2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture

Space-Time Algebra: A Model for Neocortical Computation

James E. Smith
Department of Electrical and Computer Engineering
University of Wisconsin-Madison (Emeritus)
jes@ece.wisc.edu

***Abstract* — A proposed first step in replicating the computational methods used in the brain's neocortex is the development of a feedforward computing paradigm based on temporal relationships among inter-neuron voltage spikes. A "space-time" algebra captures the essential features of such a paradigm. The space-time algebra supports biologically plausible neural networks, as envisioned by theoretical neuroscientists. It also supports a generalization of previously proposed "race logic". A key feature of race logic is that it can be directly implemented with off-the-shelf CMOS digital circuits. This opens the possibility of designing brain-like neural networks in the neuroscience domain and implementing them directly in the CMOS digital circuit domain.**

Keywords-temporal neural networks, spiking neurons,space-time algebra, race logic

I. INTRODUCTION

For over a decade, a grand challenge posed to computer researchers has been to understand, and eventually replicate, the way the brain computes – "reverse-engineer the brain", so to speak [22][25][26] [40]. Despite its universally recognized importance, computer researchers in general, and computer architects in particular, have made little progress toward this goal. In fact, theoretical neuroscientists have assumed leadership in developing biologically plausible neural computing architectures.

The challenge needn't be addressed in strictly neuroscientific terms, however. In this paper, a new computing model, which includes an important class of neuroscientific models, is developed from the computer architecture perspective. Rather than being based on principles of binary logic, however, it is a radically different model based on temporal principles.

Ultimately, we would like to construct large scale cognitive systems capable of a broad range of brain-like activities and possessing many of its very desirable features: high efficiency, flexibility, and the ability to learn dynamically, quickly, and simultaneously with operation. Given this ambitious long-term goal, a proposed first step is to understand the basic computing paradigm(s) at the same level, say, as binary combinational logic. That is, to develop an algebra that encompasses feedforward neuron-like computation.

This is only a first step because feedforward networks alone are probably insufficient for engineering the large scale future systems we desire. Nevertheless, understanding and designing such feedforward networks is a milestone of critical importance. By analogy, when designing conventional computers, we first conceive of functions as combinational blocks, to which we add structured, clock-based synchronization.

This paper is directed at an important class of biologically-based neural network models being studied by theoretical neuroscientists. Biological neurons communicate via voltage pulses, or *spikes*. The particular spiking neural models of interest here convey and process information via precise spike timing relationships measured across multiple communication paths – in contrast to the classical approach of using spike rates measured on individual communication paths. This paper makes the following three contributions.

First, a proposed "space-time" algebra captures the essential features of the spiking neural networks we are targeting. The algebra is intended to model the passage of time among inter-operating spatial computing elements (e.g., neurons). Hence, time is modeled as an unbounded sequence of non-negative values, and all implemented functions must satisfy constraints that are consistent with the forward flow of time.

Second, the paper illustrates how spiking neuron models, as envisioned by neuroscience researchers, can be implemented using the primitives of the proposed space-time algebra. This construction and modeling approach has the look-and-feel of conventional computer design methods and is very different from the neuroscience approach of discretizing real-valued numerical models.

Third, potential applications of space-time algebra may be much broader than spiking neural networks. It is shown that space-time algebra also supports a generalization of race logic [31]. A key feature of race logic is that it can be directly implemented with off-the-shelf CMOS digital circuits. Digital circuits aren't used for processing logic values, however. Rather, they process temporal values: the times of signal transitions (edges). An important implication is that we may eventually be able to design cognitive systems in the spiking neural network domain, and then, by representing temporal events as level transitions rather than spikes, implement them directly using off-the-shelf CMOS.

II. BACKGROUND

Cognitive functions, the ones of interest here, are performed in the brain's neocortex – the deeply folded, 3 mm thick outer layer that is prominent in drawings and photographs of the brain's exterior. Computation in the neocortex is performed by a huge number of interconnected neurons, organized as a hierarchical layering of columns, each column containing many tens of neurons [39].

2575-713X/18/$31.00 ©2018 IEEE
DOI 10.1109/ISCA.2018.00033

This section briefly describes the modeling of individual neurons and then turns to models of neural computation. Next is a taxonomy of neural networks. In terms of the taxonomy, the networks of interest here are Temporal Neural Networks (TNNs) – networks where information is encoded and processed via precise spike timing relationships. At the end of the section is a summary of prior research in spiking neuron models and TNN architectures.

