

Chapter 1

Brain Computation

Modern science has come to the point where the best way to understand the brain is as doing computation. We are far from and probably will never be able to predict individual acts such as Michaelangelo carving the Pieta - although we would have more to say about Romeo wooing Juliet - but for the best way to understand why we think the way we do and why we feel the way we do, we turn to computation. Its not that we will ever be able to predict exactly what we'll do in any situation, but we will be able to predict the kinds of things we are likely to do with an increasing fidelity.

Computational descriptions of humans make us uneasy in a way that other disciplines are spared. We don't rail against physicists or chemists for their models of us. The main reason that computation is singled out is that it is associated with mindless robotic behavior, the very antithesis of the rich tapestries of affect and contemplation associated with being human. However, this connection is a misinterpretation. Its true that conventional robots can be driven by computers, but computation has much more deeper things to say, in particular some stunning things to say, about how brains work. It may well be that ultimately the best way to understand our collection of human traits, such as language, altruism, emotions and consciousness, is via an understanding of their computational leverage. Don't worry, this understanding won't preclude us celebrating all our humanness in the usual ways. We will still fall in love and read Shakespeare with this knowledge just as knowing a chair is made up of atoms doesn't preclude the joy of a comfortable sit down. It would be good if we could use this knowledge to temper our darker excesses though.

The job of understanding the brain has been characterized as one of

reverse engineering.⁷ We have brains galore sitting in front of us; we just have to figure out what makes them work. This in turn involves on figuring out what the parts are and how they cooperate. What makes the task daunting is that the brain exhibits complexity that is nothing short of staggering. Myriads of brain cells that act in ways unlike any other cells in the body form tangled webs of interconnections and each of these cells is in itself a small factory of thousands of proteins that orchestrate a complex set of internal functions. Faced with all these intricate interactions, the reverse engineer tries to break the overall problem into manageable pieces.

Most of the time the researchers doing this job would use computer models routinely and not spend much time wondering what the alternative to computation could be. The fact is that so far we do not have any good alternatives to thinking about the brain other than as doing computation. Mathematics and especially physics have at times competed with computation with rival explanations of brain function, so far without result. The parts of mathematics and physics that are accessible and useful turn out to be the parts that readily fit onto computers. Chemistry and especially biology and molecular biology have emerged as essential descriptive constructs, but when we try to characterize the aspects of these disciplines that are important, more and more often, we turn to computational constructs. The main factor is that the primitives we need to describe brain function that involve the direction of physical and mental behaviors, seem naturally to fit onto the notions of iteration, decision-making and memory that are found in any standard computing language.

If the brain is a computer it is almost certainly unlike any one we've seen before and so even the computer expert has to be open to very usual and radical - but still computational - ways of getting things done. For example, the brain has to have very fast ways of coping with its very slow circuitry - a million times slower than silicon circuitry. Scientists are only just beginning to understand the ways this might be done, but enough has been learned to offer a coherent account. Thus the thrust of this book is to show how the kinds of things that we associate with thinking all have computational and neural underpinnings and the thrust of this chapter is to get started by outlining the main issues. We need to introduce the basic structure of the brain and relate it to basic ideas about programming. The foremost hurdle to understanding the brain in computational terms is the staggering slowness of its circuitry with respect to silicon. To overcome this hurdle we will sketch some of the ways in which fast results can be obtained with slow computational units.

Promises to make progress do not move skeptics. Their issue is that

although computation can be a model of the brain, and a poor one at that, it cannot truly be the ultimate description of brain function because human brains operate outside of the computational realm. To counter this negativity we will have to address the very nub of the issue and that is: What is computation? There are ways of deciding what is computation and what is not, even though, because they appeal to abstract issues involving infinities, they aren't very interesting to the reverse engineer. Nonetheless, since anti-computational arguments have been widely circulated, they need to be refuted.

1.1 Introducing the Brain

Lets get started by introducing the main organizational structure of the brain. The brain is an exquisitely complex structure that has evolved over millennia to perform very specific functions related to animal survival and procreation. It is a very complex structure that can only be understood by studying its features from many different vantage points. Lets start with the largest anatomical viewpoint. The brain has specialized components and mechanisms for input, output, short-term memory, long-term memory, and arousal. The organization and functions of these major subsystems are far from completely understood, but can be characterized in broad outline. Figure 1.1 will serve to orient you.

What Fig. 1.1 makes immediately obvious is that evolution's most recent experiment, the forebrain, has a large percentage of the total brain volume. The various parts of the forebrain will be described in the next chapter, but basically they perform in large part the basic sophisticated planning and acting that we associate with being us. However a crucial point is that they depend on the parts of the brain that evolved before them and those parts of the brain are devoted to keeping us alive and kicking. These comprise a huge network of primordial structures that are heavily optimized for a myriad of life-support functions that relate both to its internal structures and the external world. If certain parts of these of these are damaged, the consequences can be dire.

In order to think about what the forebrain does computationally, it helps to start with what it doesn't do. One of the most important of the lower brain functions is the *reflex* that connects sensory information to motor actions. Figure 1.2 shows the basic features of a reflex. Sensory information is almost directly connected to efferent¹ neurons for a fast response. Al-

¹'conducting away' as opposed to *afferent*: 'conducting towards.'

