Space-Time Computing with Temporal Neural Networks

Synthesis Lectures on Computer Architecture

Space-Time Computing with Temporal Neural Networks
James E. Smith
www.morganclaypool.com

ISBN: 9781627059480 print
ISBN: 9781627058902 ebook

DOI 10.2200/S00771ED1V01Y201704CAC039

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE #39
Series Editor: Margaret Martonosi, Princeton University

Series ISSN: 1935-3235 Print   1935-3243 Electronic

Space-Time Computing with Temporal Neural Networks

James E. Smith
Professor Emeritus, University of Wisconsin

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE #39

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

Understanding and implementing the brain's computational paradigm is the one true grand challenge facing computer researchers. Not only are the brain's computational capabilities far beyond those of conventional computers, its energy efficiency is truly remarkable. This book, written from the perspective of a computer designer and targeted at computer researchers, is intended to give both background and lay out a course of action for studying the brain's computational paradigm. It contains a mix of concepts and ideas drawn from computational neuroscience, combined with those of the author.

As background, relevant biological features are described in terms of their computational and communication properties. The brain's neocortex is constructed of massively interconnected neurons that compute and communicate via voltage spikes, and a strong argument can be made that precise spike timing is an essential element of the paradigm. Drawing from the biological features, a mathematics-based computational paradigm is constructed. The key feature is spiking neurons that perform communication and processing in space-time, with emphasis on time. In these paradigms, time is used as a freely available resource for both communication and computation.

Neuron models are first discussed in general, and one is chosen for detailed development. Using the model, single-neuron computation is first explored. Neuron inputs are encoded as spike patterns, and the neuron is trained to identify input pattern similarities. Individual neurons are building blocks for constructing larger ensembles, referred to as "columns". These columns are trained in an unsupervised manner and operate collectively to perform the basic cognitive function of pattern clustering. Similar input patterns are mapped to a much smaller set of similar output patterns, thereby dividing the input patterns into identifiable clusters. Larger cognitive systems are formed by combining columns into a hierarchical architecture. These higher level architectures are the subject of ongoing study, and progress to date is described in detail in later chapters. Simulation plays a major role in model development, and the simulation infrastructure developed by the author is described.

KEYWORDS

spiking neural networks, temporal models, unsupervised learning, classification, neuron models, computing theory

Contents

Figure Credits

Photo courtesy of the Archives, California Institute of Technology 3

Figure 3.1: From: www.thebrain.mcgill.ca. Used with permission 48

Figure 3.2: Reprinted from *Progress in Neurobiology, 39*, DeFelipe, J., and Fariñas, I. "The pyramidal neuron of the cerebral cortex: morphological and chemical characteristics of the synaptic inputs." 563–607, Copyright © 1992, with permission from Elsevier 49

Figure 3.3: Based on: Gerstner, W., and W. M. Kistler. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*, Cambridge University Press. Copyright © 2002 Cambridge University Press 51

Figure 3.7: (a) From: Hill, Sean L., et al., "Statistical connectivity provides a sufficient foundation for specific functional connectivity in neocortical neural microcircuits." *Proceedings of the National Academy of Sciences* 109, no. 42. Copyright © 2012 National Academy of Sciences. Used with permission. (b)From: Peters, Alan, and Claire Sethares. "Myelinated axons and the pyramidal cell modules in monkey primary visual cortex." *Journal of Comparative Neurology* 365, no. 2: 232–255. Copyright © 1996 Wiley-Liss, Inc. Used with permission 58

Figure 3.7: (d) From: Felleman, Daniel J., and David C. Van Essen. "Distributed hierarchical processing in the primate cerebral cortex." *Cerebral Cortex* 1, no. 1 (1991): 1–47., by permission of Oxford University Press 59

Figure 3.8: From: Perin, Rodrigo, Thomas K. Berger, and Henry Markram. "A synaptic organizing principle for cortical neuronal groups." *Proceedings of the National Academy of Sciences* 108, no. 13: 5419–5424. Copyright © 2011 National Academy of Sciences. Used with permission 61

Figure 3.9: From: Hill, Sean L., et al., "Statistical connectivity provides a sufficient foundation for specific functional connectivity in neocortical neural microcircuits." *Proceedings of the National Academy of Sciences* 109, no. 42. Copyright © 2012 National Academy of Sciences. Used with permission 62

Figure 3.10: From: Bakkum, Douglas J., Zenas C. Chao, and Steve M. Potter, "Long-term activity-dependent plasticity of action potential propagation delay and amplitude in cortical networks." *PLOS One* 3.5. Copyright © 2008 Bakkum et al. Used with permission 62

Preface

Understanding, and then replicating, the basic computing paradigms at work in the brain will be a monumental breakthrough in both computer engineering and theoretical neuroscience. I believe that the breakthrough (or a series of breakthroughs) will occur in the next 40 years, perhaps significantly sooner. This means that it will happen during the professional lifetime of anyone beginning a career in computer engineering today. For some perspective, consider the advances in computation that can be accomplished in 40 years.

When I started working in computer architecture and hardware design in 1972, the highest performing computer was still being constructed from discrete transistors and resistors. The CDC 7600, which began shipping in 1969, had a clock frequency that was two orders of magnitude slower than today's computers, and it had approximately 512 KB of main memory. Now, 40+ years later, we have clock frequencies measured in GHz and main memories in GB. The last 40 years has been an era of incredible technology advances, primarily in silicon, coupled with major engineering and architecture refinements that exploit the silicon advances.

It is even more interesting to consider the 40 years prior to 1972. That was a period of fundamental invention. In the early 1930s, Church, Gödel, and Turing were just coming onto the scene. Less than 30 years prior to 1972, in 1945, von Neumann wrote his famous report. Twenty years prior, in 1952, Seymour Cray was a newly minted engineer. Just before our 1972 division point, the CDC 7600 had a very fast in-order instruction pipeline and used a RISC instruction set (although the term hadn't yet been coined). Also in the first 40 years, cache memory, micro-coding, and issuing instructions out-of-order had already been implemented in commercially available computers; so had virtual memory and multi-threading.

To summarize: the most recent 40 years of computer architecture, spanning an entire career, has largely been a period of application and technology-driven *refinement*. In contrast, the 40 years before that was an era of great *invention*—the time when the giants walked the earth. Based on an admittedly self-taught understanding of neuroscience, I believe we are at the threshold of another 40 years of great invention—inventing an entirely new class of computing paradigms.

Computer architects and engineers have a number of roles to play in the discovery of new computing paradigms as used in the brain's neocortex. One role is developing large scale, Big Data platforms to manage and analyze all the information that will be generated by connectome-related projects. Another role is developing special purpose computing machines to support high performance and/or efficient implementations of models proposed by theoretical neuroscientists.

The role that I emphasize, however, is as an active participant in formulating the underlying theory of computation. That is, computer architects and engineers should be actively engaged in proposing, testing, and improving plausible computational methods that are fundamentally similar to those found in the neocortex.

Computer architecture and engineering, in the broad sense, encompasses CMOS circuits, logic design, computer organization, instruction set architecture, system software, and application software. Someone knowledgeable in computer architecture and engineering has a significant understanding of the entire spectrum of computing technologies from physical hardware to high-level software. This is a perspective that no other research specialization has.

I am sure that many computer architects and engineers would love to work on the extremely challenging, far-reaching problem of understanding and implementing paradigms as done in the brain. Unfortunately, there is a significant barrier to entry. That barrier is the daunting mountain of neuroscience literature combined with the well-established scientific culture that has grown up alongside it (e.g., vocabulary, representation style, mathematical style). This isn't insignificant, by the way: the language you use shapes the way you think.

So, how to overcome this barrier? Answering that question is the over-arching theme of this book.

First, it requires a lot of reading from the mountain of neuroscience literature; there is no shortcut, but the papers cited herein can provide some guidance. Then, by taking a bottom-up approach, computer architects and engineers will achieve their best leverage. At the bottom is the abstraction from biological neural systems to a mathematics-based formulation. In conventional computers, the analogous abstraction is from CMOS circuits to Boolean algebra. A computer architect, with some perspective and insight, can start with biological circuits (as complicated as they may seem) and model/abstract them to a practical mathematics-based computational method.

This book contains a description of relevant biological features as background. Then drawing from these biological features, a mathematics-based computational paradigm is constructed. The key feature is spiking neurons that perform communication and processing in *space-time*, with emphasis on *time*. In these paradigms, time is used as a freely available resource for communication and computation. Along the way, a prototype architecture for implementing feedforward data clustering is developed and evaluated.

Although a number of new ideas are described, many of the concepts and ideas in this book are not original with the author. Lots of ideas have been proposed and explored over the decades. At this point, there is much to be gained by carefully choosing from existing concepts and ideas, then combining them in new and interesting ways—engineering, in other words.

The particular modeling choices made in this book are but one set of possibilities. It is not even clear that the methods explored in this book are eventually going to work as planned. At the time of this writing, the ongoing design study in the penultimate chapter ends with an incomplete prototype neural network design.