A. Neuron Modeling

According to the more-than-century-old neuron doctrine [56], the *neuron* is the fundamental element for structure and function in the brain's neocortex. A neuron is a specialized cell that consists of a *body* surrounded by a *membrane.*

Interconnected neurons communicate via voltage pulses or *spikes*. 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. A single neuron has thousands of synapses attached to its dendrites, connecting it to thousands of axons driven by upstream neurons.

Although it has long been commonly-held that precise spike timing relationships may somehow encode information and support neural computation [1][50], Maass [32] formalized such a spiking neuron model and analyzed its functional capabilities. He considered feedforward networks composed of model neurons illustrated in Fig. 1. This is essentially a version of the Spike Response Model 0 (SRM0) developed by Kistler et al. [30].

In the model neuron, input spikes produce output spikes in the following way: 1) Each input spike (x_i) is first delayed by some amount (δ_i). 2) A synaptic weight (w_i) determines the amplitude of a generated *response function*. Response functions model the change in the neuron's *body potential* due to ions flowing through a conceptual synaptic gate swung open by the incoming spike. The greater the synaptic weight, the wider open the gate swings, and the higher the response function's amplitude. 3) The individual response functions are *summed* (*integrated*), yielding the total body potential. 4) When, and if, the body potential exceeds a *threshold* value θ, an output spike (y) is produced. Fig. 2 illustrates and describes two example response functions.

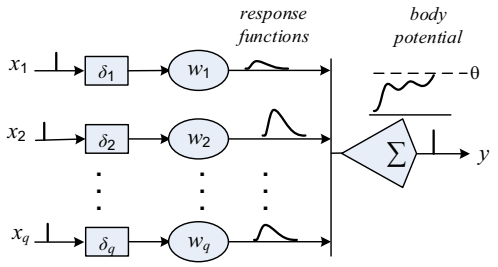

Figure 1. Spike Response neuron model. Input spikes pass through weighted synapses which produce response functions. Response functions from all inputs are summed, and an output spike is generated if/when the body potential crosses a threshold value.

Figure 2. Two response functions. a) Biologically-based biexponential response function that is the difference of two exponential decays. The faster decay models the decrease in synaptic conductance that follows the sudden increase caused by an input spike; the slower decay models membrane leakage. b) Piecewise linear approximation of biexponential response studied by Maass [32].

Nomenclature: researchers commonly use the general term "Leaky Integrate-and-Fire" (LIF) to characterize a rather broad class of neuron models [2][16][49][52]. The SRM0 model with response functions similar to those given in Fig. 2 belongs to the general class of LIF neurons.

The response functions illustrated in Fig. 2 exhibit the behavior of inputs being driven by *excitatory* neurons. Excitatory neurons are capable of precisely repeatable operation, and their axons can reach both near and far.

Inputs coming from *inhibitory* neurons, sometimes referred to as "interneurons", have response functions with a negative polarity, thereby reducing a receiving neuron's body potential. Inhibitory neurons tend to interoperate with virtually all other neurons in the same local volume and appear to function collectively, thereby applying a "blanket" of inhibition [27] that temporarily dampens all (or nearly all) nearby excitatory activity.

Synaptic weights help determine a given neuron's overall function. Through a training process, commonly occurring temporal spike patterns are learned and recorded as patterns of synaptic weights. Subsequently, when a spike pattern similar to a learned pattern is applied to the neuron's inputs, the synaptic weights all yield strong responses, and the neuron emits an early output spike. If an applied pattern is dissimilar to all learned patterns, the neuron emits a late spike or no spike at all, depending on how dissimilar it is.

A biologically-based model for synaptic weight training is Spike Timing Dependent Plasticity (STDP) [4][35][38]. The rationale and operation of STDP is typically summarized as follows. If the input spike from an upstream neuron *precedes* the output spike of the downstream neuron, then it contributes to the occurrence of the downstream neuron's spike, so its influence is *enhanced* in the future by increasing that input's synaptic weight. On the other hand, if the input spike occurs *after* the downstream neuron's output spike, it had nothing to do with generating the downstream neuron's spike so its influence is *diminished* in the future by decreasing its synaptic weight. Eventually, all the weights converge to values that collectively characterize input spike patterns. This process is described well in [37].

