Chapter 1: The Roots of Artificial Intelligence

Two Months and Ten Men at Dartmouth

The dream of creating an intelligent machine—one that is as smart or smarter than humans—is centuries old, but became part of modern science with the rise of digital computers. In fact, the ideas that led to the first programmable computers came out of mathematicians' attempts to understand human thought—particularly logic—as a mechanical process of "symbol manipulation". Digital computers are essentially symbol manipulators, pushing around combinations of the symbols '0' and '1'. To the pioneers of computing such as Alan Turing and John von Neumann, there were strong analogies between computers and the human brain, and it seemed obvious to them that intelligence could be captured in computer programs.

Most people in artificial intelligence trace the field's official founding to a small workshop in 1956 at Dartmouth College, organized by a young mathematician named John McCarthy.

In 1955, McCarthy, age twenty-eight, joined the mathematics faculty at Dartmouth. As an undergraduate, he had learned a bit about both psychology and the nascent field of "automata theory" (later to become computer science), and had become intrigued with the idea of creating a thinking machine. In graduate school in the mathematics department at Princeton, McCarthy met a fellow student, Marvin Minsky, who shared his fascination with the potential of intelligent computers. After graduating, McCarthy had short-lived stints at Bell Labs and IBM, where he collaborated respectively with Claude Shannon, the inventor of information theory, and Nathaniel Rochester, a pioneering electrical engineer. Once at Dartmouth, McCarthy convinced Minsky, Shannon, and Rochester to help him organize "a 2 month, 10 man study of artificial intelligence to be carried out during the summer of 1956."[1] The term "artificial intelligence" was McCarthy's invention; he wanted to distinguish this field from a related effort called "cybernetics".[2] McCarthy later admitted that no one really liked the name—after all,

the goal was *genuine*, not "artificial", intelligence—but "I had to call it something, so I called it 'Artificial Intelligence'."[3]

The four organizers submitted a proposal to the Rockefeller Foundation asking for funding for the summer workshop. The proposed study was, they wrote, based on "the conjecture that every aspect of learning or any other feature of intelligence can be in principle so precisely described that a machine can be made to simulate it."[4] The proposal listed a set of topics to be discussed—natural language processing, neural networks, machine learning, abstract concepts and reasoning, creativity—that have continued to define the field to the present day. Even though the most advanced computers in 1956 were about a million times slower than today's smart phones, McCarthy and colleagues were optimistic that AI was in close reach: "We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer."[5]

Problems soon arose that would be familiar to any scientific workshop organizer today. The Rockefeller Foundation came through with only half the requested amount of funding. And it turned out to be harder than McCarthy had thought to convince the participants to actually come, and then stay, not to mention agree on anything. There were lots of interesting discussions but not a lot of coherence. As usual in such meetings, "Everyone had a different idea, a hearty ego, and much enthusiasm for their own plan."[6] However, the Dartmouth summer of AI did produce a few very important outcomes. The field itself was named, and its general goals were outlined. The soon-to-be "big four" pioneers of the field—McCarthy, Minsky, Allen Newell, and Herbert Simon—met and did some planning for the future. And for whatever reason, these four came out of the meeting with tremendous optimism for the field. In the early 1960s, McCarthy founded the Stanford Artificial Intelligence Project, with the "goal of building a fully intelligent machine in a decade."[7] Around the same time, future Nobel laureate Herbert Simon predicted, "machines will be capable, within twenty years, of doing any work that a man can do."[8] Soon after, Marvin Minsky, founder of the MIT AI Lab, forecasted that, "within a generation...the problems of creating 'artificial intelligence' will be substantially solved."[9]

Definitions, and Getting On With It

None of these predicted events has yet come to pass. How far do we remain from the goal of building a "fully intelligent machine"? Would such a machine require us to reverse-engineer the human brain in all its complexity, or is there a shortcut, a clever set of yet unknown algorithms, that can produce what we recognize as full intelligence? What does "full intelligence" even mean?

"Define your terms...or we shall never understand one another."[10] This admonition from the eighteenth-century philosopher Voltaire is a challenge for anyone talking about artificial intelligence, since its central notion—intelligence—remains so ill-defined. Marvin Minsky himself coined the phrase "suitcase word"[11] for terms like *intelligence* and its many cousins, such as *thinking*, *cognition*, *consciousness*, and *emotion*. Each is packed like a suitcase with a jumble of different meanings. Artificial intelligence inherits this packing problem, sporting different meanings in different contexts.