There is no doubt many other approaches and modeling choices that could be, and should be, explored. Eventually, someone will find just the right combination of ideas—and there is every reason to expect that person will be a computer engineer.

***

If the reader's goal is to achieve a solid understanding of spike-based *Neural Computation*, then this book alone is not enough. It is important to read material from the literature concurrently. There is a long list of references at the end of this book; too long to be a practical supplementary reading list. Following is a shorter, annotated list of background material. None of the listed papers is standalone; rather, each provides a good touchstone for a particular area of related research.

Neuron-Level Biology:

Just about any introductory textbook will do. Better yet, use a search engine to find any of a number of excellent online articles, many illustrated with nice graphics.

Circuit-Level Biology:

Mountcastle, Vernon B. "The columnar organization of the neocortex." *Brain* 120, no. 4 (1997): 701–722.

Buxhoeveden, Daniel P. and Manuel F. Casanova. "The minicolumn hypothesis in neuroscience." *Brain* 125, no. 5 (2002): 935–951.

Hill, Sean L., et al. "Statistical connectivity provides a sufficient foundation for specific functional connectivity in neocortical neural microcircuits." *Proceedings of the National Academy of Sciences* (2012): E2885–E2894.

The paper by Mountcastle is a classic, mostly summarizing his groundbreaking work on the column hypothesis. The paper by Buxhoeveden and Casanova is an excellent review article. The paper by Hill et al. is from the Markram group in Switzerland; it is experimental work that attempts to answer the right kinds of questions regarding connectivity.

Modeling:

Morrison, Abigail, Markus Diesmann, and Wulfram Gerstner. "Phenomenological models of synaptic plasticity based on spike timing." *Biological Cybernetics* 98, no. 6 (2008): 459–478.

The operation of synapses is critical to the computational paradigm, and this is an excellent modeling paper specifically directed at synapses and synaptic plasticity. This and other work by Gerstner and group should be at the top of any reading list on neuron modeling.

Theory:

Maass, Wolfgang. "Computing with spiking neurons." In *Pulsed Neural Networks*, W. Maass and C. M. Bishop, editors, pages 55, 85. MIT Press (Cambridge), 1999.

Maass did some seminal theoretical research in spiking neural networks. This paper summarizes much of that work along with related research by others.

Computation:

Masquelier, T., and Simon J. Thorpe. "Unsupervised learning of visual features through spike timing dependent plasticity." *PLoS Computational Biology* 3, no. 2 (2007): e31.

Bohte, Sander M., et al., "Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks." *IEEE Transactions on Neural Networks*, 13, no. 2 (2002): 426–435.

Karnani, Mahesh, et al. "A blanket of inhibition: Functional inferences from dense inhibitory connectivity." *Current Opinion in Neurobiology* 26 (2014): 96–102.

Simon Thorpe is a pioneer in spiking neural networks of the type described in this book. All the work by Thorpe and associates is interesting reading, not just the paper listed here. The work by Bohte et al., builds on earlier radial basis function research, which should also be read. The paper by Karnani et al. is a nice discussion of inhibition and the modeling thereof.

Machine Learning:

Ciresan, Dan, et al. "Flexible, high performance convolutional neural networks for image classification." *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*. 2 (2011): 1237–1242.

The neural networks being developed in this book fit within the broad discipline of machine learning. Consequently, there are some similarities with conventional machine learning approaches. This paper describes a deep convolutional neural network of about the same scale as the networks studied here.

Meta-Theory:

Chaitin, Gregory. *META MATH! The Quest for Omega*. Vintage, 2008.

The book by Chaitin is fairly easy-to-read and is imbued with the concepts and philosophy behind algorithmic information theory. When studying a computing paradigm that is not human-designed, it is good to have a "meta-" perspective.

Acknowledgments

I would like to thank Raquel for her love and great patience. Without her unwavering support, writing this book would not have been possible. Five years ago, Mikko Lipasti started me down this research path, along with Atif Hashmi and Andy Nere, and for that I will always be thankful. I am grateful to Mark Hill for providing the initial impetus for this book and Margaret Martonosi, his successor as editor of this series, for her insightful suggestions which had a major impact on the book's eventual direction. I thank Mario Nemirovsky for providing very helpful advice along the way, and Ravi Nair, Tim Sherwood, Abhishek Bhattacharjee, and an anonymous reviewer for their many helpful comments and suggestions. Thanks to Deb Gabriel and the staff at Morgan & Claypool for their efforts. Mike Morgan deserves a special thanks not only for his support of this book, but also for providing a forum for authors to express and disseminate their ideas.

PART I

Introduction to Space-Time Computing and Temporal Neural Networks

CHAPTER 1

Introduction

Einstein rides a bicycle—a concise summary of the brain's capabilities. It can take streams of input from the vision system, the sense of balance, and the sense of place. It can combine and process this information to generate an orchestrated plan that drives myriad motor neurons to fire in a precisely coordinated way to keep the bicycle upright and moving in a chosen direction. And then there is Albert Einstein. The brain can reason deeply and abstractly, divorced from all the senses. It can conceptualize the cosmos as well as things happening deep inside an atom.

Photo courtesy of the Archives, California Institute of Technology.

In mammals, cognitive skills are performed in regions of the brain's *neocortex*. The neocortex is the folded gray surface layer of the brain—the part of the brain that is clearly visible in drawings or photographs. The human neocortex weighs about one kilogram, has a volume of about a liter, and consumes about 20 watts of power. Yet, it has capabilities that far exceed those of our conventional computing methods (including state-of-the-art artificial intelligence).

Despite intense study by neuroscientists for well over 100 years, most of the secrets regarding cognitive functions are still well hidden. It is heartening, however, to consider that after scientists have unraveled many of evolution's inventions, we see cleverness and simplicity. Consider the eye and the inter-operation of the cornea, the iris, and the retina. Consider the cardiovascular system.

Although composed of many cell types, operating in complex ways, controlled by a wide variety of hormones, the cardiovascular system is a marvel of architectural simplicity when separated from the implementation details. Decades from now, when the cognitive functions are finally understood, will we be stuck with painfully complicated models? Or, in retrospect, will we be amazed by the simple elegance? This book is predicated on an affirmative answer to the second question.

It is proposed that we approach the topic of neocortex-like cognition (computation) in a different way than we normally do when dealing with computational methods. The starting point of an alternative approach is to consider systems in which time is an essential communication and computation resource. As will be explained later, a computing system based on such a *space-time* model is unlike almost every computational system we have constructed. A good reason that evolution may have selected such an approach is that time has some ultimate engineering advantages: it is freely available, consumes no power, and requires no space.

Consequently, the objective of this book is to propose and explore computational paradigms that may belong to the same class as those employed in the neocortex. This exploration is done via the experimental design of computing systems that exhibit basic cognitive capabilities. Experiments take the form of detailed simulations, with simulation results feeding back into the paradigm formulation process.

Before going any further, we start with some essential background summarizing the way neurons work.

1.1 BASICS OF NEURON OPERATION

According to the more-than-century-old neuron doctrine [12, 105], the *neuron* is the fundamental element for structure and function in the brain's neocortex. Huge numbers of interconnected neurons communicate and compute via voltage pulses or *spikes*. When viewed as a computing problem, then, the principal computational element is the neuron, and voltage spikes are the primary means of encoding and communicating information.

Figure 1.1 illustrates two of the vast number of interconnected neurons that make up the neocortex. A neuron is a specialized cell that consists of a *body* surrounded by a *membrane*. The neuron's body is fed through inputs—*dendrites*, and it has an output—the *axon*, which branches out and is connected to the dendrites of many other neurons. A *synapse* is at the connection point between an axon and a dendrite.

Figure 1.1 shows the electrical behavior of biological neurons, *excitatory* neurons in this example. In the figure, a synapse is the connection point between an upstream neuron and a downstream neuron. A single neuron has thousands of such synapses attached to its dendrites, connecting it to hundreds or thousands of axons driven by upstream neurons.

In Figure 1.1, three virtual probes (dashed arrow lines) measure voltage levels at certain points in this tiny neural circuit. Consider a sequence of events that begins with a spike being emitted from the upstream neuron. The spike travels along the axon and reaches the synapse connecting it with the downstream neuron. Upon arriving at the synapse, the spike opens a conductive gate via a relatively complex biochemical process. The now-open conductive gate immediately allows ions to flow into the downstream neuron body, thereby raising its membrane potential. The effect on membrane potential is shown as the *response* waveform near the bottom of the figure. A more detailed picture of the relationship between a spike and its response is in Figure 1.2.

A synapse has an associated *weight*, which controls its relative conductivity. A stronger synapse has higher conductivity, resulting in a response function with higher amplitude. To continue the gate metaphor, the greater the synaptic weight, the wider open the ion gate swings in response to an incoming voltage spike.