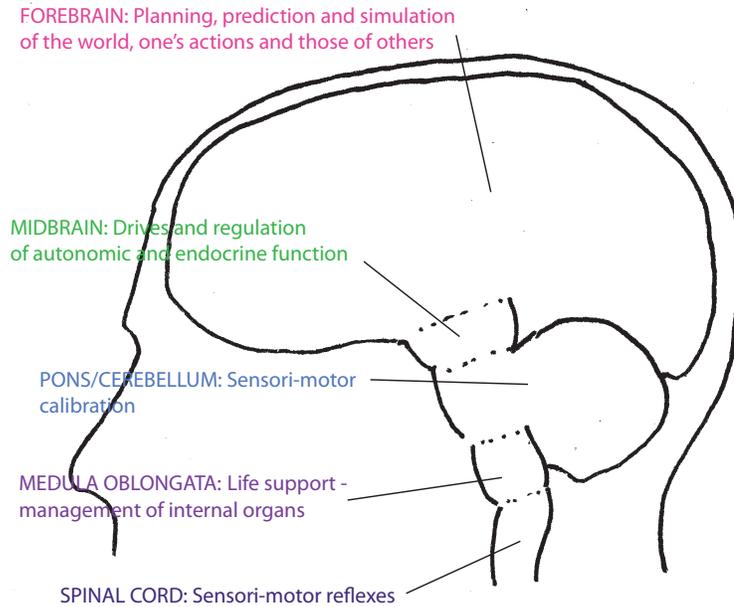


Figure 1.1: A cutaway view of the brain showing a major division between the forebrain, that handles simulation, planning and acting and the lower brain structures that handle basic life support functions.

though you might think that these reflexes are simple from experiences of withdrawing your hand from a hot plate or being whacked on the patella by a well-meaning physician, they are not. In experiments with cats without a forebrain, the basic spinal cord circuits are enough for the cat to walk, with slight assistance. Moreover the cat's feet will retract and step over small obstacles unaided. Thus the sensory motor systems of the spinal cord are in fact very sophisticated.

Reflexes can get an animal a long way. A praying mantis can snap at an unsuspecting bug as can a frog. In each case some unalterable feature of the stimulus is enough to set off the reflexive snap. Primitive robots have demonstrated a variety of behaviors can be packaged to generate useful behaviors. IRobot's Roomba vacuum clear robot has such a reflexive level of behavior.

It turns out that this optimization is an enormous boon for the forebrain, as it provides a set of sophisticated primitives that can be used as part of a programming language to realize very complex behaviors. When thinking

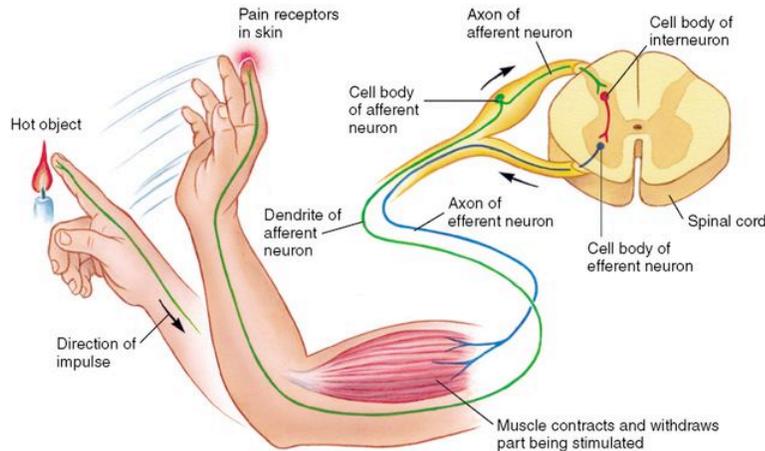


Figure 1.2: A basic sensory-motor reflex. [Permission pending]

about the virtues of human intelligence there is often a tendency to jump to its more exotic capabilities such as understanding calculus, but even the simplest ordinary behaviors are a huge advance over reflexive behavior. Suppose you are ready to make a sandwich. The plate is empty and it is time to get two slices of bread. So the next thing to do is to get them. You can effortlessly distinguish this state of affairs from the state you'd have been in if the slices were already on the plate. For either of these states, you know what to do next. Furthermore you know how to sequence through a set of actions that will produce the sandwich. You can handle mistakes - such as spilling jelly on the table - as well as unexpected difficulties - the jelly jar lid is hard to remove. And you would not repeat a step that you had already carried out. In fact in making the sandwich you have executed a complex program with many features that can be described as computation. Specifically:

- There has to be a way of defining 'state.' Picking up a piece of bread has to be programmed since its is not reflexive.
- There has to be a way of transiting between states. To make a sandwich, coordinated, sequential movements have to be made.
- There has to be a way of assembling these states and transitions into

programs. The sandwich recipe has to be remembered as a unit.

All these tasks are done by the forebrain, the subject of the next chapter. Furthermore, sandwich making is just one of literally thousands of things you know how to do. And the way most of these are encoded are as memory. As we will see in a moment the neural circuitry is very slow, so to compensate, the brain remembers how to do enormous numbers of things - basically all of the things you do - more or less exactly.

1.2 Computational Abstraction

One of the ideas that you'll have to come to grips with, and an essential tool for the reverse engineer, is models at different levels of description in the brain. You have already been introduced to the top level of abstraction, whereby the brain is divided into major anatomical and functional subdivisions. Now lets move inside the cerebral hemispheres. At the scale of a few centimeters, there is a basic organization into maps, subdivisions of cells within an anatomical unit that have identifiable roles. For example in the cerebral corticies there will be several areas responsible for the computations different aspects of of visual motion. At a level below that, divisions within these areas represent circuits of cells responsible for computations in a small area of the visual field. These circuits are made up of neurons, each of which has an elaborate set of connections to other neurons. Going one level smaller scale, the connections are defined by synapses, that regulate the charge transfer between cells.