Most people would agree that humans are intelligent and specks of dust are not. Likewise, we generally believe that humans are more intelligent than worms. As for human intelligence, IQ is measured on a single scale, but we also talk about the different dimensions of intelligence: emotional, verbal, spatial, logical, artistic, social, and so forth. Thus, intelligence can be binary (something is or is not intelligent), on a continuum (one thing is more intelligent than another thing), or multidimensional (someone can have high verbal intelligence but low emotional intelligence). Indeed, the word "intelligence" is an over-packed suitcase, zipper on the verge of breaking.

For better or worse, the field of AI has largely ignored these various distinctions. Instead, it has focused on two efforts: one scientific and one practical. On the scientific side, AI researchers are investigating the mechanisms of "natural" (i.e., biological) intelligence by trying to embed it in computers. On the practical side, AI proponents simply want to create computer programs that perform tasks as well or better than humans, without worrying

about whether these programs are actually *thinking* in the way humans think. When asked which motivation drives them, I've heard many AI people joke that it depends on where their funding currently comes from.

In a recent report on the current state of AI, a committee of prominent researchers defined the field as "a branch of computer science that studies the properties of intelligence by synthesizing intelligence."[12] A bit circular, yes. But the same committee also admitted that it's hard to define the field, and that may be a good thing: "The lack of a precise, universally accepted definition of AI probably has helped the field to grow, blossom, and advance at an ever-accelerating pace."[13] Furthermore, the committee notes, "Practitioners, researchers, and developers of AI are instead guided by a rough sense of direction and an imperative to 'get on with it.' "

An Anarchy of Methods

At the 1956 Dartmouth workshop, different participants espoused divergent opinions about the correct approach to take to develop AI. Some people—generally mathematicians—promoted mathematical logic and deductive reasoning as the language of rational thought. Others championed inductive, statistics-based methods in which programs learn from data or experience and use probabilities to deal with uncertainty. Still others believed firmly in taking inspiration from biology and psychology to create brain-like programs. What you may find surprising is that the arguments among proponents of these various approaches persist to this day. And each approach has generated its own panoply of principles and techniques, fortified by specialty conferences and journals, with little communication among the subspecialties. A recent AI survey paper summed it up: "Because we don't deeply understand intelligence or know how to produce general AI, rather than cutting off any avenues of exploration, to truly make progress we should embrace AI's 'anarchy of methods.' "[14]

Since the 2010s, one family of AI methods—collectively called *deep learning* (or *deep neural networks*)—has risen above the anarchy to become the dominant AI paradigm. In fact, in much of the popular media, the term "artificial intelligence" itself has come to mean *deep learning*. This is an unfortunate inaccuracy, and I need to clarify the distinction. AI is a field that includes a broad set of approaches, with the goal of creating machines

with intelligence. Deep learning is only one such approach. Deep learning is itself one method among many in the field of *machine learning*, a subfield of AI in which machines "learn" from data or from their own "experiences". To better understand these various distinctions, it's important to understand a philosophical split that occurred early on in the AI research community: the split between so-called *symbolic* and *subsymbolic* AI.

Symbolic AI

First let's look at *symbolic AI*. A symbolic AI program's knowledge consists of words or phrases (the "symbols") understandable to a human, along with rules by which the program can combine and process these symbols in order to perform its assigned task.

I'll give you an example. One early AI program was confidently called the *General Problem Solver,*[15] or "GPS" for short. (Sorry about the confusing acronym; the *General Problem Solver* predated the Global Positioning System.) GPS could solve problems such as the "Missionaries and Cannibals" puzzle, which you may have tackled yourself as a child. In this well-known conundrum, three missionaries and three cannibals all need to cross a river, but their boat holds only two people. If at any time the (hungry) cannibals outnumber the (tasty-looking) missionaries on one side of the river . . . well, you probably know what happens. How do all six get across the river intact?

The creators of the General Problem Solver, cognitive scientists Herbert Simon and Allen Newell, had recorded several students "thinking out loud" while solving this and other logic puzzles. Simon and Newell then designed their program to mimic what they believed were the students' thought processes.

I won't go into the details of how GPS worked, but its symbolic nature can be seen by the way its knowledge was encoded. To set up the problem, a human would write code for GPS that looked something like this:

CURRENT STATE:

LEFT-BANK = [3 MISSIONARIES, 3 CANNIBALS, 1 BOAT]

RIGHT-BANK = [EMPTY]

DESIRED STATE:

LEFT-BANK = [EMPTY]

RIGHT-BANK = [3 MISSIONARIES, 3 CANNIBALS, 1 BOAT]

In English, these lines represent the fact that initially the left bank of the river "contains" three missionaries, three cannibals, and one boat, whereas the right bank doesn't contain any of these. The desired state represents the goal of the program—get everyone to the right bank of the river.

