithms, mostly unsupervised learning algorithms, aim at discovering better representations of the inputs provided during training. Classical examples include principal components analysis and clustering. Representation learning algorithms often attempt to preserve the information in their input but transform it it in a way that makes it useful, often as a pre-processing step before performing classification or predictions, allowing to reconstruct the inputs coming from the unknown data generating distribution, while not being necessarily faithful for input configurations that are unplausible under that distribution. Manifold learning algorithms attempt to do so under the constraint that the learned representation is low-dimensional. Sparse coding algorithms attempt to do so under the constraint that the learned representation is sparse (has many zeros). Deep learning algorithms discover multiple levels of representation, or a hierarchy of features, with higher-level, more abstract features defined in terms of (or generating) lower-level features. It has been argued that an ideal representation is one that disentangles the underlying factors of variation that explain the observed data.

Applications

Applications for machine learning include machine perception, computer vision, natural language processing, syntactic pattern recognition, search engines, medical diagnosis, bioinformatics, brain-machine interfaces and cheminformatics, detecting credit card fraud, stock market analysis, classifying DNA sequences, speech and handwriting recognition, object recognition in computer vision, game playing, software engineering, adaptive websites, robot locomotion, and structural health monitoring. Machine learning techniques helped win a major software competition: in 2006, the online movie company Netflix held the first "Netflix Prize" competition to find a programme to better predict user preferences and beat its existing Netflix movie recommendation system by at least 10%. The AT&T Research Team BellKor won over several other teams with their

machine learning programme called Pragmatic Chaos. After winning several minor prizes, it won the 2009 grand prize competition for $1 million.

Software

RapidMiner, KNIME, Weka, ODM, Shogun toolbox, Orange and Apache Mahout are software suites containing a variety of machine learning algorithms.

MACHINE LEVEL OPERATIONS

Bits and Bytes

Down in the depths of your computer, below even the operating system are bits of memory. These days we are used to working at such a high level that it is easy to forget them. Bits (or binary digits) are the lowest level software objects in a computer: there is nothing more primitive. For precisely this reason, it is rare for high level languages to even acknowledge the existence of bits, let alone manipulate them. Manipulating bit patterns is usually the preserve of assembly language programmers. C, however, is quite different from most other high level languages in that it allows a programmer full access to bits and even provides high level operators for manipulating them.

Bit Patterns

- All computer data, of any type, are bit patterns. The only difference between a string and a floating point variable is the way in which we choose to interpret the patterns of bits in a computer's memory. For the most part, it is quite unnecessary to think of computer data as bit patterns; systems programmers, on the other hand, frequently find that they need to handle bits directly in order to make efficient use of memory when using flags. A flag is a message which is either one thing or the other: in system terms, the flag is said to be 'on' or 'off' or alternatively *set* or *cleared*. The usual place to find flags is in a status register of a CPU (central processor unit) or in a pseudo-register (this is a status register for an imaginary processor, which is held in memory). A status register is a group of bits (a byte perhaps) in which each bit signifies something special. In an ordinary byte of data, bits are grouped together and are interpreted to have a collective meaning; in a status register they are thought of as being

independent. Programmers are interested to know about the contents of bits in these registers, perhaps to find out what happened in a programme after some special operation is carried out.

Flags, Registers and Messages

A *register* is a place inside a computer processor chip, where data are worked upon in some way. A *status register* is a register which is used to return information to a programmer about the operations which took place in other registers.

Status registers contain flags which give yes or no answers to questions concerning the other registers. In advanced programming, there may be call for "pseudo registers" in addition to "real" ones. A pseudo register is merely a register which is created by the programmer in computer memory (it does not exist inside a processor). Messages are just like pseudo status registers: they are collections of flags which signal special information between different devices and/or different programs in a computer system.

Messages do not necessarily have fixed locations: they may be passed a parameters. Messages are a very compact way of passing information to low level functions in a programme. Flags, registers, pseudo-registers and messages are all treated as bit patterns.

A programme which makes use of them must therefore be able to assign these objects to C variables for use. A bit pattern would normally be declared as a character or some kind of integer type in C, perhaps with the aid of a typedef statement.

```
typedef char byte;
typedef int bitpattern;
bitpattern variable;
byte message;
```

The flags or bits in a register/message. have the values 1 or 0, depending upon whether they are on or off (set or cleared). A programme can test for this by using combinations of the operators which C provides.

Bit Operators and Assignments

C provides the following operators for handling bit patterns:

<< Bit shift left (a specified number or bit positions)

>> Bit shift right(a specified number of bit positions)

| Bitwise Inclusive OR

^ Bitwise Exclusive OR

& Bitwise AND

~ Bitwise one's complement

&= And assign (variable = variable & value)

|= Exclusive OR assign (variable = variable | value)

^= Inclusive OR assign (variable = variable ^ value)

>>= Shift right assign (variable = variable >> value)

<<= Shift left assign (variable = variable << value)

Bit Operators

Bitwise operations are not to be confused with logical operations (&&, ||.) A bit pattern is made up of 0s and 1s and bitwise operators operate individually upon each bit in the operand. Every 0 or 1 undergoes the operations individually. Bitwise operators (AND, OR) can be used in place of logical operators (&&,||), but they are less efficient, because logical operators are designed to reduce the number of comparisons made, in an expression, to the optimum: as soon as the truth or falsity of an expression is known, a logical comparison operator quits. A bitwise operator would continue operating to the last before the final result were known.