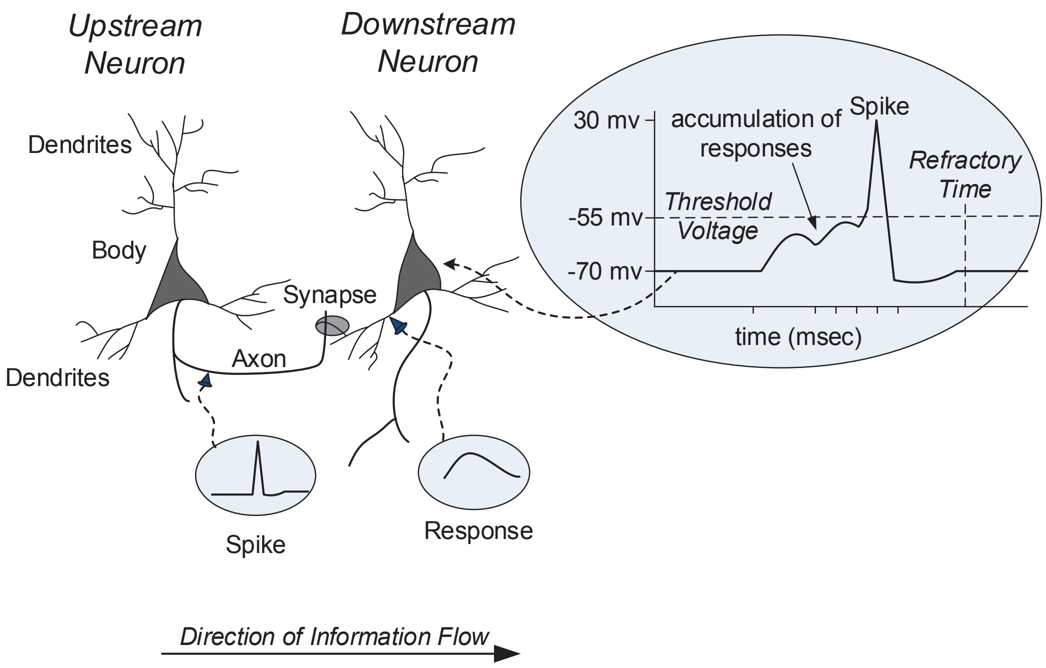


Figure 1.1: Neuron operation. Two (excitatory) pyramidal neurons are connected via a synapse. Attached dotted line "probes" illustrate dynamic operation.

Immediately after the synapse's conductive gate swings open, it starts to close with exponential decay, so the flow of ions into the downstream neuron gradually diminishes. All the while, ions leak from the neuron body, thereby decreasing the membrane potential also with exponential decay, but with a longer time constant than closing the conductive synapse gates. This combination of exponential decays with different time constants gives the excitatory response its distinctive shape, as shown in Figure 1.2. It is loosely analogous to filling a leaky bucket (the neuron body), from another leaky bucket (the synapse), where the lower body bucket leaks more slowly than the upper synapse bucket. The neuron's membrane potential is the amount of water in the body bucket.

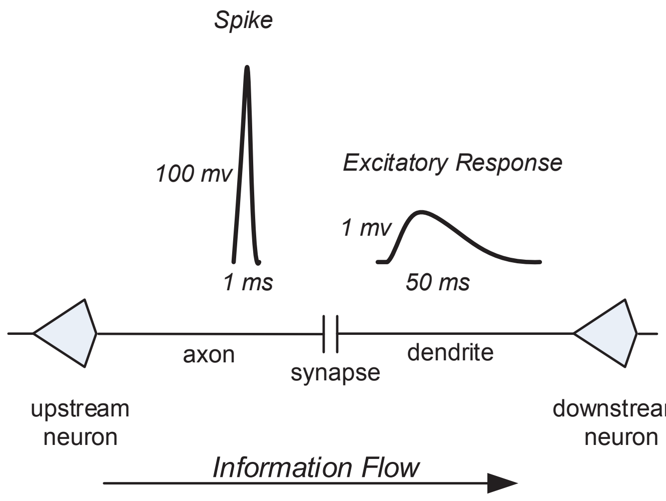


Figure 1.2: A spike emitted by an excitatory neuron flows along out along the axon. When it reaches the synapse, it invokes a response in the membrane potential of the downstream neuron (the excitatory response). The response caused by a spike from an inhibitory neuron is similar, except the polarity is reversed and the time constants may differ.

Next, consider the more detailed waveform shown on the right side of Figure 1.1. As multiple spikes are received at a neuron's input synapses, each of them will contribute its excitatory response to the neuron body. If received relatively closely together in time, the effects of the responses accumulate, raising the total membrane potential, as shown in the waveform. If the potential reaches a critical level, the *threshold voltage*, then an avalanche effect takes place, and the neuron emits an output spike that is transmitted along its axon to all of its downstream neurons. Immediately following the output spike, there is a brief *refractory period*, during which the neuron cannot fire.

In some instances, there may be insufficient input spikes to raise the membrane potential to the threshold voltage. Consequently, the membrane potential will eventually decay back to the rest potential and no spike will be emitted.

In the waveform at the right side of Figure 1.1, typical voltage levels are shown. These are a base level of roughly -70 mv, a threshold of about -55 mv, and a maximum spike level of about 30 mv. The refractory time is on the order of a few msec. In the illustration, the accumulated responses of only three input spikes in close proximity are sufficient to raise the membrane potential to the threshold. In reality, the number of spikes is commonly an order of magnitude higher, but can vary over a wide range.

As just described, synaptic weights are a major factor in determining a given neuron's overall function. Weights are established via a training process, where commonly occurring spike patterns are learned and encoded as patterns of synaptic weights. Consequently, when such a learned pattern is detected at the neuron's input, the synaptic weights yield a strong response, and the neuron emits an output spike. The closer the pattern matches the learned pattern as reflected in the weights, the sooner the output spike occurs.

The commonly used rationale for synaptic weigh training is that if the upstream neuron's spike *precedes* the output spiking of the downstream neuron, it contributes to the occurrence of the downstream neuron's spike, so its influence should be enhanced in the future by increasing the synaptic weight. On the other hand, if the upstream neuron's spike occurs *after* the downstream neuron's spike, it had nothing to do with the downstream neuron's spike so its influence should be diminished in the future (by decreasing its synaptic weight). Eventually, all the weights converge to values that collectively characterize the average input spike pattern.

There are two basic types of neurons: *excitatory* and *inhibitory*. Roughly 80% of neurons are excitatory, and most of them are pyramidal neurons, as exemplified in Figure 1.1. A spike from an inhibitory neuron invokes an *inhibitory response* when it reaches the synapses. An inhibitory response has the opposite polarity of an excitatory response so it reduces the downstream neuron's membrane potential. In this book, excitatory neurons are modeled as just described. However, inhibitory neurons are modeled at a higher level.

The spike propagation delay from one neuron to the next is of the same order as the processing latency of a neuron (the time between input and output spikes), and there can be fairly wide variations in both propagation delay and processing latency. Furthermore, when one neuron feeds another, there are often multiple connection paths between the two. That is, the axon from the upstream neuron makes multiple connections with dendrites belonging to the same downstream neuron. The propagation delays along these paths may differ; in fact, there is enough irregularity in dendrite and axon structure that the multiple propagation delays almost certainly differ.

Terms

The above description of neuron operation uses common terms for some of the neuron features and functions, although some also have technical terms. In this book, common terms are preferred and are used whenever possible. For reference, following is the correspondence between common terms and technical terms, along with their associated acronyms.

Common Term	More Technical Term
body	soma
spike	action potential (AP)
response	post synaptic potential (PSP)
excitatory response	excitatory post synaptic potential (EPSP)
inhibitory response	inhibitory post synaptic potential (IPSP)
upstream neuron	pre-synaptic neuron
downstream neuron	post-synaptic neuron

Schematics and Symbols

It is often useful to draw schematic diagrams of interconnected neurons to illustrate their inter-operation. In the neuroscience literature there doesn't seem to be a widely accepted standard for schematics, and schematics are usually drawn in a way that reflects the relative physical orientation and location of biological neurons.

In this book, schematics are drawn in a way that makes functional relationships apparent, without consideration for such things as physical orientation. Signal (spike train) flow is usually from left to right. The neuron symbols used throughout this book are illustrated in Figure 1.3.