At each step in its procedure, GPS attempts to change its current state to make it more similar to the desired state. In its code, the program has "operators" (in the form of sub-programs) that can transform the current state into a new state, and "rules" that encode the constraints of the task. For example, there is an operator that moves some number of missionaries and cannibals from one side of the river to the other:

MOVE (#MISSIONARIES, #CANNIBALS, FROM-SIDE, TO-SIDE)

The words inside the parentheses are called *arguments,* and when the program runs, it replaces these words by numbers or other words. That is, #MISSIONARIES is replaced with the number of missionaries to move, #CANNIBALS with the number of cannibals to move, and FROM-SIDE and TO-SIDE are replaced with "LEFT-BANK" or "RIGHT-BANK", depending on which riverbank the missionaries and cannibals are to be moved from. Encoded into the program is the knowledge that the boat is moved along with the missionaries and cannibals.

Before being able to apply this operator with specific values replacing the arguments, the program must check its encoded rules; for example, the maximum number of people that can move at a time is 2, and the operator cannot be used if it will result in cannibals outnumbering missionaries on a riverbank.

While these symbols represent human-interpretable concepts such as missionaries, cannibals, boat, left bank, etc., the computer running this program of course has no knowledge of the meaning of these symbols. You could replace all occurrences of "MISSIONARIES" with "Z372B" or any other nonsense string, and the program would work in exactly the same way. This is part of what the term "General" refers to in "General Problem Solver". To the computer, the "meaning" of the symbols derives from the ways in which they can be combined, related to one another, and operated on.

Symbolic AI of the kind illustrated by GPS ended up dominating the field for its first three decades, most notably in the form of *expert systems*, in which human experts devised rules for computer programs to use in tasks such as medical diagnosis and legal decision-making. There are several active branches of AI that still employ symbolic AI—I'll describe examples of it later on, particularly in discussions of AI approaches to reasoning and "common sense".

Subsymbolic AI: Perceptrons

Symbolic AI was originally inspired by mathematical logic as well as by the way people described their conscious thought processes. In contrast, *subsymbolic* approaches to AI took inspiration from neuroscience and sought to capture the sometimes *unconscious* thought processes underlying what some have called "fast perception", such as recognizing faces or identifying spoken words. Subsymbolic AI programs do not contain the kind of human-understandable language we saw in the Missionaries and Cannibals example above. Instead, a subsymbolic program is essentially a stack of equations—a thicket of often hard-to-interpret operations on numbers. In most cases, such systems are designed to learn from data how to perform a task.

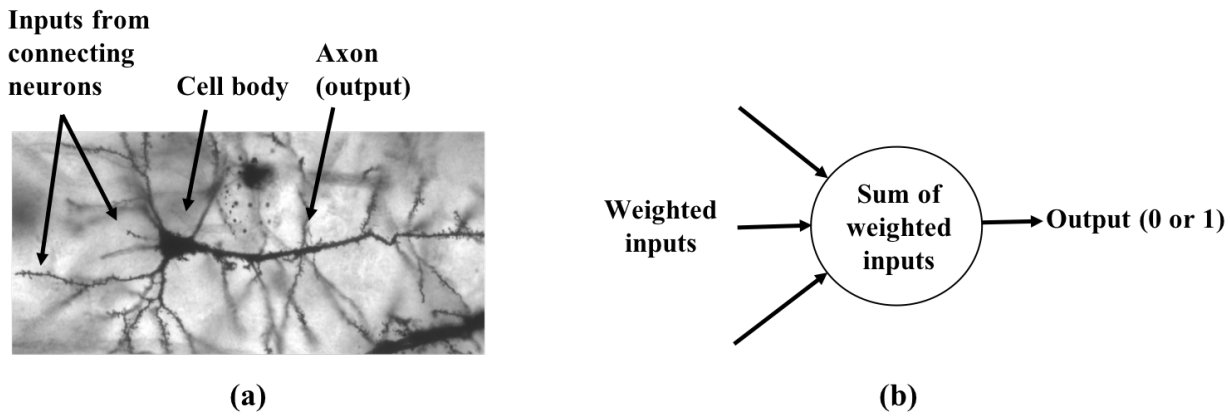

Figure 1: (a) A neuron in the brain.[16] (b) A simple perceptron.