Below is a brief summary of the operations which are performed by the above operators on the bits of their operands.

Shift Operations

Imagine a bit pattern as being represented by the following group of boxes. Every box represents a bit; the numbers inside represent their values. The values written over the top are the common integer values which the whole group of bits would have, if they were interpreted collectively as an integer.

```
128    64    32   16  8     4     2  1
_____________________________________
|  0 |  0 |  0 |  0 |  0 |  0 |  0 |  1 |   = 1
_____________________________________
```

Shift operators move whole bit patterns left or right by shunting them between boxes.

The syntax of this operation is:

```
value << number of positions
value >> number of positions
```

So for example, using the boxed value (1) above:

```
1 << 1
```

would have the value 2, because the bit pattern would have been moved one place the the left:

```
128   64   32   16   8    4    2   1
_____________________________________
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | = 2
_____________________________________
```

Similarly:

```
1 << 4
```

has the value 16 because the original bit pattern is moved by four places:

```
128   64   32   16   8    4    2   1
_____________________________________
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | = 16
_____________________________________
```

And:

```
6 << 2 == 12
128   64   32   16   8    4    2   1
_____________________________________
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | = 6
_____________________________________
```

Shift left 2 places:

```
128   64   32   16   8    4    2   1
_____________________________________
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | = 12
_____________________________________
```

Notice that every shift left multiplies by 2 and that every shift right would divide by two, integerwise. If a bit reaches the edge of the group of boxes then it falls out and is lost forever.

So:

```
1 >> 1 == 0
2 >> 1 == 1
2 >> 2 == 0
n >> n == 0
```

A common use of shifting is to scan through the bits of a bitpattern one by one in a loop: this is done by using *masks*.

Truth Tables and Masking

The operations AND, OR (inclusive OR) and XOR/EOR (exclusive OR)

perform comparisons or "masking" operations between two bits. They are binary or dyadic operators. Another operation called COMPLEMENT is a unary operator. The operations performed by these bitwise operators are best summarized by*truth tables*.

Truth tables indicate what the results of all possible operations are between two single bits. The same operation is then carried out for all the bits in the variables which are operated upon.

Complement ~ The complement of a number is the *logical opposite* of the number. C provides a "one's complement" operator which simply changes all 1s into 0s and all 0s into 1s.

~1 has the value 0
(for each bit)

~0 has the value 1

As a truth table this would be summarized as follows:

```
~value   ==  result
   0          1
   1          0
```

AND &

This works between two values. e.g. (1 & 0)

```
value 1 &    value 2       ==     result
0              0                   0
0              1                   0
1              0                   0
1              1                   1
```

Both value 1 AND value 2 have to be 1 in order for the result or be 1.

OR |

This works between two values. e.g. (1 | 0)

```
value 1 |    value 2       ==     result
0            0                    0
0            1                    1
1            0                    1
1            1                    1
```

The result is 1 if one OR the other OR both of the values is 1.

XOR/EOR ^

Operates on two values. e.g. (1 ^ 0)

```
value 1 ^    value 2       ==     result
0              0                   0
0              1                   1
```

```
1               0               1
1               1               0
```

The result is 1 if one OR the other (but not both) of the values is 1.

Bit patterns and logic operators are often used to make *masks*. A mask is as a thing which fits over a bit pattern and modifies the result in order perhaps to single out particular bits, usually to cover up part of a bit pattern.

This is particularly pertinent for handling flags, where a programmer wishes to know if one particular flag is set or not set and does not care about the values of the others.

This is done by deliberately inventing a value which only allows the particular flag of interest to have a non-zero value and then ANDing that value with the flag register. For example: in symbolic language:

```
MASK = 00000001
VALUE1 = 10011011
VALUE2 = 10011100
MASK & VALUE1 == 00000001
MASK & VALUE2 == 00000000
```

The zeros in the mask *masks off* the first seven bits and leave only the last one to reveal its true value. Alternatively, masks can be built up by specifying several flags:

```
FLAG1 = 00000001
FLAG2 = 00000010
FLAG3 = 00000100
MESSAGE = FLAG1 | FLAG2 | FLAG3
MESSAGE == 00000111
```

It should be emphasized that these expressions are only written in symbolic language: it is not possible to use binary values in C. The programmer must convert to hexadecimal, octal or denary first.

Example

A simple example helps to show how logical masks and shift operations can be combined. The first programme gets a denary number from the user and converts it into binary. The second programme gets a value from the user in binary and converts it into hexadecimal.

```
/* Bit Manipulation #1 */
   /* Convert denary numbers into binary */
   /* Keep shifting i by one to the left */
   /* and test the highest bit. This does*/
   /* NOT preserve the value of i */
```

```
#include <stdio.h>
#define NUMBEROFBITS 8
main ()
{ short i,j,bit,;
   short MASK = 0x80;
printf ("Enter any number less than 128: ");
scanf ("%h", &i);
if (i > 128)
   {
   printf ("Too big\n");
   return (0);
   }
printf ("Binary value = ");
for (j = 0; j < NUMBEROFBITS; j++)
   {
   bit = i & MASK;
   printf ("%1d",bit/MASK);
   i <<= 1;
   }
printf ("\n");
}
         /* end */
```

