Subject: Artificial Intelligence
Posted by Wayne Parham on Sun, 01 Oct 2023 22:59:08 GMT
View Forum Message <> Reply to Message

I was a boy in the 1960s, and watched the first moon landing on television. The Apollo spaceship didn't have warp drive like the Enterprise from the Star Trek television series I loved, but the moon landing had the benefit of being a non-fiction, real-life actual event. Those were incredible times.

And becoming a teenager in the 1970s was just as exciting. I was too young to appreciate the difficulties of the economy - probably America's first difficult time since WWII - but I was not too young to appreciate the excitements of advances in electronics and computer technologies. It was the perfect time for a boy like me to come of age, because I was able to afford one of the first kinds of computers that most anyone could own, a microcomputer. My first one was a Synertek System's Sym-1, which was a fairly competent machine with a BASIC interpreter in ROM and enough memory to write simple programs. It also has a lot of digital I/O lines, which made it a perfect system for robotics, which I loved.

In 1976, the first Star Wars movie came out. I loved the robots, and thought to myself, I could build R2-D2. Easy to build, really, pretty much just a platform with the Sym-1 for a brain, two stepper-motors to drive two wheels and a trailing passive castor wheel for a third. The June '77 issue of Byte Magazine had an article about a similar robot called NEWT that had a camera for input, and could recognize a wall socket to plug itself into when low on batteries. All super cool and so do-able. The software was rules-based, an importance-rated hierarchy of status and environmental possibilities and appropriate responses. I even designed a miniature version with reduced capabilities that I considered selling as a toy.

A few things happened around that same time that had a large influence upon me. First was Douglas Hofstadter's book, "Gödel, Escher, Bach." That was both inspirational and philosophical, and I loved some parts and hated others but it definitely made an impact, especially "Ant Fugue." Another was learning about neural networks and seeing Marvin Minsky's Artificial Intelligence lab at MIT.

Sadly, my first introduction to neural networks was at a time they were being somewhat dismissed but within a few years, new approaches were being discussed like a methodology called subsumption architecture and improved multi-layered neural networks using backpropagation for training. Those both caught my attention.

Rules-based software is probably the most common form of machine intelligence, and it's called "symbolic artificial intelligence." Most software - really, whether it is supposed to be "artificial intelligence" or not - is a set of rules codified in a computer program. That's what the software in a general purpose computer does: It takes input of some kind, then makes some decisions based on the input and takes actions. These inputs can be from sensors and actions can be made by motor movements. Or the inputs can be from a keyboard and the action can be to display images on a screen.

One popular type of artificial intelligence program is called an "expert system," which uses a series of rules to suggest possibilities in a particular field that replicate what an expert specialist in

that field might suggest. At its core is database or "knowledge base" and a rules engine, sometimes also called an inference engine.

There's another kind of machine intelligence called "sub-symbolic artificial intelligence" that has been a little more controversial over the years. It is the one I'm most fascinated by because it is essentially built with logic circuits that most closely emulate neurons in the human (and other animals) brains and nervous systems. Systems with these simulated neurons are called "neural networks." This approach has been around as long as symbolic AI, but it has come and gone in popularity over the years.

You can visualize the difference between symbolic and sub-symbolic systems using an analog of your own mental processes. When you think a conscious thought - perhaps balancing your checkbook or writing a document - you are using symbolic processing. Your thoughts are like an internal dialog of instructions, plans of something you intend to do. You carry those plans out mentally, taking some actions or making mental notes of what to do next. All of those instructions can be "accessed" - you can describe the process verbally or you can write it down.

On the other hand, when you form a habit or when you practice something physical, you are using sub-symbolic processing. You aren't so much thinking about what you need to do, but are rather "just doing it." You are practicing the task you need to perform, improving with each attempt. The things you "learn" are subtle, minute corrections. It's not so easy to describe the process or the ways you have improved. In fact, you do not know exactly what corrections you have made. You just seem to get better with practice.

This is very much the same way these kinds of machine intelligence systems are codified. Symbolic systems are essentially rules based and the rules can be easily recorded, modified and described. Sub-symbolic systems are iteratively trained, with minute corrections made after each iteration. The resulting state is a little less easy to quantify or to modify in a deterministic fashion. You can continually train a neural network - and you can record its entire state - but it isn't as easy to describe like you would a set of rules.

More traditional rules-based systems have seen less volatility in their popularity because all conventional computer programs are rules-based, whether or not they are described as AI or not. I think this has been one of the reasons that symbolic AI has been more common, at least over the long haul. There are a lot of applications that have been written in the last fifty years that used symbolic AI in some form. One could make the argument that all computer programs are symbolic AI, in at least some sense.

Sub-symbolic AI is controversial - and interesting, in my opinion - for the very reason that it is useful. You don't program a neural network the same way you do a rules-based system. Instead, you "train" it by presenting it with data and allowing it to make random choices based on that data at first. Each training iteration, "correct" decisions are recorded by modifying each neuron's input in a way that favors the state of that input. Incorrect decisions are recorded by modifying each neuron's inputs to oppose their input states. The recorded states are called weighting and the process of doing it is called back-propagation.

Each neuron has as many inputs as nodes that it needs to analyze. Those nodes can be sensor inputs or they can be outputs from other neurons. The output is generated based on the total of

all inputs and their weights. So for example, if the inputs are a two-dimensional array that is formed by the pixels of an image, then input weights would be set to expect something in some pixels, to expect nothing in other pixels and an in-between weight for pixels that don't matter as much. This is a common case for alpha-numeric character recognition, where each neuron is looking for exactly one character.