An early example of a subsymbolic, brain-inspired AI program was the *perceptron*, invented in the late 1950s by the psychologist Frank Rosenblatt.[17] The term "perceptron" may sound a bit 1950s science-fiction-y to our modern ears (as we'll see, it was soon followed by the "cognitron" and the "neocognitron") but the perceptron was an important milestone in AI and the influential great-grandparent of modern AI's most successful tool, deep neural networks.

Rosenblatt's invention of perceptrons was inspired by the way in which neurons process information. A neuron is a cell in the brain that receives electrical or chemical input from other neurons that connect to it. Roughly speaking,
a neuron sums up all its inputs, and if the total sum reaches a certain threshold level, the neuron fires. Importantly, different connections (*synapses*) to a neuron have different strengths; inputs from stronger connections are weighted more strongly than input from weaker connections. Neuroscientists believe that adjustments to the strength of connections between neurons is a key part of how learning takes place in the brain.

To a computer scientist (or, in Rosenblatt's case, a psychologist), information processing in neurons can be simulated by a computer program—a perceptron—that has multiple numerical inputs and one output. The analogy between a neuron and a perceptron is illustrated in Figure 1. Part (a) shows a neuron, with its inputs, cell

Figure 2: Examples of handwritten digits.[18]

body, and axon (i.e., output channel) labeled. Part (b) shows a simple perceptron. Analogous to the neuron, the perceptron adds up its inputs, and if the resulting sum is equal to or greater than the perceptron's *threshold*, the perceptron outputs the value **1** (it "fires"); otherwise it outputs the value **0** (it "does not fire"). To simulate the different strengths of connections to a neuron, Rosenblatt proposed that a numerical *weight* be assigned to each of a perceptron's inputs; each input is multiplied by its weight before being added to the sum. A perceptron's *threshold* is simply a number set by the programmer (or, as we'll see, learned by the perceptron itself).

In short, a perceptron is a simple program that makes a yes-or-no (1 or 0) decision based on whether the sum of its weighted inputs meets a threshold value. You probably make some decisions like this in your life—for example you might get input from several friends on how much they liked a particular movie, but you trust some of those friends' taste in movies more than others. If the total amount of "friend enthusiasm"—giving more weight to your more trusted friends—is high enough (i.e., greater than some unconscious threshold), you decide to go to the movie. This is how a perceptron would decide about movies, if only it had any friends.

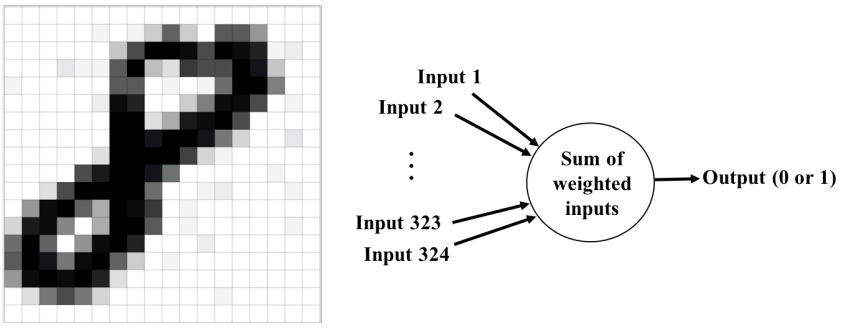

Figure 3: An illustration of a perception that recognizes handwritten '8's.[19] Each pixel in the 18×18-pixel image corresponds to an input to the perceptron, yielding 324 (= 18 × 18) inputs.

Inspired by networks of neurons in the brain, Rosenblatt proposed that networks of perceptrons could perform visual tasks such as recognizing faces and objects. To get a flavor of how that might work, let's explore how a perceptron might be used for a particular visual task: recognizing handwritten digits like those in Figure 2.

In particular, let's design a perceptron to be an '8' detector—that is, to output a **1** if its inputs are from an image depicting an '8', and to output a **0** if the image depicts some other digit. Designing such a detector requires us to (1) figure out how to turn an image into a set of numerical inputs, and (2) determine numbers to use for the perceptron's weights and threshold, so that it will give the correct output (1 for '8's, 0 for other digits). I'll go into some detail here since many of the same ideas will arise later on in my discussions of neural networks and their applications in computer vision.