Low resolution communication and computation, as embodied in spike times and synaptic weights, are an essential aspect of the neuron models considered here. Hopfield [23] describes a plausible processing regime in which inhibitory gamma oscillations divide excitatory processing into intervals of approximately 5-20 msec. (See experimental results summarized by Fries et al. [15].) Other carefully-controlled experiments indicate that excitatory neuron spiking behavior

is reliably reproducible to within 1 msec [34]. Taken together, these results suggest a spike time precision of about 1 msec can be maintained over an interval of 5 - 20 msec. This yields roughly 2 to 4 bits of temporal resolution.

Synaptic weight resolution is closely coupled to temporal resolution, because synaptic weights help determine output spike times. Intuitively, given that spiking input and output temporal resolution are no more than 4-bits, there is little to be gained by having weights with resolutions that are much higher. Hopfield [23] makes an observation along these lines. Pfeil et al. [43] discuss weight discretization and resolution in the context of neuromorphic implementations and conclude that, for a class of plausible STDP models, 4 bits of resolution are sufficient for accurate modeling.

B. Neural Networks

Broadly speaking, there are two hypothesized methods for inter-neuron communication and processing: spike rates as measured on individual communication lines and precise spike-time relationships measured across multiple lines (for an example, the reader may look ahead to Fig. 5). Call these methods "rate coding" and "temporal coding".

Neural networks can be arranged in a simple taxonomy based on usage of rate coding and temporal coding (Fig. 3). The taxonomy divides neural networks into three categories.

1) Artificial Neural Networks (ANNs) originally sprang from the rate theory. In this taxonomy, ANNs are defined very broadly: the only restriction is that computed values are confined to a limited range (just as rates would be). By this definition, any of the conventional machine learning methods based on an inner product (*inputs* • *weights*) followed by an activation function is an ANN.

2) Rate-based Neural Networks (RNNs) implement communication and computation at the spike level, yet support a rate-based computing model. Examples include Spaun [14], spiking neuron implementations that rely on conventional ANNs for synaptic weight training [12][42], and memristor-based designs [45].

3) Time-based Neural Networks (TNNs) support a temporal-based computing model and implement communication and computation at the spike level. Examples are discussed at length in Section II.C.

Strong experimentally-based arguments support a temporal theory. See [50] for an early, and frequently-cited, evidence-supported argument. Later work [54] summarizes further supporting evidence. Brette [8] provides an enlightening discussion of the philosophy behind both the rate and temporal theories; he concludes: "... the rate-based view appears as an *ad hoc* methodological postulate, one that is practical but with virtually no empirical or theoretical support." In this paper, a temporal-based approach is taken; the focus is on TNNs.

Maass [32] applied the term "Spiking Neuron Network" (SNN) to a class of networks that use spike times to encode information (as TNNs do). However, RNNs also use spiking neurons in their implementation. Consequently, over time, both TNNs and RNNs have become lumped together as "SNNs". This is unfortunate because they support two completely different computing models: rate-based and temporal-based.

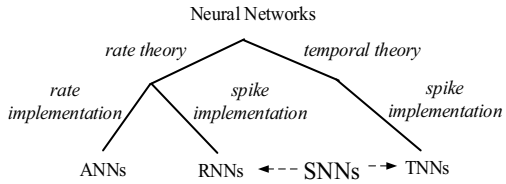

Figure 3. Neural network taxonomy based on theory and implementation method. The class of networks in this paper, TNNs, use a temporal theory and spike implementation.

An informal test to distinguish RNNs and TNNs: Do interconnection lines carry at most one spike during a given feedforward computation? If so, it is most likely a TNN. Or, must each line carry at least two spikes (the minimum to establish a rate) during a given feedforward computation? If so, it is most likely an RNN.

C. Computing with Temporal Neural Networks

A number of research efforts are aimed at constructing TNNs that perform cognitive computational tasks. These efforts target pattern clustering and classification problems, similar to the ones addressed by conventional machine learning.

Most proposed TNNs employ a *simple synapse model* as illustrated in Fig. 1. Generally speaking these TNNs employ integrate-and-fire neurons, training is via biologically plausible STDP, and inhibition is via a bulk winner-take-all (WTA) strategy, where the first spike produced by a group of neurons inhibits all the other spikes; i.e., only the "winner" survives.

The STDP mechanism for model excitatory neurons is described by Guyonneau et al. [20][21]. That work also clearly articulates and demonstrates the basic principles of computation using temporally coded spiking neurons.