Output:

```
Enter any number less than 128: 56
Binary value = 00111000
Enter any value less than 128: 3
Binary value = 00000011
```

Example:

```
/* Bit Manipulation #2 */
   /* Convert binary numbers into hex */
#include <stdio.h>
#define NUMBEROFBITS 8
main ()
{ short j,hex = 0;
   short MASK;
   char binary[NUMBEROFBITS];
printf ("Enter an 8-bit binary number: ");
for (j = 0; j < NUMBEROFBITS; j++)
   {
   binary[j] = getchar();
   }
for (j = 0; j < NUMBEROFBITS; j++)
   {
```

```
    hex <<= 1;
    switch (binary[j])
        {
        case '1': MASK = 1;
        break;
        case '0': MASK = 0;
        break;
        default:printf("Not binary\n");
                return(0);
        }
    hex |= MASK;
    }
printf ("Hex value = %1x\n",hex);
}
            /* end */
```

Example:

```
Enter any number less than 128: 56
Binary value = 00111000
Enter any value less than 128: 3
Binary value = 00000011
```

VIRTUAL MACHINE AND CLIENT SERVER

Virtual Machine

A virtual machine (VM) abides an operating system OS or conduct environment that is embedded on software which copies consecrated hardware. The end user embraces the equivalent experience on a virtual machine as they would acquire on dedicated hardware.

Individualized software designated a hypervisor copies the PC client or server's CPU, memory, hard disk, network as well as other hardware resources collectively, allowing virtual appliances to participate the resources. The hypervisor can copy integral virtual hardware platforms that are occasional from each other, assigning virtual machines to run Linux as well as Windows server operating systems on the identical underlying physical aggregation.

Virtualization conserves costs by depreciating the need for physical hardware systems. Virtual machines additional desirably use hardware, which lowers the quantities of hardware as well as associated maintenance costs, along with reduces power furthermore cooling demand. They also allay management due to virtual hardware does not collapse.

Administrators can take advantage of virtual circumstances to simplify backups, disaster recovery, new deployments as well as elementary system administration tasks.

Virtual machines do not constrain distinguished hypervisor-specific hardware. Virtualization appears although require more bandwidth, storage along with processing capacity than a conventional server or desktop if the physical hardware is going to host multiple running virtual machines.

VMs can easily move, be copied and reassigned between host servers to optimize hardware resource utilization. Because VMs on a physical host can consume unequal resource quantities, IT professionals must balance VMs with available resources.

Client Server

Client/server is a programme relationship in which one programme (the client) requests a service or resource from another programme (the server). It is seen that in client/server model, the programmes are used by single computer only.

It serves as an important concept for networking. Here, the client makes a connection with the server through local area network (LAN) or wide-area network (WAN) like Internet. After clearing the client's request, the connection gets terminated. In this case, Web browser serves as a client programme which further appeals for a service from the server. The service and resource of the server will show the delivery of such Web page.

Computer assignments in which the server accomplishes a request created by a client are very customary furthermore the client/server model has serve one of the main concepts of network computing. Most business approaches facilitate the client/server model as appears acts the Internet's core programme, TCP/IP.

For exemplary, when you examine a bank account from the computer, a client approximation in computer overtures a request to a server programme at the bank which in twist forward a approach to its own client programme and conveys a request to a database server at another bank computer. Once the account balance sheet has been acquired from the database, it is acknowledged back to the bank data client, who in turn applies it back to the client in his/her personal computer that displays the information.

Both client programmes as well as server programmes are usual constituent of a larger programme or application. On account of multiple client programmes participate the services of the equivalent server programme, a special server identified a daemon may be charged due to anticipate client requests. In marketing, the client/server had been once used to differentiate allocated computing by personal computers (PCs) from the monolithic, concentrated computing model exercised by mainframes. This differentiation has largely evaporated, although, as mainframes along with their applications possess additionally turned to the client/server model further become part of network computing.

Types of Operating System

There are abundant Operating Systems those monopolize be constructed for functioning the performances which are demanded by the user. There are many different types of Operating Systems which acquire the ability to behave the entreaties acquired from different approach. The Operating system can behave in a unique operation and furthermore multiple movements at duration, so there are many categories of operating systems those are arranged by utilizing their working mechanisms.

There are many types of operating system such as:

- Serial Processing
- Batch Processing
- Multi-Programming
- Real Time System
- Distributed Operating System
- Multiprocessing
- Parallel operating systems.

Real Time Systems

There appears additionally an operating system which is comprehended as Real Time Processing System in which duration is already adjusted. Indicates duration to show the after-effects after acquiring has adjusted by the Processor or CPU.

Real Time System is exercised at those areas in which we binds higher along with well-timed return. Such categories of approaches are exercised in reservation, so when we discriminate the demand, the CPU will conduct at that duration.

There are two types of Real Time System:

- Hard Real Time System: In Hard Real Time System, time is fixed and we can't change any moments of time of processing as the CPU will process the data as we enter it.
- Soft Real Time System: In Soft Real Time System, some moments can be change as after giving the command to CPU, the CPU will perform the operation after certain microseconds.

Multi-user System

As we comprehend that in case of Batch Processing System, there are results many jobs by the System. The System foremost compose a batch furthermore and will accomplish all jobs which gets saved in the Batch.