Our Perceptron's Inputs

The left side of Figure 3 shows an enlarged handwritten '8'. Each grid square is a pixel with a numerical "intensity" value: white squares have an intensity of zero, black squares have an intensity of 1, and gray squares are in between. Let's assume that the images we give to our perceptron have been adjusted to be the same size as this one: 18×18 pixels. The right side of Figure 3 illustrates a perceptron for recognizing '8's. This perceptron has 324 (i.e., 18×18) inputs, each of which corresponds to one of the pixels in the 18×18 grid. Given an image

like the one in Figure 3 (left), each of the perceptron's inputs is set to the corresponding pixel's intensity. Each of the inputs would have its own weight value (not shown in the figure).

Learning the Perceptron's Weights and Threshold

Unlike the symbolic "General Problem Solver" system that I described above, a perceptron doesn't have any explicit rules for performing its task; all of its "knowledge" is encoded in the numbers making up its weights and threshold. In his various papers, Rosenblatt showed that, given the correct weight and threshold values, a perceptron like the one in Figure 3 can do fairly well on perceptual tasks such as recognizing simple handwritten digits. But how, exactly, can we determine the correct weights and threshold for a given task? Again, Rosenblatt proposed a brain-inspired answer: the perceptron should *learn* these values on its own. And how is it supposed to learn the correct values? Like the behavioral psychology theories popular at the time, Rosenblatt's idea was that perceptrons should learn via *conditioning.* Inspired in part by the behaviorist psychologist B. F. Skinner, who trained rats and pigeons to perform tasks by giving them positive and negative reinforcement, Rosenblatt's idea was that the perceptron should similarly be *trained* on examples: it should be rewarded when it fires correctly and punished when it errs. This form of conditioning is now known in AI as *supervised learning.* During training, the learning system is given an example, it produces an output, and it is then given a "supervision signal", which tells it if its output was correct or not. The system then uses this signal to adjust its weights and thresholds.

The concept of supervised learning is a key part of modern AI, so it's worth discussing in more detail. Supervised learning typically requires a large set of positive examples (e.g., a collection of '8's, written by different people, as in Figure 4a) and negative examples (e.g., a collection of other handwritten digits, as in Figure 4b). Some of the positive and negative examples are used to *train* the system—these are called the *training set*.

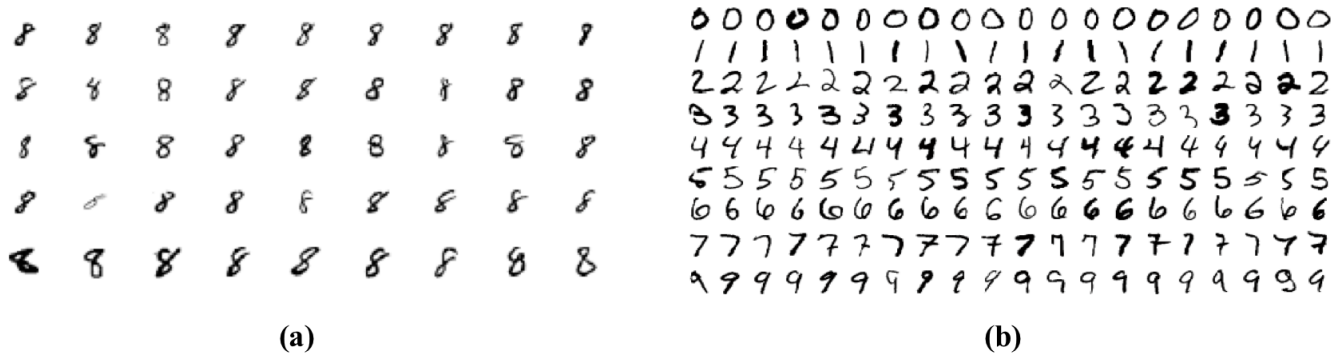

Figure 4: (a) Positive examples of digit '8'. (b) Negative examples.

The remainder—the *test set*—are used to evaluate the system's performance after it has been trained, to see how well it has learned to answer correctly in general, not just on the training examples.

Perhaps the most important term in computer science is *algorithm,* which refers to a "recipe" of steps a computer can take in order to solve a particular problem. Frank Rosenblatt's primary contribution to AI was his design of a specific algorithm, called the *perceptron-learning algorithm*, by which a perceptron could be trained from examples to determine the weights and threshold that would produce correct answers. Here's how it works. Initially, the weights and threshold are set to random values between −1 and 1. In our example, the weight on the first input might be set to 0.2, the weight on the second input set to −0.6, and so on, and the threshold set to 0.7. A computer program called a *random-number generator* can easily generate these initial values.