The different anatomical levels are easy to appreciate because they have a ready physical appearance that can be indicated. What we have to get used to in addition is the idea of computation needing and using levels of abstraction. This is easiest to appreciate by seeing how computational abstraction is essential in silicon computers. At the top, any standard computer has an operating system. This the program that does the overall management of what computation happens when. It determines when jobs are sent to the printer, when input is read in and when to clean up the memory after a bunch of running programs have left it in disarray (unsurprisingly this is called "garbage collection" in the jargon.) But among all these jobs the operating system runs the program you may have written, the user program. Your program contains instructions of things to do but to the operating system these are just data, and when it determines your turn, the instructions are carried out mechanically by the computer's processor. Of course when

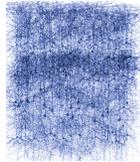
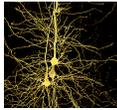
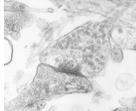
| Level | Function | Spatial Scale | Image |
|-------------|---|---------------|---|
| Sub-systems | Anatomically distinct collections of maps | 10 cm |  |
| Maps | Large-scale collections of circuits | 1 cm |  |
| Circuits | Collections of neurons organized with a specific function | 1 mm |  |
| Neuron | Basic long-range signaling unit | 10 μ |  |
| Synapse | Charge regulation in a neuron | 1 μ |  |

Table 1.1: The organization of the brain into units at different spatial scales.

you write such a program you would not have used a language that the lowest level of the machine understands, but instead you would have chosen a high-level language such as JAVA or C. The reason is that the latter instructions are too detailed, being appropriate for the minutia of the machine. The instructions you program with are translated into two lower levels, first of all assembly language, which addresses elemental operations but postpones the details of how these are done, and then finally microcode that can be understood by the machine's hardware. The point is that just to run your program, many different levels are needed, summarized in Table 1.2

Just by looking at Table 1.2 you can intuit why we are unlikely to understand the brain without a similar triage of functions into computational

levels of abstraction. We can understand the machinery that generates a voltage spike (the main signaling mechanism) in a neuron as well as how networks of neurons might use those spikes. However we cannot really understand them together. For clarity we have to keep them separate. And when we come to model altruistic behavior we have to loose track of all this machinery entirely and model and entire brain as a small number of parameters. We should not think of this technique of using abstraction levels as a disadvantage at all. Instead its a boon. By telescoping through different levels we can parcellate the brain's enormous complexity into manageable levels.

From a historical perspective computer software was designed from the bottom up. Original computer instructions were in the form of an assembly language and the more abstract user program languages were added later, as were the constructs used by modern operating systems. There is an old saw that it takes the same time to debug ten lines of code no matter which level it is written in. You can see why almost everyone writes in the highest level language possible: you get the most leverage. Nonetheless everything the computer does is carried out by the gates at its lowest level. If people had not designed the computer in the first place it would be an enormous job to figure out what it did just by looking at its outputs at the gate level, yet that daunting vista magnified is just what is facing us in understanding the brain. We have a lot of data on the function of single nerve cells and now the task is to construct the abstract levels that these cells comprise.

1.3 Different than Silicon

Our claim is that although the brain is very different than a conventional computer but it nonetheless has to deal with the same problems faced by a conventional computer. In addition the flip side is that what we hold dear as very human traits may be more understandable if we see them as representing computational solutions to problems that came up during evolution. The exposition examines what those problems are and how silicon computers deal with them to help speculate on how the brain might handle them for most of our time, but now lets pause to introduce just how shockingly different the brain must be.

The major factor separating silicon and cells is time. The switching speed of silicon transistors, which limits the speed at which a processor can go, is in the nanosecond regime. In contrast neurons send messages to each other using voltage pulses or, in the jargon, "spikes." Neurons can send these

spikes at a top speed of 1000 spikes per second, but in the main processing areas the average rate is 10 spikes per second, with 100 spikes per second regarded as a very high rate. Figure 1.3 shows some typical spikes recorded by Usrey and Reid from the cat thalamus.

For a long time it was thought that a spike was a binary pulse, but recent experiments suggest ways in which it that it could signal an analog value, so lets assume that its message is on the order of a byte per pulse. Even with this assumption, nerve cells communicate 10 million times slower than silicon transistors. You can envisage the collection of neural spikes as a code, rather like the code for a player piano. Such a piano uses special sheet music in the form of thick paper with perforations rotating over a drum with raised spring-loaded protrusions. As the drum rotates a lack of perforations depresses the protrusions causing piano keys to be struck. In an analogous way the neural spike code is a discrete code that is interpreted by the body as commands for perception and action.

Given 10^{11} nerve cells, only about 10^{10} are sending sending spikes at 10 Hz. It would be easily to compress this data by a factor of 10 so that roughly 100 seconds of your brain's neural firing could be saved in a Terabyte of storage. The unsolved task for brain scientists is to break this code.

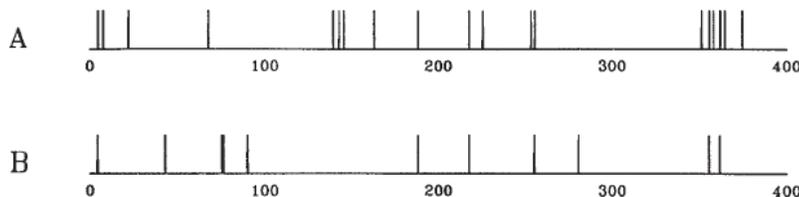


Figure 1.3: Typical spike frequencies from two neurons in the cat thalamic nucleus

The slow communication rate is sandwiched from above by the time for the fastest behavior. Evidence from behavioral experiments suggest that essential computations take about 200 to 300 milliseconds. This means that the average neuron has to get its computation done with 2 to 3 spikes. From these analyses, the consensus is that the way the brain must do it is to have most of the answers pre-computed in some tabular format so that they just have to be looked up. Evidence in favor of this view comes from the rich connectivity between nerve cells. Each neuron connects to about 10,000 other neurons, compared to connectivity between gates on a silicon

chip of just a handful. As shown in Fig 1.4, the size of the gates in silicon are comparable to the processes of a living cell. It is the neuron's huge connectivity that gives it one of its biggest advantages.