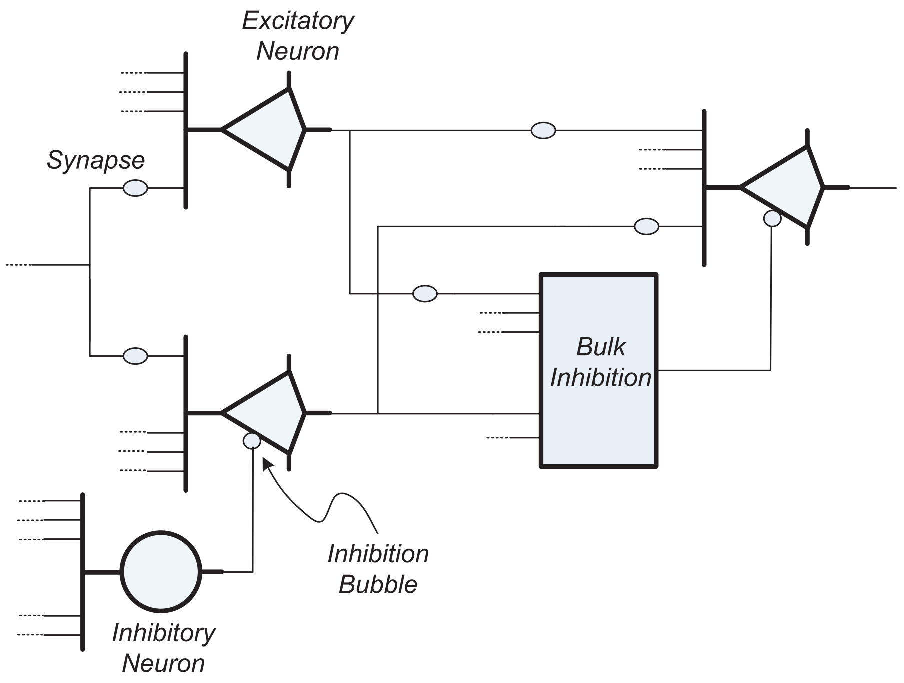


Figure 1.3: Illustration of symbols used in figures.

An excitatory neuron has the general shape of a pyramid, turned on its side. Inputs are on the left and the output is on the right. An inhibitory neuron is drawn as a circle. In this work, however, inhibitory neurons will seldom be drawn individually. Rather, because inhibitory neurons operate in bulk, a collection of inhibitory neurons will be drawn as a single box. Inhibition will then be represented at the target sites (excitatory neurons) as a bubble.

The symbol for a synapse is optional and usually not used; it only reflects the presence of a connection, which is redundant when the connection is explicitly drawn. Nevertheless, for illustrative purposes, it is sometimes useful to explicitly draw a synapse.

1.2 SPACE-TIME COMMUNICATION AND COMPUTATION

Any physical computer consumes both space and time—it operates in *space-time*. The concept of space-time is usually employed in modern physics to describe non-intuitive relativistic effects: the warping of space-time near a black hole. However, space-time also exists in a boring non-warped form almost everywhere else. We live and breathe in a uniform space-time environment where the passage of time can be a useful communication and computation resource.

Here we use the physics term "space-time" rather than more commonly used "spatio-temporal". There are two reasons for this. First, the two-syllable "space-time" rolls off the tongue much more easily than the five-syllable "spatio-temporal." Second, and more importantly, we are interested in computing methods that rely on the passage of physical time.

1.2.1 COMMUNICATION

The potential of the "time" part of space-time is illustrated by the simple system in Figure 1.4. Information is transmitted from sender to receiver over a channel; call it a *wire*, but the communication medium could be anything, an axon for example. Information is conveyed through voltage pulses or *spikes* that travel on the wire having a *constant* transmission delay (Δ0).

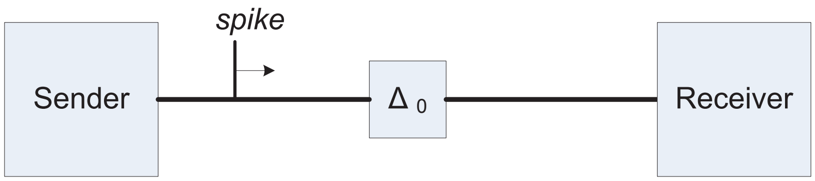


Figure 1.4: A single wire system for communicating information. Spikes travel from sender to receiver with a fixed delay.

For sake of discussion, assume both the sender and receiver have the ability to measure time intervals to within 1 msec. Hence, the sender is able to emit spikes that are separated by time intervals to a precision of 1 msec, and the receiver is able to determine the time interval between two received spikes, also to a precision of 1 msec. These measurements are performed independently at both the sender and receiver. As will now be shown, this capability yields an extremely energy efficient communication method based on spatially separated time measurements.

Assume the total energy cost for communication comes from generating, transmitting, and receiving individual spikes. Then a simple energy efficiency metric is the ratio of information sent (in bits) to the number of spikes that transmit the information. If *n* spikes transmit *b* bits of information, then the *energy efficiency metric* is *b/n* bits per spike. A higher number is better.

Although the term "spike" is used, spikes can be given conventional binary interpretations. In terms of bits, the communication model is one in which 0's and 1's are transmitted over the wire at 1 msec intervals. Very importantly, sending a binary 1 (a spike) consumes energy, but sending a 0 (non-spike) does not.

In order to send a packet of information over the wire, one could use conventional bit-serial encoding. That is, the sender can first send a synchronizing *start* spike on the wire, then immediately at 1 msec intervals it can send a serial binary sequence of spikes/non-spikes representing 1's/0's. For example, to send a byte of information, there is a start spike plus a bit-serial pattern containing from 0 to 8 spikes spread over eight 1 msec intervals. If all byte patterns are equally likely to be sent over the wire, then an average of 1 + 4 spikes will be required for sending a byte of information. This is an average energy efficiency of 8 bits per 5 spikes or 1.6 bits per spike.

An alternative method, is to send a *start* spike, wait *n* msec, then send a *stop* spike, where *n* is a binary number between 1 and 256: essentially the content of the byte. The information "*n*" is encoded by an *n* msec time interval between *start* and *stop* spikes. Because the sender and receiver have timing agreement to within 1 msec (due to uniform space-time), both the sender and receiver will agree on the intervening time interval. In this scenario 8 bits of information are transmitted by 2 spikes, so the energy efficiency is 4 bits per spike. This is over twice as efficient as the 1.6 bits per spike provided by the bit-serial scheme. Furthermore, if a bus is used instead of a single wire and all the wires in the bus share the same start spike, it is relatively easy to approach 8 bits per spike on a bus of modest width.

Where does the high energy efficiency come from? The answer follows from the observation that the bit-serial scheme takes a total of 9 msec to communicate a byte, and the two-spike scheme takes an average of about 128 msec. The two methods illustrate a tradeoff between energy consumption and time. By taking more time, we use less energy. From the perspective of evolution developing the neocortex, the "time" part of space-time may well have been used in a similar manner as an effective way of reducing energy requirements. In the neocortex, data precision is very low, three to four bits—not even a byte. This means that communication times are reasonable, even if not as good as bit-serial. Furthermore, increased communication latencies can be at least partially offset by the vast parallelism in the neocortex.

Observe that with this approach, the time required to transmit information *is* the information. Furthermore, this communication method is completely self-synchronized—the mere presence of the start spike is all that is needed to coordinate the communication. So, the same voltage spikes simultaneously coordinate and communicate information. This is another, less direct way of saving energy.

1.2.2 COMPUTATION

Normally, we don't think of sending information on a wire as performing a mathematical function, but if we use time as a resource, computation can be performed simultaneously with transmitting information. Consider Figure 1.5, where there are two wires with different delays: one is Δ_0 as before, and the other is Δ_0 plus 1 time unit (say 1 msec). Here, the communication process performs the elementary function $f(n) = n + 1$. The time difference between the receipt of the spike on the first wire and the receipt of the spike on the second wire determines the function's output. It is as if the sender pushes an encoded value n into a function evaluation box and $n + 1$ pops out the other side.

In general, by selecting the delay appropriately, one can both communicate and perform $f(n) = n + c$ for any constant c. Observe that the time required to compute the sum *is* the sum.

Although $n + c$ is a simple function, evaluating it consumes energy in a conventional digital CMOS implementation. That is, when adding two binary numbers, several energy-burning signal transitions are required per adder stage. In the case of the wire system, however, addition is a byproduct of communication over paths having different delays. The only cost is transmitting two spikes over wires with fixed delays—the addition is performed simultaneously with communication.

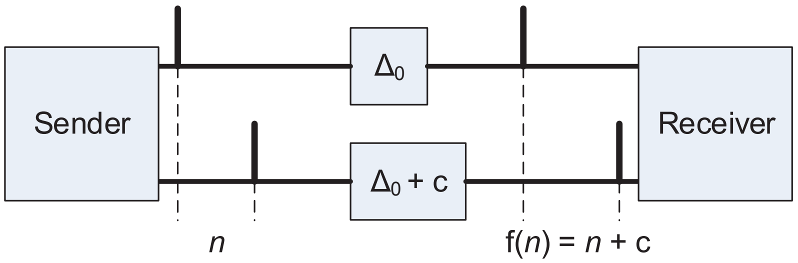


Figure 1.5: Spike-based computation of $f(n) = n + 1$.

In the neocortex this apparently simple method of computation-through-delay potentially leads to a fairly complex set of controllable computations. Recall from the neuron overview that two neurons are typically connected via several different paths with different delays. Consequently, the synapses on these different paths, via their trained weights, may selectively activate and/or de-activate the multiple parallel paths in some coordinated way, thereby achieving a programmable family of functions—which are implemented through delays in the communications paths. This is illustrated in Figure 1.6.