Now we can start the training process. The first training example is given to the perceptron; the perceptron multiplies each input by its weight, sums up all the results, compares the sum with the threshold, and either fires or not. At this point, the training process looks at the correct answer (e.g., was the input an '8'?). If the perceptron was correct, the weights and threshold don't change. But if the perceptron was wrong, the weights and threshold are changed a little bit, making the perceptron's sum on this training example closer to producing the right answer. Moreover, the amount each weight is changed depends on its associated input value; that is, the blame for the error is meted out depending on which inputs had the most impact. For example, in the '8' of

Figure 3, the higher intensity (here, black) pixels would have the most impact, and the pixels with zero intensity (here, white) would have no impact. (For interested readers, I have included some mathematical details in the notes.[20])

The whole process is repeated for the next training example. The training process goes through all the training examples multiple times, modifying the weights and threshold a little bit each time the perceptron makes an error. Just as the psychologist B. F. Skinner found when training pigeons, it's better to learn gradually over many trials; if the weights and threshold are changed too much on any one trial, then the system might end up learning the wrong thing (such as an over-generalization that "the bottom and top halves of an '8' are always equal in size"). After many repetitions on each training example, the system eventually (we hope) settles on a set of weights and a threshold that results in correct answers for all the training examples. At that point, we can evaluate the perceptron on the test examples to see how it performs on images it hasn't been trained on.

An '8' detector is useful if you care only about '8's. But what about recognizing other digits? It's fairly straightforward to extend our perceptron to have ten outputs, one for each digit. Given an example handwritten digit, the output corresponding to that digit should be **1**, and all the other outputs should be **0**. This extended perceptron can learn all of its weights and thresholds using the perceptron-learning algorithm; the system just needs enough examples of handwritten digits from which to learn.

Rosenblatt and others showed that networks of perceptrons could learn to perform relatively simple perceptual tasks; moreover, Rosenblatt proved mathematically that for a certain, albeit very limited, class of tasks, perceptrons with sufficient training could, in principle, learn to perform these tasks without error. What wasn't clear was how well perceptrons could perform on more general AI tasks. This uncertainty didn't seem to stop Rosenblatt and his funders at the Office of Naval Research from making ridiculously optimistic predictions about their algorithm. Reporting on a press conference Rosenblatt held in July of 1958, the *New York Times* featured this recap:

> The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself, and be conscious of its existence. Later perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech and writing in another language, it was predicted.[21]

Yes, even at its beginning, AI suffered from a hype problem. I'll talk more about the unhappy results of such hype shortly. But for now, I want to use perceptrons to highlight a major difference between symbolic and subsymbolic approaches to AI.

The fact that a perceptron's "knowledge" consists in a set of numbers—namely, the weights and thresholds it has learned—means that it is hard to uncover the rules the perception is using in performing its recognition task. The perceptron's rules are not symbolic; unlike the General Problem Solver's symbols, such as LEFT-BANK, #MISSIONARIES, or MOVE, a perceptron's weights and threshold don't stand for particular concepts. It's not easy to translate these numbers into rules that are understandable by humans. The situation gets much worse with modern neural networks that have millions of weights.

One might make a rough analogy between perceptrons and the human brain. If I could open up your head and watch some subset of your 100-billion neurons firing, I would likely not get any insight into what you were thinking or the "rules" you used to make a particular decision. However, the human brain has given rise to language, which allows you to use symbols (words and phrases) to tell me—often imperfectly—what your thoughts are about or why you did a certain thing. In this sense, our neural firings can be considered to be *subsymbolic*, in that they underlie the symbols our brains somehow create. Perceptrons, as well as more complicated networks of simulated neurons, have been dubbed as "subsymbolic" in analogy to the brain. Their advocates believe that to achieve artificial intelligence, language-like symbols and the rules that govern symbol

processing cannot be programmed directly, as was done in the General Problem Solver, but must emerge from neural-like architectures similar to the way that intelligent symbol-processing emerges from the brain.

The Limitations of Perceptrons

After the 1956 Dartmouth meeting, the symbolic camp dominated the AI landscape. In the early 1960s, while Rosenblatt was working avidly on the perceptron, the big four "founders" of AI, all strong devotees of the symbolic camp, had created influential—and well-funded—AI laboratories: Marvin Minsky at MIT, John McCarthy at Stanford, and Herbert Simon and Allen Newell at Carnegie-Mellon. (Remarkably, these three universities remain to this day among the most prestigious places to study AI.) Minsky, in particular, felt that Rosenblatt's brain-inspired approach to AI was a dead end, and moreover was stealing away research dollars from more worthy symbolic AI efforts.[22] In 1969 Minsky and his MIT colleague Seymour Papert published a book, *Perceptrons*,[23] in which they gave a mathematical proof showing that the types of problems a perceptron could solve *perfectly* was very limited, and that the perceptron-learning algorithm would not do well in scaling up to tasks requiring a large number of weights and thresholds.