Also, the innermost difficulty is that if a mechanism or jobs needs an input as well as output operation, in such case, it is not achievable and there will be the wastage of the duration when composing the batch processing as CPU will remain idle during the particular time.

Although with the help of multi programming we can achieve multiple programmes on the system at a duration as besides in multi-programming, the CPU determination never gets idle, so with the help of Multi-Programming we can achieve ample algorithms on system when functioning with programme and can acknowledge the supplement or other programme for sprinting extra CPU that will at that time behaves as secondary programme following the completion of original programme. Also in this, we can further differentiate our input means which a user can additionally interact with the system.

The multi-programming operating systems never utilize many cards on account of approach that is accessed on the spot by the user. Since the Operating System also utilizes the process of allocation and de-allocation of the memory which shows providing memory space to all the running and all waiting processes. There must be the proper management of all the running jobs.

Distributed System

Distributed means data is stored and processed on multiple locations. When a data is stored on to the multiple computers, those are placed in different locations. Distributed in terms of network means, network collections of computers connected with each other.

If you want to take some data from other computer, then we uses the distributed processing system, as we can also insert and remove the data from one location to another location.

In this, data is shared between many users, and we can also access all the Input and Output Devices by Multiple Users.

Other Types

There are other worthwhile types of operating systems not made by Microsoft. The greatest problem with these operating systems lies in the fact that not as many application programmes are written for them. However if you can get the type of application programmes you are looking for, one of the systems listed below may be a good choice.

- *Unix*: A system that has been around for many years and it is very stable. It is primary used to be a server rather than a workstation and should not be used by anyone who does not understand the system. It can be difficult to learn. Unix must normally run an a computer made by the same company that produces the software.
- *Linux*: Linux is similar to Unix in operation but it is free. It also should not be used by anyone who does not understand the system and can be difficult to learn.
- *Apple MacIntosh*: Most recent versions are based on Unix but it has a good graphical interface so it is both stable (does not crash often or have as many software problems as other systems may have) and easy to learn. One drawback to this system is that it can only be run on Apple produced hardware.

SYSTEM VIRTUAL MACHINE

In computing, a system virtual machine is a virtual machine that provides a complete system platform and supports the execution of a complete operating system (OS). These usually emulate an existing architecture, and are built with the purpose of either providing a platform to run programs where the real hardware is not available for use (for example, executing on otherwise obsolete platforms), or of having multiple instances of virtual machines leading to more efficient use of computing resources, both in terms of energy consumption and cost effectiveness (known as hardware virtualization, the key to a cloud computing environment), or both. A VM was originally defined by Popek and Goldberg as "an efficient, isolated duplicate of a real machine".

System virtual machines

System virtual machine advantages:

- Multiple OS environments can co-exist on the same primary hard drive, with a virtual partition that allows sharing of files generated in either the "host" operating system or "guest" virtual environment. Adjunct software installations, wireless connectivity, and remote replication, such as printing and faxing, can be generated in any of the guest or host operating systems. Regardless of the system, all files are stored on the hard drive of the host OS.
- Application provisioning, maintenance, high availability and disaster recovery are inherent in the virtual machine software selected.
- Can provide emulated hardware environments different from the host's instruction set architecture (ISA), through emulation or by using just-in-time compilation.

The main disadvantages of VMs are:

- A virtual machine is less efficient than an actual machine when it accesses the host hard drive indirectly.
- When multiple VMs are concurrently running on the hard drive of the actual host, adjunct virtual machines may exhibit a varying and/or unstable performance (speed of execution and malware protection). This depends on the data load imposed on the system by other VMs, unless the selected VM software provides temporal isolation among virtual machines.
- Malware protections for VMs are not necessarily compatible with the "host", and may require separate software.

Multiple VMs running their own guest operating system are frequently engaged for server consolidation in order to avoid interference from separate VMs on the same actual machine platform.

The desire to run multiple operating systems was the initial motivation for virtual machines, so as to allow time-sharing among several single-tasking operating systems.

In some respects, a system virtual machine can be considered a generalization of the concept of virtual memory that historically preceded it. IBM's CP/CMS, the first systems to allow full virtualization, implemented time sharing by providing each user with a single-user operating system, the CMS. Unlike virtual memory, a system virtual machine entitled the user to write privileged instructions in their code. This approach had

certain advantages, such as adding input/output devices not allowed by the standard system.

As technology evolves virtual memory for purposes of virtualization, new systems of memory overcommitment may be applied to manage memory sharing among multiple virtual machines on one actual computer operating system. It may be possible to share "memory pages" that have identical contents among multiple virtual machines that run on the same physical machine, what may result in mapping them to the same physical page by a technique known as Kernel SamePage Merging. This is particularly useful for read-only pages, such as those that contain code segments; in particular, that would be the case for multiple virtual machines running the same or similar software, software libraries, web servers, middleware components, etc.

The guest operating systems do not need to be compliant with the host hardware, thereby making it possible to run different operating systems on the same computer (e.g., Microsoft Windows, Linux, or previous versions of an operating system) to support future software.

The use of virtual machines to support separate guest operating systems is popular in regard to embedded systems. A typical use would be to run a real-time operating system simultaneously with a preferred complex operating system, such as Linux or Windows. Another use would be for novel and unproven software still in the developmental stage, so it runs inside a sandbox. Virtual machines have other advantages for operating system development, and may include improved debugging access and faster reboots.