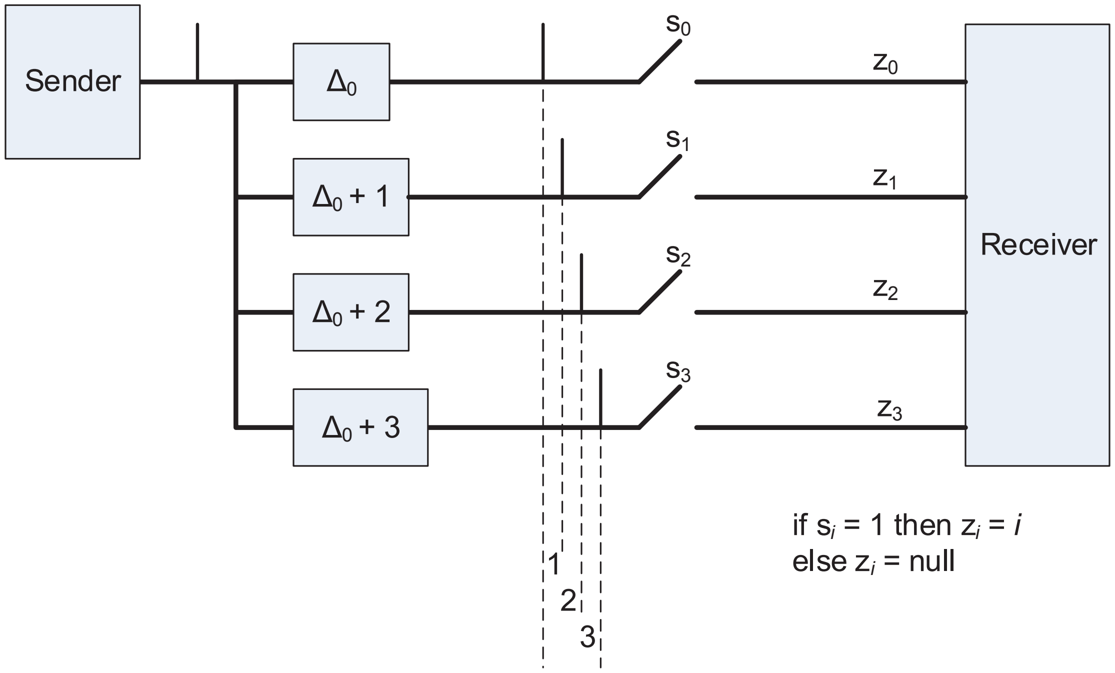


Figure 1.6: Sending a single spike results in multiple received spikes having different delays. Switches (synapses) then control the presence (closed switch) or absence (open switch) of a received value.

Figure 1.6 shows a single spike being sent, with multiple copies being received and each copy having a different delay. Furthermore, switches, labeled s_i, can be opened or closed to allow the delayed spike through to the receiver or not. Simply put, for every i, if the switch s_i is closed, then the output z_i is equal to i. In this way, a fairly complex vector (the z_i) can be constructed at the receiving end, in response to a single transmitted spike.

For example, in Figure 1.6 if $s_0 = 1$ (closed), $s_1 = 0$ (open), $s_2 = 1$, $s_3 = 1$, then the output is $z_0 = 0$, $z_1 = \text{null}$, $z_2 = 2$, $z_3 = 3$. This may seem like an odd little function, but depending on the switch settings, a large set of output vectors can be produced in response to an input spike. That is, the switch settings (synapses) can encode a rather rich amount of information with a single transmitted spike.

The wire examples just given implement their functions in a passive way. Active, more complex computation can be performed by devices that take spike times as an input encoding and then emit an output spike at a time that is a function of the input spike times. Reflecting back to the neuron overview given above, this is exactly what a biological neuron does. That is, each input spike produces a response function of time, the response functions from multiple input spikes are added, and when (if) a threshold is met, an output spike is generated.

There is plenty of experimental support for active computation based on spike timing in the neocortex. For example, excitatory neurons are capable of maintaining a precision of 1 msec: "stimuli with fluctuations resembling synaptic activity produced spike trains with timing reproducible to less than 1 millisecond" [60]. This and other experimental support for computation via precise neuron timing will be laid out in Section 3.5.6.

1.2.3 DISCUSSION

Using time as a communication and computation resource has at least two significant advantages: energy efficiency and the ability to compute via localized self-timed elements (neurons) while achieving a coordinated global outcome; coordination is provided by the uniform passage of physical time. For these reasons, and others, a fundamental hypothesis motivating the work in this book is that the neocortex uses time in this way.

Hypothesis: The neocortex uses time as an important communication and computation resource. Results are entirely dependent on the physical time it takes to compute and communicate them.

Although a physical implementation that relies on time as a freely available resource may have significant implementation advantages, striving for a neuromorphic hardware implementation that literally uses time in this way is not an objective of the work presented here. In this work, we are not aiming for highly efficient analog networks that literally communicate via voltage spikes.

Rather, the premise is that advantages of using time as a communication/computation resource pushed evolution in a certain direction: toward a particular way of computing in the neocortex, toward a certain class of computational paradigms. And, very importantly, these paradigms are fundamentally unlike the ones we normally use when designing conventional computer systems.

It is asserted that the way that we (computer designers) ordinarily deal with time restricts the class of functions we can implement in a natural, intuitive way. Consequently, we tend to implement the class of functions that are readily amenable to Turing machine or von Neumann style implementations based on algorithmic steps, where the physical time consumed in implementing each step does not affect values computed during that step. With all computation models that we normally use, results computed during a time step are independent of the time consumed in a physical implementation.

It is further asserted that implementations that rely on physical time as a computing resource readily support a different class of functions than those supported by time-independent implementations.

If the above assertions are true, then the neocortex supports functions that are different than the ones supported by our conventional time-independent computer implementations. Consequently, the types of computation models used in the neocortex may be concealed from us because we innately think only about computing systems with time-independent implementations.

The issue is not whether such cognitive functions are theoretically computable in the Turing sense, because they are. Rather, the issue is whether our way of thinking about computation, and the role of time in particular, inhibits us from *implementing* them, or possibly even considering them. In contrast, driven by advantages such as simplicity and low energy consumption, evolution may have discovered effective time dependent methods for implementing cognitive functions.

So, at a high level, it is the general class of space-time computing paradigms we are interested in. However, because they differ from the way we ordinarily think about computing, we can't use our ordinary methods to describe and develop them. If we can't use our familiar methods, how do we proceed? The answer is that we put forward some general principles, then use experimentally observed operation of the biological neocortex for guidance. We then explore a broad new class of paradigms, from the bottom up, in a trial-and-error fashion, starting with the example that we know exists ("I think, therefore I am"). However, we are not restricted to biologically based examples—there may be other ways of using space-time besides the way the neocortex uses it. It is hoped that exploration of space-time methods, starting with neocortex-like operation, will eventually reveal a new class of computational methods that are capable of greatly expanded cognitive functions.

A more formal definition of space-time computing and its connection to operation of the neocortex are given in Chapter 2. Because the work in this book falls under the broad categories of both neural networks and machine learning, the following two background sections provide some perspective.

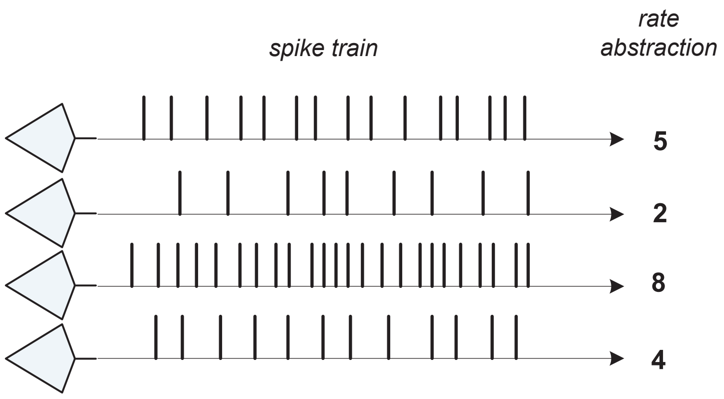
1.3 BACKGROUND: NEURAL NETWORK MODELS

The computing systems considered in this book are a certain type of spiking neural network (SNN). The next section discusses what this means and places the particular type of SNNs studied here into the larger context of artificial neural networks (ANNs). A simple taxonomy of neural networks is developed, based on (1) the ways that information is encoded and (2) the ways that neurons operate on this information.