Subsequent work [36][37][48][51][53] yields neural architectures that appear superficially similar to hierarchical, layered ANNs. However, compute nodes implement a temporal spiking paradigm, rather than an inner-product-and-squash paradigm. In much of this work, simple, non-leaky excitatory neuron models are used. Inhibition is via WTA. SpikeNet [11] is a large scale network simulator based on these principles.

To illustrate the scale of architectures being developed by neuroscience researchers, Fig. 4 is a relatively simple example due to Bichler et al. [5], which uses an LIF neuron model coupled with a special form of STDP to track car trajectories on a freeway. Input sensors use Address-Event Representation (AER), an efficient way of transmitting sparse spike timing information [13].

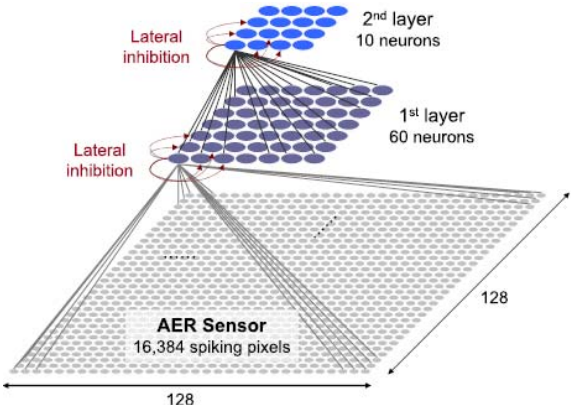

Figure 4. TNN studied by Bichler et al. [5]. Lateral inhibition is implemented as a WTA function.

Recent spiking neural architectures, which arguably define the state-of-the art for TNN models, are given in Kheradpisheh et al. [28][29]. This work pushes in the direction of multiple excitatory neuron layers (however, by deep ANN standards, the implemented networks remain relatively shallow.) Related work by Wysoski et al. [55] uses an integrate-and-fire neuron model to construct an ambitious hierarchical bimodal system that combines both auditory and vision pathways.

In 1995, Hopfield [23] observed that multiple synaptic paths connecting the same two neurons provide a powerful means for encoding temporal information. Call this a *compound synapse model*. Hopfield suggests ways compound synapses might be used for achieving behavior roughly similar to classical radial basis functions (RBFs)[9]. Gerstner et al. [17] make similar suggestions.

RBF-like functionality based on multipath connections was later explored by Natschläger and Ruf [41]. The main idea behind RBF neurons is that compound synaptic connections are similar to tapped delay lines. Training determines a pattern of selected delays. For training, Natschläger and Ruf use a version of STDP which stretches biological plausibility perhaps, but retains the important localized-training property. Bohte et al. [6] extend this work by refining the training method, extending it to multiple layers, and identifying multiple clusters. In a later paper [7], Bohte et al. take training in a different direction by proposing error back-propagation, a global training method similar to those used in conventional artificial neural networks. This method has been further enhanced [46].

The *tempotron model*, proposed by Gütig and Sompolinsky [18], is an SRM0 model with biexponential response functions. Its *tempotron training rule* uses supervised, yet localized, STDP-based training – labels are a primary input to the training process. Gütig et al. [19] apply the tempotron to model the retina. Zhao et al. [57] use tempotron neurons to construct a system which combines the AER signaling convention, with a TNN classifier. It uses what is essentially WTA inhibition and consists of a single layer of excitatory neurons, with one neuron assigned to each class.

Liquid State Machines [33][44] are based on the same principles as TNNs: temporal encoding and spiking neuron models. However, they contain feedback established via pseudo-random interconnection patterns. Although they are not feedforward TNNs, the theory in this paper may potentially be extended to include them.

III. Space-Time Networks

In this paper, we are interested in computing paradigms implemented as a feedforward network of functional blocks. Communication and computation is based on the timing of temporal events. In the specific context of TNNs, the temporal events are spikes transmitted between neurons. Hence, the following discussion is phrased in terms of voltage spikes, although other ways of communicating via temporal events can be used (Section V).

A. Communication

In Fig. 5, a vector of information is encoded as a *volley* of discretely-timed spikes. Spikes are shown on timelines as dark vertical bars. In this example, values are simply encoded as times relative to the time of the first spike in the volley (t_{min}). The first spike encodes the value 0, and successive spikes encode values relative to 0. The symbol "∞" denotes the case where there is no spike on a given line. In the example, the encoded vector is [0, 3, ∞, 1].