Techniques

Different virtualization techniques are used, based on the desired usage. *Native execution* is based on direct virtualization of the underlying raw hardware, thus it provides multiple "instances" of the same architecture a real machine is based on, capable of running complete operating systems. Some virtual machines can also emulate different architectures and allow execution of software applications and operating systems written for another CPU or architecture. Operating-system-level virtualization allows the resources of a computer to be partitioned via kernel's support for multiple isolated user space instances, which are usually called containers and may look and feel like real machines to the end users. Some computer architectures are capable of hardware-assisted virtualization, which enables

efficient full virtualization by using virtualization-specific hardware capabilities, primarily from the host CPUs.

Virtualization of the underlying raw hardware (native execution)

This approach is described as full virtualization of the hardware, and can be implemented using a type 1 or type 2 hypervisor: a type 1 hypervisor runs directly on the hardware, and a type 2 hypervisor runs on another operating system, such as Linux or Windows.

Each virtual machine can run any operating system supported by the underlying hardware. Users can thus run two or more different "guest" operating systems simultaneously, in separate "private" virtual computers.

The pioneer system using this concept was IBM's CP-40, the first (1967) version of IBM's CP/CMS (1967–1972) and the precursor to IBM's VM family (1972–present). With the VM architecture, most users run a relatively simple interactive computing single-user operating system, CMS, as a "guest" on top of the VM control program (VM-CP). This approach kept the CMS design simple, as if it were running alone; the control program quietly provides multitasking and resource management services "behind the scenes". In addition to CMS communication and other system tasks are performed by multitasking VMs (RSCS, GCS, TCP/IP, UNIX), and users can run any of the other IBM operating systems, such as MVS, even a new CP itself or now z/OS. Even the simple CMS could be run in a threaded environment (LISTSERV, TRICKLE). z/VM is the current version of VM, and is used to support hundreds or thousands of virtual machines on a given mainframe. Some installations use Linux on IBM Z to run Web servers, where Linux runs as the operating system within many virtual machines.

Full virtualization is particularly helpful in operating system development, when experimental new code can be run at the same time as older, more stable, versions, each in a separate virtual machine. The process can even be recursive: IBM debugged new versions of its virtual machine operating system, VM, in a virtual machine running under an older version of VM, and even used this technique to simulate new hardware.

The standard x86 instruction set architecture as used in the modern PCs does not actually meet the Popek and Goldberg virtualization requirements. Notably, there is no execution mode where all sensitive

machine instructions always trap, which would allow per-instruction virtualization.

Despite these limitations, several software packages have managed to provide virtualization on the x86 architecture, even though dynamic recompilation of privileged code, as first implemented by VMware, incurs some performance overhead as compared to a VM running on a natively virtualizable architecture such as the IBM System/370 or Motorola MC68020. By now, several other software packages such as Virtual PC, VirtualBox, Parallels Workstation and Virtual Iron manage to implement virtualization on x86 hardware.

Intel and AMD have introduced features to their x86 processors to enable virtualization in hardware.

As well as virtualization of the resources of a single machine, multiple independent nodes in a cluster can be combined and accessed as a single virtual NUMA machine.

Emulation of a non-native system

Virtual machines can also perform the role of an emulator, allowing software applications and operating systems written for another computer processor architecture to be run.

Operating-system-level virtualization

Operating-system-level virtualization is a server virtualization technology which virtualizes servers on an operating system (kernel) layer.

It can be thought of as partitioning: a single physical server is sliced into multiple small partitions (otherwise called virtual environments (VE), virtual private servers (VPS), guests, zones, etc.); each such partition looks and feels like a real server, from the point of view of its users.

For example, Solaris Zones supports multiple guest operating systems running under the same operating system such as Solaris 10. Guest operating systems can use the same kernel level with the same operating system version, or can be a separate copy of the operating system with a different kernel version using Solaris Kernel Zones. Solaris native Zones also requires that the host operating system is a version of Solaris; other operating systems from other manufacturers are not supported. However, Solaris Branded Zones would need to be used to have other operating systems as zones.

Another example is System Workload Partitions (WPARs), introduced in version 6.1 of the IBM AIX operating system. System WPARs are software partitions running under one instance of the global AIX OS environment.

The operating system level architecture has low overhead that helps to maximize efficient use of server resources. The virtualization introduces only a negligible overhead and allows running hundreds of virtual private servers on a single physical server. In contrast, approaches such as full virtualization (like VMware) and paravirtualization (like Xen or UML) cannot achieve such level of density, due to overhead of running multiple kernels. From the other side, operating system-level virtualization does not allow running different operating systems (i.e., different kernels), although different libraries, distributions, etc. are possible. Different virtualization techniques are used, based on the desired usage. Native execution is based on direct virtualization of the underlying raw hardware, thus it provides multiple "instances" of the same architecture a real machine is based on, capable of running complete operating systems. Some virtual machines can also emulate different architectures and allow execution of software applications and operating systems written for another CPU or architecture. Operating-system-level virtualization allows the resources of a computer to be partitioned via kernel's support for multiple isolated user space instances, which are usually called containers and may look and feel like real machines to the end users. Some computer architectures are capable of hardware-assisted virtualization, which enables efficient full virtualization by using virtualization-specific hardware capabilities, primarily from the host CPUs.