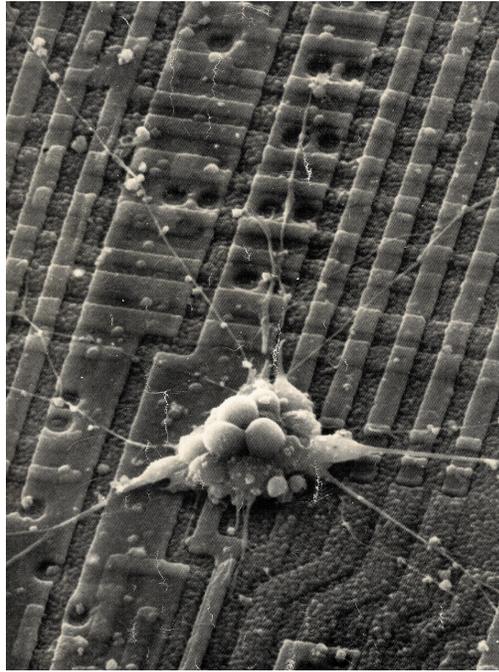


Figure 1.4: An exotic electron micrograph of a neuron artificially grown on a silicon wafer reveals the comparable scales of the two technologies. The raised clump is the neuron's body or *soma*. One of the spidery processes coming out of the soma, is its axon which connects it to an average of 10^4 other cells. In contrast silicon transistor connections between gates are limited to a handful.

Another factor that the brain uses to overcome the speed handicap is the power of nerve cells themselves. The exact computing power of a neuron is unknown, but the best guess is that it is much more powerful than a transistor. The synapses that connect it to other cells are closer to transistors, but undoubtedly more powerful. So that the neuron itself has been likened to a microprocessor albeit a specialized one. Since the brain has approximately 100 billion nerve cells - much less than the US fiscal debt in 2004 dollars, but still a lot - that can work simultaneously, the parallel computing power is obviously one source of compensation for the brain's slow circuitry.

1.4 Brain Algorithms

If the brain is doing computation at some point we have to be able to say how that computation gets done in time to direct our daily activities. In computer science this question is resolved by coming up with a specific *algorithm* or colloquially recipe that runs on the brain's processors. We still do not quite know how to specify the brain's processors in detail but we suspect that nerve cells will be at the center of the answer. We can say more about the computational algorithm. Scoring an algorithm's effectiveness is a matter of counting the number of its operations and dividing by the number of operations per unit time. But the standard ways of counting operations on silicon don't work for the brain.

Algorithms on silicon computers

On serial silicon computers most algorithms are dominated by the size of the input. For example consider sorting n numbers. Here is the recipe: Go through the list and move the biggest number to the top. Then go through the $n - 1$ remaining numbers and pick the second largest and move it to the penultimate position. After you get down to fixing the last two elements, you are done. This is called 'bubble sort' because the larger numbers bubble up to the top. This basic algorithm would take $\frac{n(n+1)}{2}$ steps since we have the sum of the numbers from 1 to n . In practice we don't sweat the factor of $\frac{1}{2}$ or the 1 and say "on the order of" or $O(n^2)$ operations²

Of course computer science majors all know that there is a faster algorithm that takes advantage of the fact that two *sorted* lists can be merged in $O(n)$ steps. Here is the algorithm that uses the merging property:

```
Sort(List)
IF List has two elements
order them
RETURN the resultant list
ELSE Merge(Sort(Front-of-List),Sort(Back-of-List))
```

You start out with the original list and contract to merge its two sorted sublists. Of course they need to be handled in the same way but the key is that they are smaller by half. For large lists, many IOUs are created

²In the jargon 'O' is called 'Big Oh' or simply 'Oh.'

in this process that are resolved when we get down to two or one element lists. Once this happens the many outstanding merging processes can be completed, resulting in a sorted list. To see the idea try it on a piece of paper with small lists of say four to six numerical elements.

A careful accounting shows that this only requires, in the notation, $O(n \log n)$ operations, which of course is better than the easier-to-understand $O(n^2)$ algorithm that was considered first.

The Brain's tricks for fast computation

The clever sorting algorithm is the best that can be done but still won't do for a model of brain computation. The main reason is that the neurons that are the candidates for the principal computing elements are very slow, over a million times slower than silicon. A reasonable size for n is 1,000,000 for human vision, and neurons are computing at 10 binary 'bits' per second, so you can see why an $O(n \log n)$ algorithm is not a candidate. An algorithm that had to serially poll each of these cells would take an impractical 100,000 seconds. Fortunately the brain has several ways of cutting corners.

Probably, Approximately Correct: The brain uses algorithms that are just good enough The $O(n \log n)$ algorithm for sorting is provably the best there is. It can sort *any* sequence in its allotted time. But this is not the case for all problems. For some algorithms, such as finding the best round-trip route through a set of cities, getting the shortest possible path is very expensive. You have to examine all of the paths, and that is exponentially many. Naturally this is prohibitively expensive for humans. But in many cases if we just want a reasonably good solution, this can be had at a reasonable cost. Thus one of the main ways of speeding things up is to use probably, approximately correct algorithms, or PAC algorithms. These were pioneered by Valiant.⁹

The brain uses a fixed size input. In the analysis for the sorting algorithms on silicon computers, the assumption is that the dominant factor is the size of the input. Where the size of the input can grow arbitrarily, this is the correct analysis. For example, suppose we pick a give cost for bubble sort so that now there is no 'Big O,' but instead we know that the cost is exactly $1000n^2$ on a given computer. Now your friend gets a computer that is 1000 times faster so that the cost is exactly n^2 . You have to use the old computer but can use the merge-sort algorithm. You beat your friend when

$$1000n \log n < n^2$$

So for example when $n = 10,000$ you'll be 2.5 times faster.