This is an energy-efficient communication method. If time can be resolved to n bits (2^n time values), a vector can be communicated with an efficiency of slightly less than one spike per n bits of communicated information. (Slightly less because one of the lines always carries a value of 0.) As n grows large, energy-efficiency improves proportionately. Sparse codings, where most lines carry no spike, further improve energy efficiency.

Counteracting this effect, as n is increased, the total time to send a message grows exponentially (2^n). Because of this exponential time growth, the method appears practical only for very low resolution data (3 or 4 bits), corresponding to communication times of 8 to 16 time intervals per message (volley). But, very low resolution data is okay. As described earlier, the biological neocortex operates on low resolution data.

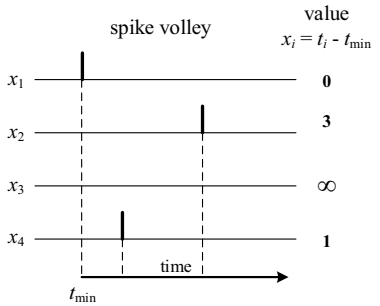

Figure 5. A spike volley consists of spikes occurring at discrete time intervals.

B. Computation Overview

Fig. 7 is a block diagram of a temporal computing network. Input values are first encoded as spike volleys. Computation then proceeds as a single wave of spikes sweeps forward (left-to-right) through the network. Each line carries at most one spike during a given computation.

Each functional block (e.g., a neuron) is initially quiescent. It begins computing at the time the first spike of a volley arrives, and it produces an output spike at a time that depends solely on observed input spikes (or it may produce no output spike). The only information a functional block receives is input spike times viewed from its local frame of reference; there is no other coordinating or synchronizing mechanism.

Essentially all the proposed TNN models (surveyed earlier in Section II.C) are feedforward, spike-based computing networks as just described. Consequently, the formal model laid out in the next section is targeted at such TNN models; however, as will be discussed later, the model encompasses much more than conventional TNNs.

C. Space-Time Computing Networks

Although feedforward computing networks have multiple input signals and multiple output signals, discussion here assumes a single output without losing generality. We begin with the defining features of space-time functions.

Spikes are assigned values which may be interpreted as times in a discretized timeline. In order to model points in time, define the set N_0^∞ to consist of 0, the natural numbers, and the special element "∞", which models the situation where there is no spike on a given communication line. The symbol "∞" has defined properties typically associated with infinity. That is: $\infty > n$ and $\infty + n = \infty$ for all $n \in N_0$.

Definition: A function $z = F(x_1...x_q)$, $x_{1...}x_q$, $z \in N_0^\infty$, is a *Space-Time Function* if it satisfies the following properties.
1) *computability:* F implements a computable total function.
2) *causality*: For all $x_j > z$, $F(x_1...,x_j,...x_q) = F(x_1...,\infty,...x_q)$, and if $z \neq \infty$, then $z \geq x_{\min}$.
3) *invariance*: $F(x_1 + 1, ..., x_q + 1) = F(x_1,..x_q) + 1$.

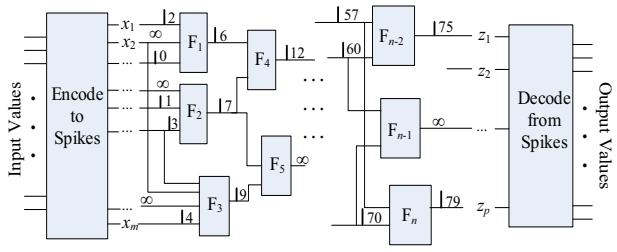

Figure 7. A feedforward computing network that operates on values encoded as spike timing relationships. Due to causality, spike times are monotonic increasing from inputs to outputs. Computed results emerge at the network's output, also in the form of spike timing relationships. These may be decoded into output values as shown here, or they may be passed on to other networks in their encoded form.

The three properties are very general. The first requires that the function be physically implementable and produce an output for all inputs. The other two properties are consistent with the uniform passage of time. *Causality*: if we apply a spike-based interpretation, then an output spike can only be affected by input spikes that occur at the same time or earlier. Furthermore, there are no spontaneous output spikes. *Invariance*: if all input spike times uniformly shift by unit time, then the output spike time does as well. Invariance naturally extends to any constant number of time shifts.