Virtualization-enabled hardware

Examples of virtualization-enabled hardware include the following:

- Alcatel-Lucent 3B20D/3B21D emulated on commercial off-the-shelf computers with 3B2OE or 3B21E system
- ARM TrustZone
- Boston Circuits gCore (grid-on-chip) with 16 ARC 750D cores and Time-machine hardware virtualization module.
- Freescale PowerPC MPC8572 and MPC8641D
- IBM System/360 Model 67, System/370, System/390, and zSeries mainframes
- IBM Power Systems
- x86:

  - AMD-V (formerly code-named Pacifica)
  - Intel VT-x (formerly code-named Vanderpool)
- HP vPAR and cell based nPAR
- GE and Honeywell Multics systems
- Honeywell 200/2000 systems Liberator replacing IBM 14xx systems
- Honeywell Level 62/64/66
- IBM System/360 and System/370 models with emulators supporting programs for older IBM systems
- Honeywell Level 6 minicomputers emulated predecessor 316/516/716 minis
- Oracle Corporation (previously Sun Microsystems) SPARC sun4v (SPARC M6, T5, T4, T3, UltraSPARC T1 and T2) – utilized by Oracle VM Server for SPARC, also known as "Logical Domains"
- Xerox Sigma 6 CPUs were modified to emulate GE/Honeywell 600/6000 systems

HARDWARE VIRTUALIZATION

Hardware virtualization is the virtualization of computers as complete hardware platforms, certain logical abstractions of their componentry, or only the functionality required to run various operating systems. Virtualization hides the physical characteristics of a computing platform from the users, presenting instead an abstract computing platform. At its origins, the software that controlled virtualization was called a "control program", but the terms "hypervisor" or "virtual machine monitor" became preferred over time.

Concept

The term "virtualization" was coined in the 1960s to refer to a virtual machine (sometimes called "pseudo machine"), a term which itself dates from the experimental IBM M44/44X system. The creation and management of virtual machines has been called "platform virtualization", or "server virtualization", more recently.

Platform virtualization is performed on a given hardware platform by *host* software (a *control program*), which creates a simulated computer environment, a *virtual machine* (VM), for its *guest* software. The guest software is not limited to user applications; many hosts allow the execution of complete operating systems. The guest software executes as if it were

running directly on the physical hardware, with several notable caveats. Access to physical system resources (such as the network access, display, keyboard, and disk storage) is generally managed at a more restrictive level than the *host* processor and system-memory. Guests are often restricted from accessing specific peripheral devices, or may be limited to a subset of the device's native capabilities, depending on the hardware access policy implemented by the virtualization host.

Virtualization often exacts performance penalties, both in resources required to run the hypervisor, and as well as in reduced performance on the virtual machine compared to running native on the physical machine.

Reasons for virtualization

- In the case of server consolidation, many small physical servers are replaced by one larger physical server to decrease the need for more (costly) hardware resources such as CPUs, and hard drives. Although hardware is consolidated in virtual environments, typically OSs are not. Instead, each OS running on a physical server is converted to a distinct OS running inside a virtual machine. Thereby, the large server can "host" many such "guest" virtual machines. This is known as Physical-to-Virtual (P2V) transformation.
- In addition to reducing equipment and labor costs associated with equipment maintenance, consolidating servers can also have the added benefit of reducing energy consumption and the global footprint in environmental-ecological sectors of technology. For example, a typical server runs at 425 W and VMware estimates a hardware reduction ratio of up to 15:1.
- A virtual machine (VM) can be more easily controlled and inspected from a remote site than a physical machine, and the configuration of a VM is more flexible. This is very useful in kernel development and for teaching operating system courses, including running legacy operating systems that do not support modern hardware.
- A new virtual machine can be provisioned as required without the need for an up-front hardware purchase.
- A virtual machine can easily be relocated from one physical machine to another as needed. For example, a salesperson going to a customer can copy a virtual machine with the demonstration software to their laptop, without the need to transport the physical computer. Likewise, an error inside a virtual machine does not harm the host system, so there is no risk of the OS crashing on the laptop.

- Because of this ease of relocation, virtual machines can be readily used in disaster recovery scenarios without concerns with impact of refurbished and faulty energy sources.

However, when multiple VMs are concurrently running on the same physical host, each VM may exhibit varying and unstable performance which highly depends on the workload imposed on the system by other VMs. This issue can be addressed by appropriate installation techniques for temporal isolation among virtual machines.

There are several approaches to platform virtualization.

Examples of virtualization use cases:

- Running one or more applications that are not supported by the host OS: A virtual machine running the required guest OS could permit the desired applications to run, without altering the host OS.
- Evaluating an alternate operating system: The new OS could be run within a VM, without altering the host OS.
- Server virtualization: Multiple virtual servers could be run on a single physical server, in order to more fully utilize the hardware resources of the physical server.
- Duplicating specific environments: A virtual machine could, depending on the virtualization software used, be duplicated and installed on multiple hosts, or restored to a previously backed-up system state.
- Creating a protected environment: If a guest OS running on a VM becomes damaged in a way that is not cost-effective to repair, such as may occur when studying malware or installing badly behaved software, the VM may simply be discarded without harm to the host system, and a clean copy used upon rebooting the guest .