The thing is, training is a lengthy process that requires a lot of data. And the end result isn't really a concrete network "state" that is easy to analyze. It's more like a fuzzy soup of weighting data. Another problem is the training conditions often create unusual side effects, that are a lot like "edge cases" in more traditional programs.

An edge case in a rules-based program is something that is rare and that might not have been considered by the programmers. In such a scenario, the program may not have any rule to deal with the edge case and its default may not be appropriate.

Similarly, a dataset used to program a neural network may have peculiarities that adjust the network weighting to make unusual choices. An example is something that happened to one AI research team, when training its image-recognition networks to identify a photo of a dog. Since dogs are often running when photos are taken of them, the photographs often have blurry backgrounds. So after training, the network could often identify dogs but then again, it often incorrectly identified other objects with blurry backgrounds as dogs.

The Large Language Model (LLM) approach used to make the "chatbots" that suddenly became so popular in 2022 can suffer a similar fate. The LLM approach is essentially a language parser that uses a neural network to scan and process sentences, having a large database of information - sometimes the whole internet or a large part of it - as its training datasource. Its training sometimes results in improper inferences, and so certain questions or lines of dialog cause it to respond with inaccurate answers or even utter nonsense. This is another example of the "blurry dog" problem, and the silly responses from the chatbots are called "hallucinations."

For these kinds of reasons, neural networks have been controversial. They often do incredible things, but then again, they sometimes make really bad (and sometimes embarrassing) decisions right when they are being demoed to an important stakeholder. This has caused them to fall out of favor more than once, creating what some have called "AI winters," where funding from government, scientific and corporate organizations dries up, seemingly overnight.

Where I personally have been most impressed by neural networks is projects where they have been used for complex locomotion training, like bipedal and quadrupedal walking, jumping and even more advanced movements such as acrobatics and dancing. If you think about it, that's a place where the training of a neural network makes a lot of sense. It's very effective for that kind of thing. It has proven to be very good at things like speech recognition and image recognition, provided it is trained enough to prevent the "blurry dog" scenario.

Another computing approach that I particularly like is called genetic algorithms. The idea is to start off with a set of randomly chosen procedural approaches or "individuals" to solve a problem, with each approach having a series of steps or rules. The individuals are "evolved" iteratively with each successive run being evaluated by determining the success of each individual approach after each iteration.

Success of an individual is determined by its "fitness," which is its effectiveness in solving the problem. Those that are most successful are selected for inclusion in the next iteration. Each chosen individual is modified or "mutated" before the next iteration by adding small random changes or by morphing with other successful individuals. The next generation of candidates is then used in the next iteration of the algorithm. The algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

Some people experimented with the idea of using rules-based symbolic AI but taking inputs and making them "fuzzy." The idea is that many - perhaps most - things aren't true or false but rather "mostly true" or "occasionally false" or just plain "maybe." Fuzzy logic attempts to address that by making inputs act more like an analog signal - something between "on" or "off." The decision-making process becomes more of a gradient and the output is then a percentage too. It can give an answer with an estimate of probability.

A similar method is called probabilistic machine learning, which is similar to fuzzy logic or could be said to be an extension of that approach. Probabilistic models are a class of machine learning algorithms for making predictions based on probability and statistics. These models identify uncertain relationships between variables in a data-driven manner to find patterns and trends inherent in the data.

One popular form of probabilistic model is called the Monte Carlo method, which is a sort of multiple probability simulation. It's a mathematical technique that is used to estimate the possible outcomes of an uncertain event. The Monte Carlo method is actually as old as modern computing itself, having been invented by John von Neumann and Stanislaw Ulam during World War II to improve decision making under uncertain conditions. It was named after a city that is known for its casinos - Monaco - because the element of chance is used in the modeling approach, similar to a game of roulette.

At its core, the Monte Carlo method uses a model containing several possible results which defines a probability distribution for any variable that has inherent uncertainty. Several iterations are run, recalculating results over and over, each time using a different set of random numbers between the minimum and maximum values. This procedure can be repeated thousands of times to produce a large number of likely outcomes. The more iterations that are run, the more accurate is the long-term prediction, at least in a sense of overall trends.

Subsumption architecture is another interesting approach, useful for physical control architectures. It's not a neural network, but it isn't a "top-down" rules approach either. Instead of guiding behavior by symbolic mental representations of the world, subsumption architecture couples sensory information to action selection in an intimate and bottom-up fashion.

It does this by decomposing the complete behavior into sub-behaviors. These sub-behaviors are organized into a hierarchy of layers. Each layer implements a particular level of overall behavior, and higher levels are able to subsume lower levels in order to create viable behavior. For example, a robot's lowest layer could be "avoid an object". The second layer would be "wander around", and that runs beneath the third layer "explore the world". Because a robot must have the ability to "avoid objects" in order to "wander around" effectively, the subsumption architecture

creates a system in which the higher layers utilize the lower-level competencies. The layers, which all receive sensor-information, work in parallel and generate outputs. These outputs can be commands to actuators, or signals that suppress or inhibit other layers.

Most AI systems do not use a singular approach, but rather combine several or at least a couple of those described above. Even when one architecture is dominant, others are generally used as sort of checks-and-balances or to aid functionality that isn't particularly feasible in the primary architecture. And in the case of neural networks and other approaches that are "trained," complex systemic behavior. In fact, that's usually the case and is one of the benefits but also one of the biggest challenges in artificial intelligence systems.