However this analysis breaks down for biological systems as the number of inputs and outputs are for all practical purposes fixed. When this happens, it pays to optimize the hardware. To pursue the example of vision, suppose now that each image measurement could be sampled in parallel. Now you do not have to pay the 1,000,000 factor. Furthermore suppose that you wanted to use this parallel 'bus' of image measurements to look something up. Now you only need $O(\log n)$ measurements. Furthermore, the brain ameliorates this cost as well by using a pipeline architecture so that the log factor is amortized quickly over a series of stages. Each stage can be thought of as answering one of twenty questions so that by the time the process exits the answer has been determined.

The brain uses sensors and effectors that are designed for the world. The design for the light sensing photoreceptors used by vision is believed to have started with the properties of sea water. It turns out that the visible spectrum is especially good at penetrating water and so could be exploited by fish. Once the hardware was discovered, it worked on land as well.

In the same way the human musculo-skeletal system is especially designed for the human ecological niche. We cannot outrun a cheetah, but we can climb a tree much better. The niche is even more sharply illustrated by current robotics. Although silicon computers can easily out-calculate humans, the design of robotic bodies is still very much inferior to that of human bodies. Furthermore the general problems that these human bodies solve seemingly effortlessly are still very much superior to their robotic counterparts except in a few special cases. To appreciate this further, try wearing gloves and going about your normal everyday activities. You will quickly find that you are frustrated in nearly every situation that requires detailed hand coordination. If this does not convince you, put on boxing gloves! You will still be better off than a parallel-jaw gripper robot arm.

Of course there are difficult situations that can overwhelm a human body, such as staring into the sun or trying to jump too broad a gap. But these situations are for Darwin Award contestants. For almost all the problems of everyday existence, the human body is an exquisite design that works wonders.

The brain amortizes the cost of computation over its lifetime One contributing factor to fine motor coordination that we have just discussed is the design of the physical system. So far no robot can even come close to the strength-to-weight capabilities of the human musculo-skeletal system. But there is another factor too and that is that the process of designing the control algorithms that work so well happens over many years.

Babies learn to control their arms sequentially. They'll lock all the out-board joints and try movements. When the nervous system has a model of this reduced system, they'll unlock the next more distal joint and try again. The new system has fewer variables than it would if starting from scratch. Naturally it is essential to have parents that patiently care-take while this process is happening, but the essential feature is that the computation is amortized over time. A wonderful analogy is *GoogleTM*. To make fast and web searches, overheated warehouses of server computers crawl the web around the clock to find and code its interesting structure. The coded results are what makes your query lightning fast. In the same way efficient motor behavior reflects a multi-year process of coding the way the body interacts with its physical surround. The result is that reaching for a cup is fast and effortless and carrying it upstairs without spilling is a snap.

1.5 Could the Brain *Not* be a Computer?

To answer this question one must first understand computation in the abstract. This is because the popular notion of computing is irretrievably tied to silicon machines. Furthermore these machines have evolved to augment human needs rather than exist on their own and as such have not been made to exhibit the kinds of values inherent in biological choices. Thus an immediate reaction to the idea that brains are kinds of computers is that, based on the limitations and peculiarities of modern silicon computers do, is to reject the idea as baseless. To counter such intuitions will take a bit of work starting with a formal characterization of computation. We can describe what a computer is abstractly so that if a brain model can be shown to be incompatible or compatible with this description, then the issue is settled. It could turn out that that brains are unreachably better than formal computation. I don't think so for a moment, but the crucial point is to frame the question correctly.

Turing Machines

What is computation? Nowadays most of us have an elementary idea of what a computer does. So much so that it can be difficult to imagine what the conceptual landscape looked like before its advent. In fact the invention or discovery of formal computation was a remarkable feat, astonishing in retrospect. Enter Alan Turing, a brilliant mathematician who led the team that broke the German enigma code in World War II. Turing's approach, that resulted in the invention of formal computation, was constructive: He

tried to systematize the mechanisms that people went through when they did try and do mathematical calculations. The result was the Turing machine, a very simple description of such calculations that defines computation. See Box TURING MACHINE. Although there have been other attempts to define computation, they have all been shown to be equivalent to Turing's definition. Thus it is the standard: if a Turing machine cannot do it, its not computation.

TURING MACHINE

All computation can be modeled on a universal machine, called a Turing machine. Such a machine has a very simple specification, as shown in Figure 1.5. The machine works by being in a "state" and reading a symbol from linear tape. For each combination of state and tape symbol, the machine has an associated instruction that specifies a triple consisting of the new state, a symbol to write on the tape, and a direction to move. Possible motions are one tape symbol to the left or right. Although the Turing machine operation appears simple, it is sufficiently powerful that if a problem can be solved by any computer it can be solved by a Turing machine.

This example shows a very simple program for erasing a block of contiguous ones. The head is moved along the tape serially, replacing each 1 with a 0. When the end symbol is encountered, the program terminates. You can define any number of auxiliary symbols to help write the program, or alternately find ways around using them. Here for example you could avoid the # symbol just by terminating after you see the second 0. For more of a challenge try to add two numbers together and then for even more of a challenge try to multiply two numbers together. Remember that they are in a unary representation, just like that used by convicts marking jail time. To get you started, think of the initial tape as containing for example:

```
0000#1111#000#111#0000#00000000
```

and the answer being

```
0000#0000#000#000#0000#1111111100
```

Your program will probably find a 'one,' replace it with a zero and then go and put it in the answer region, repeating the process until all the ones were used up. For multiplication the answer would be

```
0000#1111#000#000#0000#11111111111100
```

This program will require even more sawing back and forth on the tape.

As you try to write more complex programs you will quickly be overwhelmed by the tedium associated with the low level description used by the TM. But the point is that in principle, any program in any computer language has a TM equivalent.

when “in” a state. In terms of everyday behavior, if you are making a cup of tea and you are in a state where *{the kettle is nearby and empty}*, then presumably the next action is to *{put water in the kettle}*. The power of thinking in this way cannot be overestimated. When we describe ways of formalizing the brain’s programs in Chapter 5, it will be in terms of the *state, action* terminology.