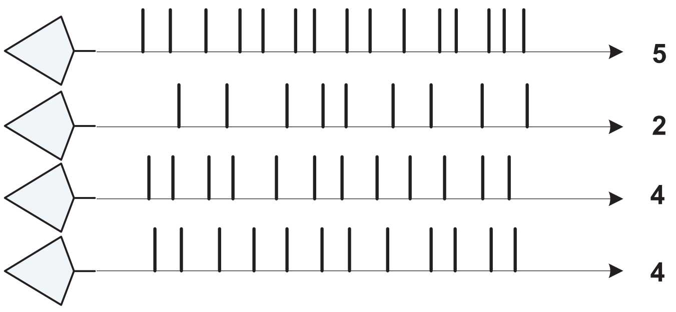
The assumption is that interneuron communication is via streams of spikes (*spike trains*) generated by upstream neurons and transmitted over axons. An input spike train passes through synapses and dendrites and causes downstream neurons to respond as described in the preceding section.

1.3.1 RATE CODING

The classical way of interpreting the information contained in spike trains is to use spike rates. In the taxonomy, there are two ways of representing rates; both are shown in Figure 1.7a. One way represents information in original spike train form, i.e., individual spikes are tracked in the neural network. The other rate coding representation encodes information as numerical values. Both methods are based on the assumption that spike rates on individual lines encode information independently of the rate codings on other lines. This is illustrated in Figure 1.7b, where a change in the third line from 8 to 4 has no effect on the other lines.



(a)



(b)

Figure 1.7: Information encoded as spike rates. (a) Spike rate information may be represented as a spike train, or as an abstract, numerical, rate. (b) If the spikes on one of the lines changes (the third line is reduced from 8 to 4), the rates (information) on the other lines does not change.

In Figure 1.7, rates are normalized so they fit within the range 0 to 8. Classical ANNs fit this coding model; in classical ANNs rates are typically scaled to the range of 0 to 1. Also included are the more recent convolutional neural networks, deep neural networks, restricted Boltzmann machines (RBMs), and various related species. These networks are the basic building blocks, the workhorses, of most of the leading-edge machine learning methods. Some may add other functional blocks, "pooling," for example, where the outputs of a number of neurons are reduced to a single value by taking the maximum or a mean [86].

Most of today's neural networks simply encode input values as numbers, and the word "rate" doesn't usually enter the conversation. However, these networks are descended from early neural networks that were based on rate assumptions, so it's in their DNA. Put it this way: if one wanted to somehow establish biological plausibility of conventional neural networks, then the plausibility argument would almost certainly rest on a numerical spike rate abstraction.

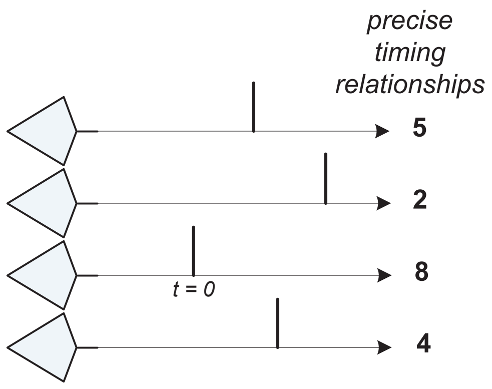
In a smaller group of proposed neural network models, the spike train representation is retained, even though information is encoded based on spike rates. This includes spike train representations based on various forms of rate modulation. A good example is the Spaun system [24]. Spaun is based on coordinated rate-modulated information coding. In that system, a group of neurons represents a vector space. A neural network based on memristors [82] is another example of a system that employs rate coding via rate modulated spike trains. GPU systems also have been applied to rate-based spiking neural networks [28].

1.3.2 TEMPORAL CODING

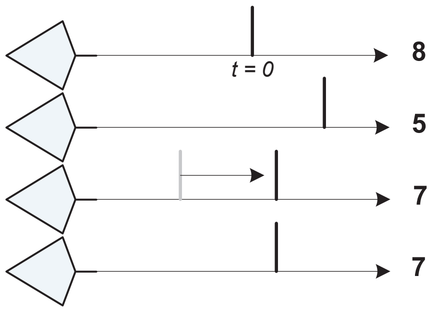
With temporal coding, information is encoded via precise timing relationships among individual spikes across multiple parallel transmission lines (axons connected to dendrites). There are several methods for encoding information based on such spike timing relationships. The basic method used in this book is explained via an example. See Figure 1.8.

This method encodes values according to spike times relative to the time of the first spike in a group. In Figure 1.8a, the values range from 0 to 8. Given that time goes from left to right, the leftmost spike is first. Because it is first, it is assigned the maximum value of 8. Then all the other spikes encode values that are relative to the 8. For example, the latest occurring spike (2nd line from the top) is assigned the much lower value of 2. The bottom spike is 1/3 of the relative time separating the 2 and the 8 and therefore encodes a value of 4, and the top spike that occurs slightly earlier encodes a value of 5. Details of this coding method are given in Section 6.2.

Because actual spike time relationships convey information, network transmission delays are a crucial consideration in neural networks using this coding method. A change in network delay along some axon-synapse-dendrite path can modify the information being conveyed on that path. For example, in Figure 1.8b the spike on the third line is delayed to a later time. As a consequence, not just the value on the third line changes, but *all* the values change because the spike at $t = 0$ changes. This is quite unlike what happens with rate coding when we change the value on one of the lines as described above. Here, the values on lines other than the third appear to change even though there is no apparent physical connection. In a physical implementation, however, there actually *is* a connection: physical time.



(a)



(b)

Figure 1.8: (a) Relative spike times encode information across a bundle of lines. The earliest spike has the highest value and later spikes have values that are temporally relative to the first. (b) If the spike on one of the lines is delayed (the spike on the third line shifts later in time) then the values on all the other lines change as well as the value on the third line.

1.3.3 RATE PROCESSING

In networks where information is coded as abstract rates (numbers), the neurons naturally operate on rates, i.e., numerical values falling in some fixed range. Abstract rate coding combined with rate-based neurons defines the classic ANNs and their many descendants.

In rate-based systems, values are confined to a certain fixed range. Given rate-coded inputs, the typical ANN paradigm first computes a vector inner product (weights · input values) followed by a non-linear activation, or "squashing," function of the resulting inner product. Refer to Figure 1.9. That is, it squashes the output value so that it fits within the pre-defined range. A simple squashing function that saturates for values outside the range is a good example. ANNs are typically constructed with a differentiable squashing function, so tanh or 1/(1-e^{-x}) are often used. These are examples of "sigmoid" functions, so-called because of the general S-shape of their curves. Differentiability is important for training methods that employ back propagation and gradient descent based on partial derivatives [52].

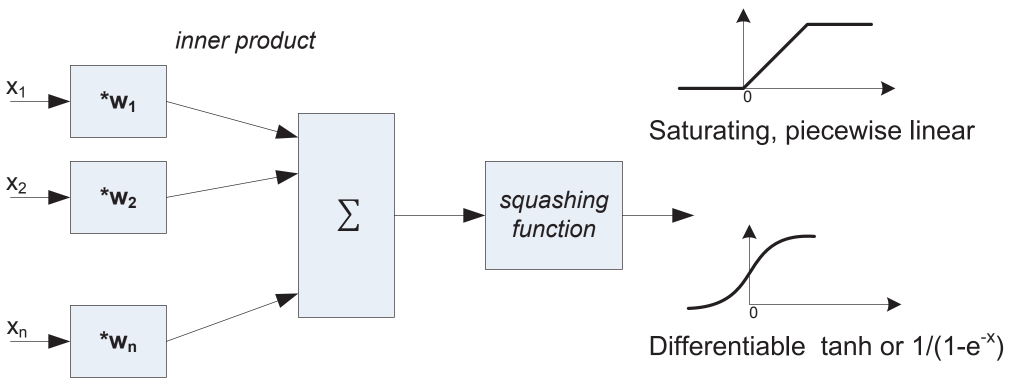


Figure 1.9: Model neuron as used in conventional neural networks.

1.3.4 SPIKE PROCESSING

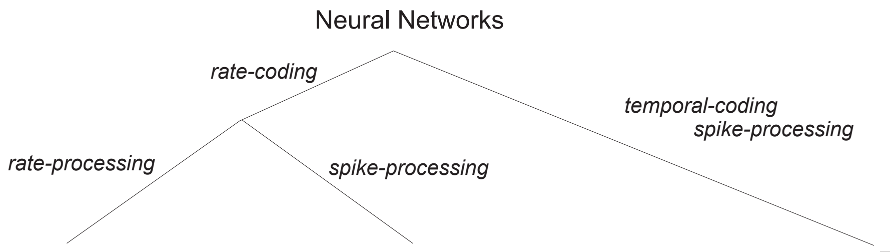
A neural network that employs spike processing uses neurons that operate on spike inputs. The input spikes can either encode a rate (via a spike train) or encode a temporal vector via a single spike on each neuron input.

Regardless of the coding method, these neurons typically model neuron operation as depicted in Section 1.1, i.e., biology-like spike-and-response behavior. Such neuron models are generally called "leaky integrate and fire" (LIF) neurons. (All of this is discussed in detail in Chapter 5.)