Full virtualization

In full virtualization, the virtual machine simulates enough hardware to allow an unmodified "guest" OS designed for the same instruction set to be run in isolation. This approach was pioneered in 1966 with the IBM CP-40 and CP-67, predecessors of the VM family.

Hardware-assisted virtualization

In hardware-assisted virtualization, the hardware provides architectural support that facilitates building a virtual machine monitor and allows guest OSs to be run in isolation. Hardware-assisted virtualization

was first introduced on the IBM System/370 in 1972, for use with VM/370, the first virtual machine operating system.

In 2005 and 2006, Intel and AMD provided additional hardware to support virtualization. Sun Microsystems (now Oracle Corporation) added similar features in their UltraSPARC T-Series processors in 2005.

In 2006, first-generation 32- and 64-bit x86 hardware support was found to rarely offer performance advantages over software virtualization.

Paravirtualization

In paravirtualization, the virtual machine does not necessarily simulate hardware, but instead (or in addition) offers a special API that can only be used by modifying the "guest" OS. For this to be possible, the "guest" OS's source code must be available. If the source code is available, it is sufficient to replace sensitive instructions with calls to VMM APIs (e.g.: "cli" with "vm_handle_cli()"), then re-compile the OS and use the new binaries. This system call to the hypervisor is called a "hypercall" in TRANGO and Xen; it is implemented via a DIAG ("diagnose") hardware instruction in IBM's CMS under VM (which was the origin of the term *hypervisor*)..

Operating-system-level virtualization

In operating-system-level virtualization, a physical server is virtualized at the operating system level, enabling multiple isolated and secure virtualized servers to run on a single physical server. The "guest" operating system environments share the same running instance of the operating system as the host system. Thus, the same operating system kernel is also used to implement the "guest" environments, and applications running in a given "guest" environment view it as a stand-alone system.

Hardware virtualization disaster recovery

A disaster recovery (DR) plan is often considered good practice for a hardware virtualization platform. DR of a virtualization environment can ensure high rate of availability during a wide range of situations that disrupt normal business operations. In situations where continued operations of hardware virtualization platforms is important, a disaster recovery plan can ensure hardware performance and maintenance requirements are met. A hardware virtualization disaster recovery plan involves both hardware and software protection by various methods, including those described below.

Tape backup for software data long-term archival needs. This common method can be used to store data offsite, but data recovery can be a difficult and lengthy process. Tape backup data is only as good as the latest copy stored. Tape backup methods will require a backup device and ongoing storage material.

Whole-file and application replication

The implementation of this method will require control software and storage capacity for application and data file storage replication typically on the same site. The data is replicated on a different disk partition or separate disk device and can be a scheduled activity for most servers and is implemented more for database-type applications.

Hardware and software redundancy

This method ensures the highest level of disaster recovery protection for a hardware virtualization solution, by providing duplicate hardware and software replication in two distinct geographic areas.

Bibliography

Baumann, P. R.: *Computer Assisted Instruction Programs in Geography: Five Climate Programs*, Oneonta, New York, 1970.

Bhaskara Rao: *Military Conversion : Impact on Science and Technology*, Discovery, Delhi, 2003.

Bird, R., Brown, R., & Howard, P.: *Business Finance*, Roseville, 1990

Bird, Richard: *Federal Finance in Comparative Perspective*, Toronto, Canadian Tax Foundation, 1986.

Damrosch, L.F.: *Law and Force in the New International Order*, Boulder, Westview Press, 1991.

Davies .A.: *Telecommunications and Politics. The Secentralised Alternative*, London, Pinter, 1994.

Devlin B. : *Data Warehouse: From Architecture to Implementation,* Addison-Wesley, 1997.

Goldsmith, W.: *The Financial Development of India, 1860-1977*, New Haven, Yale University Press, 1983.

Holzmann, Gerald J. and Bjorn Pehrson: *The Early History of Data Networks,* Los Alamitos, CA, IEEE Computer Society Press, 1994.

Hurewitz, Jacob C.: *Diplomacy in the Near and Middle East: A Documentary Record 1535-1914*, Princeton, New Jersey, 1956.

James R.: *The Control Revolution: Technological and Economic Origins of the Information Society*, Cambridge, MA: Harvard University Press, 1986.

James W.: *The Computer in the United States: From Laboratory to Market, 1930-1960*, Armonk, NY: M. E. Sharpe, 1993.

John Keegan, *Intelligence in War*. New York: Knopf, 2003.

Katherine Davis: *The Computer Establishment*, New York: McGraw-Hill, 1981.

Kidder, Tracy: *Soul of a New Machine*, New York, Avon, 1981.

Kimball Ralph *: The Data Warehouse Toolkit*, , Wiley, 1996.

Laura Brav: *The Practical Guide to Humanitarian Law,* Lanham, MD: Rowman & Littlefield Publishers, Inc., 2007.

MacKinley, A.C.: *The Econometrics of Financial Markets,* Princeton University Press, 1996.

Margaret A. Ellis: *Designing and Coding Reusable C++,* Addison-Wesley, 1995.

Müller-Brockmann, Josef: *The Graphic Designer and His Design Problems.* New York: Hastings House, 1983.

Peter Lancaster and Kestutis Salkauskas: *"Curve and Surface Fitting, An Introduction*," Academic Press, New York, 1986.

Rand, Paul. *Design Form and Chaos.* New Haven: Yale University Press, 1993.