To return to the formal computational theme, Turing machines are not without controversy because they do have limitations. In general, a TM cannot tell whether the program of another arbitrary TM will halt or keep going forever. Of course it can for some TMs but not in general. Furthermore TM calculations cannot use random numbers since the very definition of a random number is that a TM cannot decide whether it is random or not. And it cannot use real numbers either since there are infinitely many more real numbers than TMs. Experienced programmers know that when they have randomness in their programs, the program is using pseudo-random numbers, that is, numbers that behave enough like random numbers to get useful answers to programs that need them. Similarly programs use integers to sample the space of real numbers, again getting numbers close enough to the real thing for the calculations to have meaning. Finally, as we will elaborate on in a moment, a TM cannot use everyday formal logic either without some form of limitation, since if it tries to prove a theorem that is not true, there is a possibility that the program will run forever and not halt.

The question is though: are these limitations important or do we need a more powerful model such as physics? What are the prospects for a physics-based computing? Some scientists think big and one is Seth Lloyd.² He calculated the operations that would be needed to simulate the universe since its inception. The estimate is that you need no more than 10^{120} operations on 10^{90} bits for the memory. These numbers are more than the universe has, but that is because any simulation will have some overhead. Also you might not get the same universe if you started with a different random number seed; indeed if the process depended on truly random numbers, you might not get our universe at all. One important take-home message from this vantage point is that to the extent that the universe can be described in terms of computation, then presumably our small brains can too! But a larger point is that perhaps there is a prospect of harnessing quantum computing to solve difficult problems. Potentially many solutions could be coded as probabilistic quantum states in a machine that could then pick the best one. While this is intriguing the technical problems in making this work at any useful scale are enormous. For an introduction see.²

A very pessimistic view of computational prospects is represented by Roger Penrose who has written three books with the theme that TMs are too weak a model to encompass the brain.⁶ (Penrose also holds out hope for the quantum computation and proposes a way in which it might be utilized by cells that is wildly speculative.) But what of his arguments that TMs do not have sufficient power? One of Penrose's main arguments settles around Gödel's theorem. Since the theorem itself is very subtle and important, we'll take some time to introduce its elements (See Box GÖDEL'S CONSTRUCTION)

GÖDEL'S THEOREM

The end of the nineteenth century marked the zenith of an enormous wave of discoveries of the industrial revolution. Parallels in science such as Maxwell's equations abounded. In philosophy this enthusiasm was captured by the idea of Logical Positivism: things would steadily get better. This optimism was soon to be corrected by the carnage of World War I, but had a shocking parallel setback in the scientific arena as well.

One of the proponents of the wave of optimism was David Hilbert. With the formalization of mathematics as logic, it was believed that theorems were to be discovered mechanically by starting with the right axioms and turning the crank of logical inference. To abuse Fukuyama's much more recent phrase, we were looking at the "end of mathematics." Mathematical discovery was to be reduced to a procedure that seemed to obviate the need for creative discovery. Thirty years later, this optimism was shattered by Kurt Gödel. Gödel was a genius and troubled individual who later in life succumbed to mental problems, but in the 1930s he was at the height of his powers. One of the key open questions in arithmetic was: were its axioms sufficient? Gödel's answer was that this question could in fact not be resolved. In a stunning demonstration he showed that there were true theorems in arithmetic that could not be proved.

A central portion of his proof showed that logic itself could be seen as properly contained in arithmetic. Any logical statement could be reduced to a unique number. As a consequence any statement about arithmetic in logic was just a form of 'syntactic sugar' to disguise its arithmetic origin. His construction makes use of prime numbers and is easy to appreciate.

The heart of Gödel's argument consists of an unambiguous mapping between formulas in logic and numbers. The mapping construction begins by assigning the elementary symbols in predicate calculus(logic) to numbers as shown in Table 1.5.

Next for variables x, y, z we use primes after the number 12, for sentential variables (such as $p \supset q$) we use primes after 12 squared and for predicates primes after 12 to the power of 3.

This was the basis for describing logical formulas in terms of prime numbers. To give you an idea of how this is done, consider the formal statement that every

number has a successor: $\exists x(x = sy)$. We can first look up the Gödel numbers for the individual signs and they are: 41381357179. Then the Gödel number for the formula is:

$$2^4 \times 3^{13} \times 5^8 \times 7^{13} \times 11^5 \times 13^7 \times 17^{17} \times 19^9$$

Thus any formula's number in principle can be factored into its prime divisors and, by organizing it into ascending primes and their powers, the formula can be recovered. The genius of such a mapping is that it allows one to talk about statements about logic in terms of logic. The reason is that the statements can be turned into numbers via the mapping. This then gives them a legitimate expression inside the logical formulas. From this Gödel was able to prove that there were true propositions in logic that could not themselves be proved. The gist of the argument is as follows:

1. Using the mapping, one can construct a formula P that represents the statement 'The formula P is not provable using the rules of logic.'
2. One can then show that P provable iff $\neg P$ is provable. If this is true logic is inconsistent. So if logic is consistent P is unprovable.
3. Even though P is unprovable it can be shown to be true. Therefore logic is incomplete.

Thus logic was incomplete. The interested reader is referred to Newman and Nagel's classic³ for a brilliant exposition of the gist of his arguments.

Penrose argues that since human's understand this theorem that points to a fundamental weakness of logic (in proving statements about arithmetic) but computers are forced to *use* logic ergo humans think out of the logical box and are more powerful than TMs. However if we understand a proof it has to be logical. The trick is that the referents of the proof are highly symbolic sentences. Gödel's brilliant insight was that when these were about mathematics, they could be reduced to arithmetic. hence the referents of the logical statements are regularized and no special machinery is necessary. We are not saying it is easy; after all there has only been one Kurt Gödel!