Minsky and Papert pointed out that if a perceptron is augmented by adding an additional "layer" of simulated neurons, the types of problems that the device can solve is, in principle, much broader.[24] A perceptron with such an added layer is called a *multilayer neural network.* Such networks form the foundations of much of modern AI; I'll describe them in detail in the next chapter. But for now, I'll note that at the time of Minsky and Papert's book, multilayer networks were not broadly studied, largely because there was no general algorithm, analogous to the perceptron-learning algorithm, for learning their weights and thresholds.

The limitations Minsky and Papert proved for simple perceptrons were already known to people working in this area when Minsky and Papert's book came out;[25] Frank Rosenblatt himself had done extensive work on multilayer perceptrons and recognized the difficulty of training them.[26] It wasn't Minsky and Papert's

mathematics that put the final nail in the perceptron's coffin; rather it was their speculation on multilayer neural networks:

> [The perceptron] has many features to attract attention: its linearity; its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation. There is no reason to suppose that any of these virtues carry over to the many-layered version. Nevertheless, we consider it to be an important research problem to elucidate (or reject) our intuitive judgment that the extension is sterile.[27]

Ouch. In today's vernacular that final sentence might be termed "passive-aggressive." Such negative speculations were at least part of the reason that funding for neural-network research dried up in the late 1960s, at the same time that symbolic AI was flush with government dollars. In 1971, at the age of 43, Frank Rosenblatt died in a boating accident. Without its most prominent proponent, and without much government funding, research on perceptrons and other subsymbolic AI methods largely halted, except in a few isolated academic groups.

AI Winter

In the meantime, proponents of symbolic AI were writing grant proposals promising impending breakthroughs in areas such as speech and language understanding, commonsense reasoning, robot navigation, and autonomous vehicles. By the mid-1970s, while some very narrowly focused expert systems were successfully deployed, the more general AI breakthroughs that had been promised had not materialized.

The funding agencies noticed. Two reports, solicited respectively by the British Science Research Council in the UK and the Defense Advanced Research Projects Agency (DARPA) in the US, reported very negatively on progress and prospects for AI research. The UK report in particular acknowledged that there was promise in the

area of specialized expert systems—"programs written to perform in highly specialised problem domains, when the programming takes very full account of the results of human experience and human intelligence within the relevant domain"[28]—but concluded that the results to date were "wholly discouraging about general-purpose programs seeking to mimic the problem-solving aspects of human [brain] activity over a rather wide field. Such a general-purpose program, the coveted long-term goal of AI activity, seems as remote as ever."[29] This report led to the end of government funding for AI research in the UK; similarly, DARPA drastically cut funding for basic AI research in the US.

This was an early example of a repeating cycle of bubbles and crashes in AI. The two-part cycle goes like this. Phase 1: New ideas create a lot of optimism in the research community. Results of imminent AI breakthroughs are promised, and often hyped in the news media. Money pours in from government funders and venture capitalists for both academic research and commercial startups. Phase 2: The promised breakthroughs don't occur, or are much less impressive than promised. Government funding and venture capital dries up. Startup companies fold and AI research slows. This pattern became familiar to the AI community: "AI Spring", followed by over-promising and media hype, followed by "AI Winter". This has happened to various degrees in cycles of five-to-ten years. When I got out of graduate school in 1990, the field was in one of its winters, and had garnered such a bad image that I was even advised to avoid the term "artificial intelligence" on my job applications, since the field had garnered a bad image.

Easy Things are Hard

The cold AI winters taught practitioners some important lessons. The simplest lesson was noted by John McCarthy, 50 years after the Dartmouth conference: "AI was harder than we thought."[30] Marvin Minsky pointed out that in fact AI research had uncovered a paradox: "Easy things are hard."[31] The original goals of AI—computers that could converse with us in natural language, describe what they saw through their camera eyes, learn new concepts after seeing only a few examples—these are things that young children can easily do, but, surprisingly, these "easy things" have turned out to be harder for AI to achieve than diagnosing complex

diseases, beating human champions at chess and Go, and solving complex algebraic problems. As Minsky went on, "In general, we're least aware of what our minds do best."[32] The attempt to create artificial intelligence has, at the very least, helped elucidate how complex and subtle are our own minds.

---

[1] McCarthy J. et al. (1955). A proposal for the Dartmouth summer research project in artificial intelligence. Submitted to the Rockefeller Foundation. Reprinted in *AI Magazine*, 27 (4), 2006, 12–14.