For example, the Spaun system [24] uses leaky-integrate-and-fire neurons to operate on rate-coded spike trains. The neurons were co-developed with the rate-modulated coding system. Consequently, the spiking neurons in Spaun are able to retain a rate-modulated coding discipline throughout the system. Other systems convert input values to Poisson distributed spike bursts, with the mean determined by the input value [5, 9, 23, 76, 82]. LIF neurons then operate on these rate modulated spike trains.

1.3.5 SUMMARY AND TAXONOMY

Based on the above discussion, we can construct a neural network taxonomy that accounts for the way information is communicated and processed. Refer to Figure 1.10. On the left branch of the taxonomy, both rate-coding and rate-processing are used by all classical ANNs and their descendants, including deep convolutional neural networks.



ANNs in the large	SNNs	TNNs
Anything that uses: inner product & squash, i.e., virtually every machine learning method in use today—deep convolutional nets, RBMs, etc.	Eliasmith et al., 2012	Maass, 1997
	Brader et al., 2007	Thorpe and Imbert, 1989
	Diehl and Cook, 2015	Masquelier and Thorpe, 2007
	O'Connor et al., 2015	Bichler et al., 2012
	Beyeler et al., 2013	Kheradpisheh et al., 2016
	Querlioz et al., 2013	Natschläger and Ruf, 1998
		Maass et al., 2002
		Probst et al., 2012

Figure 1.10: Neural network taxonomy, based on coding and processing methods. Examples are given for each. The class of networks in this book, TNNs, use both spike coding and spike processing.

The other two neural network categories, the ones that use spike processing, are generally considered to be SNNs. One type of SNN keeps rate information in spike train form and processes at the spike level. The other type of SNN, as used in this book, encodes information using precise spike timing relationships and processes information at the spike level. These SNNs of the second type are both spike-coding and spike-processing. To avoid confusion, this second type of SNN is given the specific name *Temporal Neural Network* (TNN) to emphasize the use of both temporal coding and temporal processing. Introducing a new name for these networks is not gratuitous—some of the better-known SNNs are based on rate coding; the term TNN distinguishes the methods used here.

TNNs define a broad space-time computing universe. In this universe, communication and computation delays aren't an unfortunate byproduct of a physical implementation—rather, delays and latencies are an integral part of the way communication and computation is done. A computed result is not independent of the time it takes for signals to move through a neural network. Rather, a computed result *is* the time it takes for the signals to move through the network.

1.4 BACKGROUND: MACHINE LEARNING

The applications of interest in this book and the applications targeted by conventional machine learning approaches overlap significantly, if not completely. In the broad sense of the term, the work presented here *is* machine learning. However, mainstream machine learning is not based on a space-time paradigm as considered here. On the contrary, mainstream machine learning uses time independent methods, as does virtually every other computing method that has been devised.

Nevertheless, at a high level, features of some mainstream machine learning implementations are similar, or analogous, to the features of the TNNs described in this book. One important feature of many machine-learning methods is the use of layers of interconnected feedforward functional blocks. Included in this class of systems are deep convolutional networks as well as virtually all the other feedforward neural network variants. Hence, many machine-learning approaches share the same general structure as the systems studied here: a layered hierarchy of interconnected computational units.

Both machine learning and the approach given here rely on training (or "learning," depending on the point of view). However, it is with training that a big distinction arises. Whereas with conventional machine learning, the training process is global and extremely time-consuming, the space-time approach leads to a simple, localized training process that consumes no more computation time than evaluation.

Finally, and very importantly, widely used machine learning functions—feedforward pattern clustering and classification—are excellent drivers for the initial development of TNNs. Feedforward clustering (via unsupervised training) is of primary interest in this book. Grouping according to similarity, as done with clustering, is a basic cognitive function, and there is fairly strong experimental support for biological equivalents.

In simple terms, the clustering function maps each of a large number of input patterns onto a much smaller set of output patterns based on pattern similarity. Similar patterns belong to the same cluster. Hence, the output pattern identifies a cluster, and similar input patterns map to the same cluster identifier. Typically, the number of clusters is much smaller than the size of the input space. The clustering function is determined via a training process during which input patterns to be separated into clusters are applied to the network in an unsupervised manner.

1.5 APPROACH: INTERACTION OF COMPUTER ENGINEERING AND NEUROSCIENCE

The idea of using space-time paradigms where time is a communication and computation resource is intriguing on its own, and such paradigms could be studied independently of neuroscience. However, it is the functional capabilities of the neocortex that makes space-time computing so appealing. Therefore, it makes sense to use knowledge gleaned from neuroscience to provide guidance when proposing and exploring space-time paradigms. Consequently, the search for computational paradigms with the functionality of the neocortex will involve a combination of the disciplines of neuroscience and computer engineering.

This might suggest the coordination of computer engineering and neuroscience to co-develop a single, shared theory. However, a significant problem with such a shared theory is that neuroscience and computer engineering use theory in different ways.

A theory is useful to a neuroscientist to explain and motivate experiments that expand knowledge of the biological brain. In terms of the standard computer architecture hierarchy extending from hardware implementation up through high level software, this approach is downward-looking—toward the physical "hardware," i.e., the biology. Neuroscience drives toward theory that is inclusive: include as many biological features as possible into the theory.

Theory, in the form of a computational model, is useful to a computer engineer to drive the construction of computer systems having improved and expanded capabilities. In terms of the architecture stack, this is upward-looking. Computer engineering drives toward theory (models) that are exclusive: use abstraction and modeling to exclude unnecessary implementation details.

Here is an example. In neuroscience research, once a computational model has been developed it is common to add noise and jitter to the model and show that it still works. That is, the noise and jitter naturally present in the biological system are deemed to be an important modeling consideration, and it is therefore important to show that the proposed computing model still works despite their degrading effects. In computer engineering research, on the other hand, there is no reason to artificially add noise and jitter to a model, unless they *improve* the model in some way. For a computer engineer, the thing to do with a new computing model is to design a computer.

Even within the realm of computer engineering, however, research methods aimed at cognitive processing will differ from those used by computer engineers today. With conventional computer engineering, system design generally proceeds top-down, where a desired function is given in a well-specified manner, then software and/or hardware are designed, typically using layers of abstraction.

In the construction of space-time paradigms, we don't start with a well-defined function, and we don't have known layers of abstraction that we can fall back on. Rather, we start with a general description of what a desired function should do. For example, drawing from machine learning, we might train a spiking neural network showing it a number of example faces and then ask it to identify similar faces. Importantly, we do not provide a precise functional definition of "similarity" *a priori*. Rather, the implementation and the training process imply what "similarity" means, and we can only measure the quality of the internal similarity function indirectly via benchmarking with some assumed accuracy metric.

The research process then proceeds bottom-up rather than top-down. We design a system having a certain structure, then simulate it via benchmarks, which yield an accuracy (or some similar metric), and then, based on this result, we might tweak, or perhaps significantly change, the structure (which changes the implied function) and try again. The researcher's insight and ingenuity guides the design process toward a useful computational method.

In summary, a proposed engineering research method for studying space-time computation consists of the following steps.

1. Postulate primitive communication and computation elements upon which a paradigm can be constructed. These primitive elements may be determined through the study of neuroscience research results, both theoretical and experimental.

2. Propose a computational paradigm based on the primitives. Again, the basic features may be determined via the study of neuroscience research results. There should be enough detail that a working implementation can be constructed. Implementation includes simulation models.

3. Experimentation is performed via simulation. A typical experiment consists of first training the network, then evaluating it to determine the extent to which desired cognitive capabilities are achieved.

4. Depending on the experimental results, a proposed paradigm may be accepted, modified, or rejected. Then a new, possibly expanded paradigm is put forward, possibly with a different set of primitive elements, and the process continues.

In (3) above, the experiments are directed at understanding the functionality that can be achieved. This differs from today's computer engineering simulations where the function being performed by a computer is not at issue, only the performance or energy consumption are of concern.

In (1) and (2), one might begin with features that appear *plausible* in the biological neocortex. However, in this context, the standard of plausibility may fall significantly short of standards of acceptance used by the biological neuroscience community. From the computer engineering perspective, there is no reason to be a stickler for plausibility when the ultimate goal is an innovative way of computing. One should err on the side of expansiveness. Interesting ideas should be pursued.

Neuroscience, in contrast, generally maintains a very high standard of plausibility for acceptance. Consequently, there may be a significant gap between what is accepted as plausible when trying to engineer new computing system vs. what is plausible for neuroscientific acceptance. As research progresses in both computer engineering and neuroscience, this plausibility gap should close. Ideally, over time, the gap might close completely, resulting in a unified theory that is embraced by both computer engineering and neuroscience.

1.6 BOTTOM-UP ANALYSIS: A GUIDING ANALOGY

Imagine that a civilization of space aliens, one that uses completely different computing methods than we, visits earth and discovers our state-of-the-art computer systems containing superscalar cores.