One potentially confusing point is that of being able to manage concepts at different levels of abstraction. When working a a given level one has to be careful to stay within that level to have everything make conceptual sense. You can switch levels but you have to be careful to do the bookkeeping required to go back and forth.⁵ The failure to do this can lead to endless confusion⁸ and the delight in experiencing the ambiguity that comes with it.¹

To take a famous example, the philosopher Searle has lampooned computation by imagining a person in a closed room whose task is to translate between Chinese and English. The person receives instructions in the form

of: ‘when you get this english sentence, output the following chinese kanji characters.’ The point of the ‘Chinese Room’ setting is that the person can exhibit the illusion of understanding chinese without doing so in the slightest. But here the argument rests on blurring abstraction boundaries. If the person is meant to be an anthropomorphism of a neuron obviously there will be no language understanding, but if we jump abstraction levels to have a complete person, then just as obviously there would be some understanding as the person would see regularities between symbol usages. It depends on the abstraction vantage point you pick.

Similarly for the alleged mystery in understanding Gödel’s theorem. If we take the vantage point of the coded formula, the proof is such that anyone trained in logic can understand it. It only when we try to simultaneously entertain consequences of the uncoded logical formulae together with statements at a different level of abstraction that use their coded form that things get confusing.

Turing Machines and Logic

At this point there might be one last puzzling question in your mind. For one thing the standard computer is constructed with networks logic ‘gates,’ each of which implements an elementary logic function such as the logical AND of two inputs: If they are both TRUE then the output is TRUE else the output is false. In contrast we have the exotic result of Gödel that there exist true statements in arithmetic that cannot be proved. So what is the status of logic vis a vis Turing Machines? The resolution of this ambiguity is that two kinds of logic are involved. The fixed architecture of the Turing Machine can be modeled with *Propositional Logic* whereas the logic we implicitly use in every day reasoning is modeled with *Predicate Logic*. The crucial difference is that the latter has variables that can arbitrarily correspond to tokens that represent real world objects. Thus to express the familiar ‘All men are mortal,’ we need predicate logic to have a variable that can taken on the value of any particular man. We write: for all x the proposition $P(x)$ - being a man- being true implies that the proposition $Q(x)$ - being mortal - is also true. Formally

$$\forall x P(x) \supset Q(x)$$

To prove the theorem Socrates is a man therefore Socrates is mortal, we can use a standard method in elementary logic of constructing a truth table that reveals that whenever the preconditions of the theorem are true the consequent is true.

However this is not what computer theorem provers necessarily do. A standard computer method is known as *resolution* and can be introduced as follows. The first step is to covert all the logical statements to a special form called *clause form*. For the details of how to do this you will have to consult a text such as.⁴ The steps are mechanical but take up too much space to describe here. In the simple example the result is:

$$\{P(Socrates), \neg P(x) \vee Q(x), Q(Socrates)\}$$

Now to show that the third clause follows from the first two the procedure is to negate it and show that there is no way to assign the variable x such that the set of clauses are all satisfied (true). This can be done by staring with the first two. The only option here for x is $x = Socrates$, and since $P(Socrates)$ and $\neg P(Socrates)$ cannot simultaneously be true, we conclude that $Q(Socrates)$ must be true and add it to the set of clauses. Now judiciously choose that clause and $\neg Q(Socrates)$. These cannot be simultaneously true and so we can conclude the null clause denoted '*nil*.' Thus since the negation of the consequent is unsatisfiable, the consequent is and therefore the theorem is true.

This has been a precis of what is in fact a whole scientific field of endeavor termed *automatic theorem proving* that uses this technique and a host of others. However the main issue is hinted at by the simple example. To use the process a particular binding- here $x = Socrates$ - must be chosen. This is ridiculously easy in the example but in a general case there will be many variables and many, many bindings for each variable. Nonetheless they can be organized so that they can be tried in a systematic way. The upshot is that if the theorem is true, an automated theorem prover using resolution will eventually produce the *nil* clause and prove the theorem in a finite number of steps, but if its not true it will grind through possibilities forever. This kind of situation will be familiar to any programmer: is the program almost done and about to stop at any moment or is it going to grind on forever owing to some bug?

Now for the punch line on the theorem proving and TMs. Silicon computers use gates that implement *propositional logic* formulas such as AND, OR and NOT. This is different than the *first order predicate logic* used by mathematicians to prove theorems. To do the latter, a silicon computer has to simulate predicate logic using its propositional logic hardware. If humans are TMs they have to do the same, with the same formal consequences.

As an epilogue the same point can be illustrated with human deduction. Consider Fig. 1.6. In this deck, letters are on one side and numbers on the

other. Your task is to test the claim that in this deck, “every even numbered card has a consonant on the other side.” you can only turn over two cards. Which cards do you pick? If you are like most people, you’ll have to think about this a bit and might even make a mistake. You should turn over the first and forth cards. But the main points is that it seems the brain can *simulate* logical deduction, in the same way a computer can simulate predicate calculus, but it its unlikely to use logic as a primitive construct - because if it did presumably the card problem would be trivial.

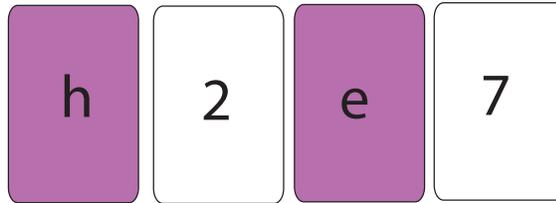


Figure 1.6: Cards have letters on one side and numbers on the other. Which two should you turn over to check whether every even numbered card has a consonant on the other side?