Definition: A *Space-Time Computing Network* is a feedforward interconnection of space-time functional blocks.

Lemma 1: All space-time computing networks implement space-time functions.

Outline of proof: Begin with a topological sort of the directed graph implied by the feedforward composition of functional blocks. Then, prove by induction on the sequence of blocks in the topological sort.

As a consequence of Lemma 1, if we start with a basic set of space-time functions, spiking neurons, for example, then any TNN constructed with these basic functions implements a space-time function. We are not required to use spiking neuron models as primitives, however. Other, even simpler, primitives may also be used for constructing space-time computing networks, as the following example illustrates.

Example. Three primitive functional blocks that satisfy causality and invariance are shown in Fig. 6a. The *inc*rement function (+1) emits an output spike 1 time unit after an input spike. This can be generalized to adding any constant c by stringing together c *inc*s. The *min* function (∧) emits an output spike at the same time as the first-arriving input spike. The *lt* function (≺) emits an output spike at the same time as input spike *a* iff *a* arrives earlier than input spike *b*. Otherwise, no output spike is emitted. Fig. 6b is a small example network with spike times shown.

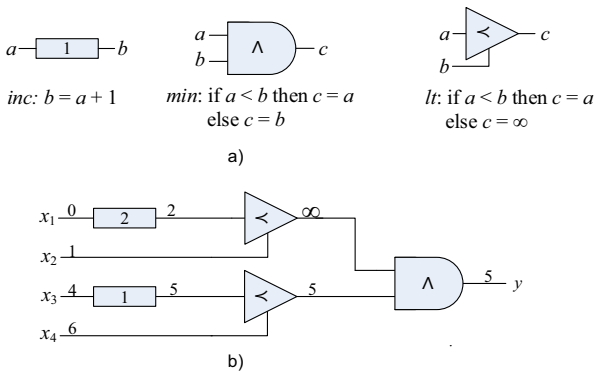

Figure 6. a) Three primitive functional blocks that satisfy causality and invariance. b) A small example network composed of these primitive blocks.

D. Formal Space-Time Algebra

The *space-time* (*s-t*) *algebra* provides a mathematical structure for modeling the relationships between events occurring in linear, discretized time.

Definition: The *s-t algebra* is a bounded distributive lattice $\mathbb{S} = (N_0^\infty, \wedge, \vee, 0, \infty)$. $\mathbb{S}$ consists of a bottom element 0, a top element ∞, and the natural numbers. $\mathbb{S}$ is well-ordered and is closed under addition.

$\mathbb{S}$ is not complemented. However, by definition, $\wedge$ and $\vee$ are distributive, associative, commutative, idempotent, and satisfy the absorption laws. For constructing *s-t* computing networks, we are interested in primitive functions defined over the *s-t* algebra. For this purpose, we will use the *max* ($\vee$), *min* ($\wedge$), *lt* ($\prec$), and *inc* (+1) functions.

E. Bounded Space-Time Functions

Definition: A *Bounded Space-Time Function* $z = F(x_1...x_q)$ is a space-time function further restricted to a bounded amount of history. That is, there is a finite k, such that for any $x_j < x_{max} - k$, $F(x_1..,x_j,..x_q) = F(x_1..,\infty,..x_q)$, where x_{max} is the largest non-∞ element of $\{x_1..,x_j,..x_q\}$.

When combined with causality, the bounded history restriction effectively says that a functional block can only look at a finite (size k) window of past input history to determine its output value. This guarantees that a practical neuron implementation contains only a finite amount of state. Considering the low resolution of biologically plausible neurons, a realistic neuron model needs to track a spike time window 8 to 16 time units wide, at most.

F. Normalized Function Tables

One way of specifying *bounded s-t* functions is to use a function table, analogous to a Boolean truth table. Fig. 7 is a small example. The table shown is in *normal form*. A table is *normalized* if 1) at least one input in each row is a 0 and 2) the output is not ∞. All normalized inputs that yield a non-∞ output have an entry in the table. Observe that it follows from the bounded history requirement that a normalized function table must have a finite number of entries.

An important observation is that although a normalized function table contains only a finite number of rows, it specifies a total function over the infinite set N_0^∞ due to the invariance property. Given a normalized table, to compute the output for an arbitrary unnormalized input vector, one should first normalize the input vector by subtracting x_{min} from each element. If the normalized vector is in the table, then x_{min} should be added to the corresponding table entry to yield the output value. If the normalized entry is not in the table, then by definition the output is ∞.

x_1	x_2	x_3	y
0	1	2	3
1	0	∞	2
2	2	0	2

Figure 7. A normalized function table having three inputs (x_i) and one output (y).