Rao, N. Venkateshwara and Prashant K. Mathur: *Handbook of Animation, Cartoon and Multimedia*, Kanishka, Delhi, 2008.

Ravishankar, S. : *Computer Graphics OOPS with C++*, Himalaya, Delhi, 2000.

Ray Harryhausen and Tony Dalton, *A Century of Stop Motion Animation: From Mélès to Aardman*. New York: Watson-Guptil Publications, 2008.

Reiley, A. C.: *Onward Industry*, New York, 1931.

Ruchi Mishra: *Computer Graphics*, Global Vision Pub, Delhi, 2010.

Samara, Timothy. *Design Elements: A Graphic Style Manual*. Rockport Publishers; 2007.

Shukla, A.S. : *Handbook of Multimedia and Animation*, Rajat Pub, Delhi, 2008.

Sidnie Feit: *TCP/IP: Architecture, Protocols, and Implementation.* McGraw-Hill, 1993.

Slagmulder, R. : *Interorganizational Cost Management*, Productivity Press, 1999.

Stephen Cavalier, *The World History of Animation*. Berkley, California: University of California Press, 2011.

Stinson, R. : *Cryptography: Theory and Practice,* Chapman & Hall/CRC, 2006.

Stone, H.S.: *Introduction to Computer Architecture*. Chicago, Ill.: Science Research Associates, 1980.

Stross, Randall: *The Microsoft Way, The Real Story of How the Company Outsmarts Its Competition*, New York: Addison-Wesley, 1996.

Tayal, Sumit Prakash: *Computer Network*, Laxmi Pub, Delhi, 2009.

Teukolsky, S.A. and W.T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, Cambridge, England, 1988.

Ullman J.D.; *The Design and Analysis of Computer Algorithms*; Addison-Wesley, 1974.

Vince, John: *"3-D Computer Animation*," Addison-Wesley, New York, 1992.

Warren S.: *The Embodiments of Mind*. MIT Press, Cambridge, 1965.

William Aspray: *Computer, A History of the Information Machine*, New York, Basic Books, 1996.

Ye, Nong: *The Handbook of Data Mining*, Mahwah, NJ: Lawrence Erlbaum, 2003.

Zhidkov, N. P. : *Computing Methods, Pergamon Press*, Oxford and Addison-Wesley, Reading, Mass., 1965.

Index

❑❑❑

ABOUT THE AUTHOR

Ahmad Ali AlZubi is a full Professor at Computer Science Department, King Saud University, Saudi Arabia. He obtained his PhD from National Technical University of Ukraine (NTUU) in Computer Networks Engineering in 1999. His current research interests include but not limited to Computer Networks, Grid Computing, Cloud Computing, AI, Machine learning and Deep Learning and their applications in various fields, and services automation. He has also gained valuable industry experience, having worked as a consultant and a member of the Saudi National Team for E-Government in Saudi Arabia. He has author a book title Heart Disease Prediction Using Machine Learning having ISBN: 978-81-19477-42-5

Abdulrhman A. Alkhanifer is a computer scientist with a passion for digital transformation. He leverages his academic background and industry experience to drive innovation in this field. Presently he is an Assistant Professor in the Computer Science Department, King Saud University, Saudi Arabia.

Beyond academia, he holds leadership positions that fuel digital transformation efforts. He serves as the Dean of E-Transactions and Telecommunication at King Saud University, and is also the CEO of Knowledge Developers, a university-owned company specializing in digital solutions. Dr. Alkhanifer's expertise extends beyond his own institutions. He is a sought-after advisor for both government and private entities.

His research interests reflect his focus on digital transformation, lying at the intersection of software engineering, user experience (UX) design, machine learning, and artificial intelligence (AI).

ABOUT THE BOOK

Artificial Intelligence (AI) involves creating systems that exhibit intelligent behavior, mimicking cognitive functions such as learning, reasoning, problem-solving, perception, and language understanding. At its core, AI seeks to develop algorithms and models that enable machines to perform tasks typically requiring human intelligence. These tasks range from simple pattern recognition and data processing to complex decision-making and autonomous operation. Intelligent Systems, encompassing AI, extend their reach beyond traditional AI applications. This broader term includes technologies like expert systems, knowledge-based systems, and robotics. Intelligent Systems aim not only to replicate human-like intelligence but also to adapt and interact intelligently with dynamic environments. Artificial Intelligence (AI) aims to create systems capable of performing tasks that typically require human intelligence, such as learning, reasoning, and problem-solving. Key technologies include Machine Learning (ML), which enables systems to learn from data; Natural Language Processing (NLP), which allows for understanding and generating human language; and Computer Vision, which interprets visual information. AI-powered robotics combine these technologies to navigate and interact with environments autonomously. While AI presents vast opportunities for innovation and efficiency, it also raises ethical considerations regarding privacy, security, and fairness, underscoring the need for responsible development and deployment. Artificial Intelligence for Beginners explores the foundational concepts, technologies, and applications that enable machines to perform tasks requiring human-like intelligence.

IARA PUBLICATION

ISBN 978-81-19481-79-8

India | UAE | Nigeria | Malaysia | Montenegro | Iraq | Egypt | Thailand | Uganda | Philippines | Indonesia

IARA Publication || www.iarapublication.com || info@iarapublication.com