And just to drive another point home, see how easy this becomes if we translate this problem into a familiar experience. Four people have met at a bar, two teen agers and two older people. Two of them have left for the rest room, so your view is of a teenager and an old person and the drinks of the other two a beer and a coke. Whose identity cards should you check?

1.6 Book Overview

The idea of levels of abstraction forms the backbone of the organization of this book. To synthesize a description of all the different things that a brain does we have to stitch together many different levels of abstraction. We start in the middle in **Chapter two** by introducing the brain’s functional subsystems. Although they all are all interconnected, each of the subsystems has a specialized job to do and if they are somehow damaged, that job cannot be taken over by any of the others. Each of the subsystems is composed of nerve cells that share common structure. This commonality is taken up in **Chapter three** which discusses the structure of nerve cells and ways in which they can do computation. When we use more abstract descriptions in later chapters it is important to keep in mind that all the high

level computations ultimately have to be translated to this level. **Chapter four** focuses on the brain's major subsystem, the cortex. The nerve cells signal very more than a million times slowly than silicon transistors so that it is amazing that complex brain computations can be done at all. The answer crudely is to use experience to pre-compute all the responses that we have to make and put them into a table that we can look up. As astonishing as this is, it is the strategy used by the brain and the cortex is the table. Programs in the brain can be constructed if the entries in the table contain information about what entry to lookup next, but there has to be mechanisms for making this happen. This the the subject of **Chapter five** which describes the function of the Basal Ganglia and its use of the brain's reward system. What makes the brain's programs difficult to understand is that they are extremely compressed and one way that they can be is that they anticipate the details of the construction of the body that encases them. Thus when we make a fight or flight decision, a lot of the details of how the body reacts are encoded in the body's musculo-skeletal system in a way that needs the barest direction. This is the subject of **Chapter six**. The program's that direct the body are a collaboration of the genes and the process of development during infancy. **Chapter seven** introduces some of the ways that the brain's neuron's can use to become properly wired up. How many programs can the brain run at a time? This question is unsettled but one line of development contends that the answer is only a few. This issue is explored in **Chapter eight**. One of the hallmarks of being human is the display of emotions. It turns out that emotions serve a vital purpose for program development by helping sift through a myriad of alternatives. This role for emotions is described in **Chapter nine**. Another hallmark of humans is our big brain. How did we evolve it alone among the animal kingdom? One recent answer is that of sexual selection which introduces an instability in the genetic process. This is explained with genetic computation in **Chapter ten**. Although many animals exhibit altruistic behaviors, altruism is another signature feature of humans. **Chapter eleven** shows ways in which altruistic behavior is rational when humans have to share resources over time. Game theory provides the explanatory computational resources. Finally we take up consciousness. **Chapter twelve** describes how consciousness can be thought of as an "emotion of authorship," that distinguishes when consequences in the world are due to our own behavior as opposed to that of other causes.

1.7 Key ideas

1. Overall brain organization. Just like any complex system, the brain is composed of distinct sub-parts with specialized functions. In order to proposed good computational models it os essential to understand what these parts do.
2. Computational abstraction levels. An essential take-home lesson from complex systems is that they are inevitably hierarchically organized. This means that to understand the brain, we'll have to put together programs at several different levels of abstraction.
3. Algorithms. On the other hand if you want to show that computation is a good model then the way to go about it is to show how the things that people do can be described by recipes or programs that make use of the brain's neurons in a feasible way.
4. What is computation? Computation is described by what a Turing Machine can do. If you want to show that the brain is not a computer, the way to go about it is to demonstrate something that the brain does that cannot be reduced to a Turing Machine program.

| LEVEL | Description | Function |
|--------------|--------------------|---|
| SOFTWARE | OPERATING SYS. | Control the running of other programs; manage input output |
| | USER PROGRAM | A particular program with a specialized objective; written in a high-level language |
| | ASSEMBLY LANG. | The translation into basic machine instructions that are for the most part hardware independent |
| | MICROCODE | Machine instructions that can be interpreted by a particular machine's hardware |
| HARDWARE | CIRCUITS | Addition, Multiplication, Storing etc |
| | GATES | Basic logical operations such as AND and OR |

Table 1.2: The standard computer uses many levels of abstraction in order to manage the complexity of its computations. The hardware level also can be divided into abstraction levels consisting of circuits that are composed of basic switches or 'gates.'

| logical symbol | Gödel number | meaning |
|----------------|--------------|------------------|
| \neg | 1 | not |
| \vee | 2 | or |
| \supset | 3 | if ... then ... |
| \exists | 4 | there exists |
| $=$ | 5 | equals |
| 0 | 6 | zero |
| s | 7 | successor of |
| (| 8 | bracket |
|) | 9 | bracket |
| , | 10 | punctuation mark |
| + | 11 | plus |
| x | 12 | times |

Table 1.3: The main insight in Gödel's proof rests on the ability to uniquely map logical formulae onto numbers. This starts with the assignments shown above.

Bibliography

- [1] Hofstadter Douglas R. *Gödel Escher Bach: an Eternal Golden Braid*. Penguin, 2000.
- [2] Seth Lloyd. *Programming the Universe: A Quantum Computer Scientist Takes On the Cosmos*. Alfred A. Knopf, 2006.
- [3] Ernest Nagel and James R. Newman. *Gödel's Proof*. New York University Press, 2002.
- [4] Monty Newborn. *Automated Theorem Proving: Theory and Practice*. Springer-Verlag, 2001.
- [5] Allen Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.
- [6] Roger Penrose. *Shadows of the Mind: A Search for the Missing Science of Consciousness*. Oxford University Press, 1994.
- [7] Steven Piker. *How the Mind Works*. Norton Press, 1999.
- [8] John R. Searle. Minds, brains, and programs. *Behavioral and Brain Sciences*, 3(3):417–457, 1980.
- [9] Leslie Valiant. *Circuits of the Mind*. Oxford University Press, 1994.