[2] Cybernetics was an interdisciplinary field that studied "control and communication in the animal and in machines". See Wiener, N. (1961). *Cybernetics.* MIT Press.

[3] Quoted in Nilsson, N. J. (2012). *John McCarthy: A Biographical Memoir*. National Academy of Sciences.

[4] McCarthy, J. et al. (1955). A proposal for the Dartmouth summer research project in artificial intelligence. Submitted to the Rockefeller Foundation. Reprinted in *AI Magazine*, 27 (4), 2006, 12–14.

[5] Ibid.

[6] Solomonoff, G., "Ray Solomonoff and the Dartmouth summer research project in artificial intelligence, 1956." http://www.raysolomonoff.com/dartmouth/dartray.pdf

[7] Moravic, H. (1988). *Mind Children: The Future of Robot and Human Intelligence.* Harvard University Press, p. 20.

[8] Simon, H. A. (1965). *The Shape of Automation for Men and Management*. Harper & Row, p. 96. Note that Simon's use of "man" rather than "person" was par for the course in 1960s America.

[9] Minsky, M. (1967). *Computation: Finite and Infinite Machines*. Prentice-Hall, p. 2.

[10] Redman, B. R. (1977). *The Portable Voltaire*. Penguin Books, p. 225.

[11] Minsky, M. (2006). *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. Simon & Schuster, p. 95.

[12] One Hundred Year Study on Artificial Intelligence (AI100), 2016 Report, https://ai100.stanford.edu/2016-report, p. 13.

[13] Ibid. p. 12.

[14] Lehman, J. et al. (2014). An anarchy of methods: Current trends in how intelligence is abstracted in AI. *IEEE Intelligent Systems*, 29 (6), 56–62.

[15] Newell, A., and Simon, H. A. (1961). GPS, a program that simulates human thought. P-2257. Rand Corporation, Santa Monica, CA.

[16] Photo of neuron by MethoxyRoxy (https://creativecommons.org/licenses/by-sa/2.5), https://upload.wikimedia.org/wikipedia/commons/d/dc/Pyramidal_hippocampal_neuron_40x.jpg

[17] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review,* 65 (6), 386–408.

[18] From the MNIST database of handwritten digits, http://yann.lecun.com/exdb/mnist/.

[19] '8' image from https://medium.com/@ageitgey/machine-learning-is-fun-part-3-deep-learning-and-convolutional-neural-networks-f4035931872

[20] Mathematically, the perceptron-learning algorithm is the following. For each weight w_j: $w_j \leftarrow w_j + \eta\,(t - y)\,x_j$, where t is the correct output (1 or 0) for the given input, y is the actual output of the perceptron, x_j is the input associated with weight w_j, and η is the *learning rate*, a value given by the programmer. The arrow signifies an update. The threshold is incorporated by creating an additional "input" x_0 with a constant value of 1, whose associated weight $w_0 = -threshold$. With this added input and weight (called the *bias*), the perceptron fires only if the sum of the inputs times weights (i.e., the dot product between the input vector and the weight vector) is greater than or equal to zero. Often the input values are scaled and other transformations are applied in order to keep the weights from growing too large.

[21] Quoted in Olazaran, M. (1996). A sociological study of the official history of the perceptrons controversy. *Social Studies of Science*, 26 (3), 611–659.

[22] Bowden, M. A. (2006). *Mind as Machine: A History of Cognitive Science,* Volume 2, Clarendon Press, p. 913.

[23] Minksy, M. L. and Papert, S. L. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press.

[24] In technical terms, any Boolean function can be computed by a fully connected multilayer network with linear threshold units and one middle ("hidden") layer.

[25] Olazaran, M. (1996). A sociological study of the official history of the perceptrons controversy. *Social Studies of Science*, 26 (3), 611–659.

[26] Nagy, G. (1991). Neural networks—then and now. *IEEE Transactions on Neural Networks,* 2 (2), 316–318.

[27] Minksy, M. L. and Papert, S. L. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press, p. 231-232.

[28] Lighthill, J. (1973). Artificial intelligence: A general survey. In *Artificial Intelligence: A Paper Symposium.* London: Science Research Council.

[29] Ibid.

[30] Quoted in C. Moewes and A. Nürnberger (2013). *Computational Intelligence in Intelligent Data Analysis*. Springer, p. 135.

[31] Minsky, M. L. (1987). *The Society of Mind.* Simon & Schuster, p. 29.

[32] Ibid.