A superscalar processor contains many features to enhance performance, as well as features to improve reliability and energy efficiency. Instructions are fetched based on predicted branches; logical registers are renamed to physical registers and placed in issue buffers. Instructions are executed in parallel, often out of their original sequence. Hierarchical cache memories support both instruction fetches and data loads and stores. Clock gating enables functional units only when needed. Error detecting and correcting codes provide protection against bit failures in memories.

If the aliens try to understand the basic computing paradigm, it would be a very long, arduous task. And it would almost certainly proceed in a bottom-up fashion. The first breakthrough would probably be the realization that precise voltage levels are not important; that voltage levels can be modeled as ones and zeros. Then, after some effort, it might be discovered that logic gates perform elementary operations and that information flows from the output of one to the inputs of others.

When knowledge reaches the point of understanding combinational functions and binary arithmetic, it would be seen as a major breakthrough. Given enough components, one could then build huge combinational networks capable of performing very useful computations. A few hundred million transistors can implement an extremely large feedforward dataflow graph; obviously not with all the capabilities of a modern day processor, but extremely useful nevertheless.

Eventually, it might be discovered that the paradigm is based on synchronized feedback-connected combinational blocks. And, finally, the von Neumann paradigm might be discovered (or re-invented). Or, the aliens may arrive at a different paradigm, a dynamic dataflow architecture, for example, which is also composed of synchronized combinational blocks.

If we compare the basic von Neumann paradigm, which is quite simple, with a modern day superscalar processor the difference is huge. A simple von Neumann processor can be constructed with a few thousand logic gates, yet a superscalar processor uses several orders of magnitude more. Why the difference?

The vast majority of the logic in a modern day superscalar processor is there to increase performance, enhance reliability, and improve energy efficiency. Meanwhile, the von Neumann paradigm is concealed, deeply enmeshed in all the circuitry implementing the performance and reliability improvements. For example, the program counter is an essential part of the von Neumann architecture. Yet, if one were to try to find *the* program counter in a superscalar processor, the task is ill-stated. There isn't a single program counter; there may be many tens or even hundreds, depending on the number of concurrent in-flight instructions. A branch predictor would probably drive the aliens crazy. It consumes a lot of transistors, yet computed results do not depend on it. Whatever its predictions, the computed results are always the same.

It may be the same in the neocortex, which is a mass of interconnected neurons that communicate through an even more massive set of synapses. In essence, the neocortex is a gigantic analog system built of unreliable parts, connected via paths with variable delays. Maintaining stability of this huge analog system very likely requires constant adjustment via a complex set of biological mechanisms, as well as consuming a large fraction of neurons for maintaining reliability. Cognition itself may appear almost as a second order effect that is deeply intertwined in the multitude of supporting neurons and mechanisms. The important point here is that many of the mechanisms at work in the neocortex are very likely there for reasons other than implementing the computational paradigm. This is expanded upon in Section 4.7.

The analogy of the space aliens' bottom-up discovery of conventional computing paradigms provides some guidance in the search for the cognitive paradigm(s) used in the neocortex. There is broad consensus that voltage spikes convey information. There is also consensus regarding the basic operation of neurons. The critical next step is to understand the paradigm at a level analogous to combinational networks (i.e., feedforward networks where the output is a function of only the current input(s)). Given that understanding, one could begin to engineer large feedforward spiking neural networks capable of useful tasks. Then, armed with feedforward functionality, engineers may be able to design and construct more efficient and practical recurrent networks with feedback, thereby achieving the next big step in implementing higher-level cognitive tasks.

1.7 OVERVIEW

This book traces the author's ongoing development of a particular TNN model. It is one of many possible TNNs in the universe of space-time paradigms. A TNN based on biological neuron behavior is best developed in parallel with a simulation model. Hence, simulation is central to this work. Benchmarks, taken from the discipline of machine learning, provide simulator datasets. The overall modeling and implementation approach is illustrated in Figure 1.11.

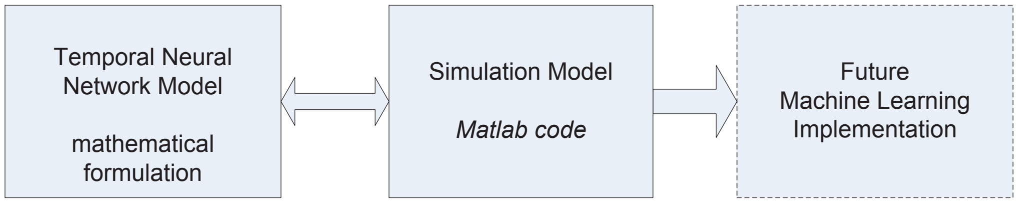

Temporal Neural Network Model
mathematical formulation
Simulation Model
Matlab code
Future Machine Learning Implementation

Figure 1.11: Overview of modeling and implementation methods.

A *Temporal Neural Network Model* is developed. The model specifies the way data is communicated and the way excitatory neurons and bulk inhibition behave. It also specifies the grouping of neurons into computational units that form the building blocks for scalable, hierarchical systems.

A *Simulation Model* is developed concurrently with the neural network model. The simulator implements algorithms, in high-level language software, that translate the neural network model into executable code.

Finally, the simulator is developed with an eye toward an eventual *Machine Learning Implementation*. For example, arithmetic precision during simulation is limited both for plausibility reasons and to assure feasibility of an eventual short precision fixed-point implementation. Hence, the simulation model may serve as a prototype for an eventual TNN-based machine learning implementation.

Chapter 2 is a theoretical discussion of space-time computing that provides perspective and guidance for constructing neocortex-like computing systems. A broad class of space-time computing networks is formally defined and are shown to be isomorphic to TNNs.

Chapter 3 is a biological overview that describes those features of the biological brain (the neocortex, specifically) that underlie the models developed in later chapters. It is not intended to be a summary or tutorial of what is a very broad topic; rather, it focuses on selected experimental results that provide guidance and support for the remainder of the book.

Chapter 4 begins with the experimental results from Chapter 3 and specifies an initial set of modeling assumptions that underlie the TNNs to be constructed later.

In Chapter 5, individual neuron modeling is discussed at some length. First is the development of a plausible spiking neuron model, in which each neuron operates within its own temporal frame of reference. Multiple synaptic paths connect the same pair of neurons. Because the individual paths exhibit a range of delays, they must each be included in the model. This is a critical first step toward a method of space-time computation.

Chapter 6 is a transition. It describes single excitatory neurons from a computational perspective. There are two levels of excitatory neuron development. At the lower level, individual synaptic paths are modeled. At the higher level, the multiple paths connecting two neurons are combined into a single, compound synapse. It is the higher-level compound synapse model that is used in the remainder of the book.

In Chapter 7, excitatory neurons are incorporated into columns, thereby forming a computational unit. Useful space-time computation takes place when multiple neurons in a column operate collectively. The modeling of inhibition is introduced in this chapter. The focus here, and in the remainder of this work, is on feedforward pattern clustering (unsupervised learning).

Chapter 8 describes the simulation model that is used for design space exploration. The structure of the simulator closely tracks the model definition given in the preceding chapters.

Chapter 9 describes the ongoing development of a prototype clustering TNN using the dataset from the widely used machine learning benchmark MNIST.

Chapter 10 summarizes and draws overall conclusions.

CHAPTER 2
Space-Time Computing

"Aspects of time are often said to be more 'abstract' than their spatial analogues because we can perceive the spatial, but can only imagine the temporal." [17]

Our overarching objective is to understand the types of computing paradigms employed in the neocortex, and then implement them with conventional digital technology. If there ever was a problem that calls for "thinking outside the box," this is it. Fortunately, this may be one of those rare situations where the boundaries of "the box" can actually be specified.

The box contains the ways that we ordinarily think about computing in space and time. When we reason about computation, we put most of the complexity in the spatial dimension and try to simplify the temporal dimension as much as possible. Consequently, we use only simple forms of abstract time, and these simplifying abstractions are used so universally that we seldom think about them. However, with space-time computing, time and temporal abstractions come to the forefront.

In this chapter, the argument is made that space-time paradigms may offer new ways of thinking about and designing computers with advanced cognitive capabilities. Before proceeding, some discussion of meta-architecture terms follows; these will be used throughout this chapter and the remainder of the book.

2.1 DEFINITION OF TERMS

- A *function* is a mapping from inputs to outputs, without regard to any implementation.
- A *model implementation* is a well-defined, clear, and unambiguous method for evaluating a function, i.e., a method for calculating an output for a given input. Sometimes, a model implementation is expressed in mathematical terms. An example of a model implementation is a formally defined Turing machine. Another example of a model implementation is the VHDL description of a microprocessor.
- A *physical implementation* (or *physical computer*) is a physically constructed computer system that evaluates a function. Normally, a physical implementation is based on a model implementation. An example physical implementation is a *finite-tape* Turing machine, sitting on a table, moving the physical tape to and fro (it's been done). Another example is a silicon microprocessor embedded in a smart phone. Physical time is naturally present in any physical implementation.