For the example in Fig. 7, if given the unnormalized input [3, 4, 5], we first normalize by subtracting the smallest value (3 in this case) to yield [0, 1, 2]. The table entry at [0, 1, 2] is 3. Therefore, due to invariance, the function's value at [3, 4, 5] is 6. (We subtracted 3 to get the normalized inputs, so 3 is added to get the final value.)

G. Functional Completeness

From this point forward, discussion is restricted to *bounded s-t* functions. Consideration of functional completeness begins with a lemma, followed by the main completeness theorem.

Lemma 2: The *max* function can be implemented using only *min* and *lt* functions.

Proof: by construction (Fig. 8). Three cases are: case 1: $a < b$; case 2: $a = b$; case 3: $a > b$. At each point in the network, the value for each of the three cases is given in parenthesis. At the network output c – case 1: $c = b$ if $a < b$. case 2: $c = a = b$ if $a = b$. case 3: $c = a$ if $a > b$. Hence, $c = max(a, b)$. (Thanks to Cristóbal Camarero)

Theorem 1: The *min*, *increment*, and *lt* primitives are functionally complete for the set of bounded *s-t* functions.

Proof summary: The proof is by construction of a minterm canonical form. Space restrictions do not allow a full proof; a summary and illustrative example are given. For convenience, the *max* function is used (Lemma 2).

Row j of the bounded function table corresponds to $minterm_j$. A minterm is implemented via a *max* function and a *min* function whose outputs are combined by an *lt* function (refer to Fig. 9). In the implementation of $minterm_j$, every input x_i is incremented by constant δ_{ij} , where $\delta_{ij} = y_j - x_i$. For example $minterm_1$, inputs $[x_1, x_2, x_3] = [0, 1, 2]$ and the output $y_1 = 3$. Consequently, the increments δ_{i1} are [3, 2, 1].

The δ_{ij} are defined so that only the one input pattern exactly matching the corresponding $minterm_j$ yields a non-∞ *lt* result: the value y_j. For that input pattern only, the *max* function yields a value less than the value computed by the corresponding *min* function, thereby allowing the row's output value y_j to pass through.

The case where one of the minterm inputs $x_i = \infty$ is illustrated by $minterm_2$ in Fig. 9. In this example, x_3 is fed directly to the *min* function. If a value applied to x_3 is greater

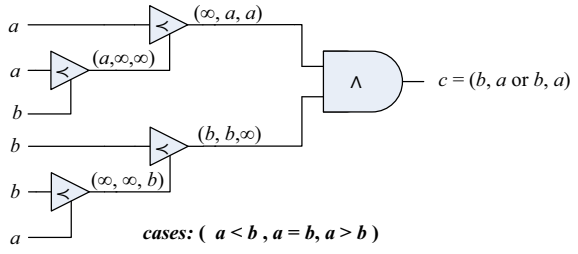

Figure 8. Proof by construction that the *max* function can be implemented using only *min* and *lt* primitives. Values at points in the network are divided into three cases.

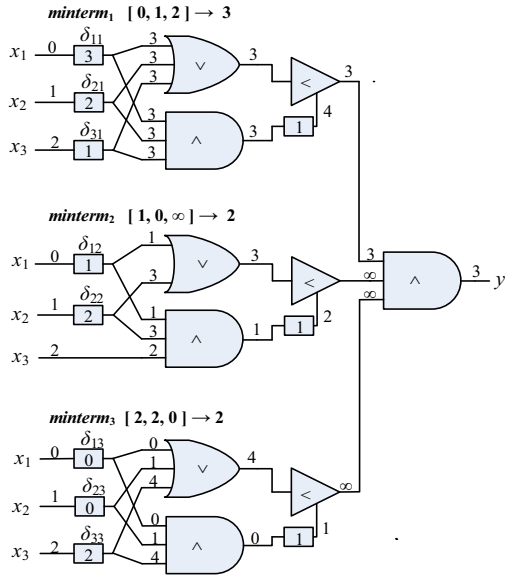

Figure 9. Example of minterm canonical form for function table given in Fig. 7. If input [0, 1, 2], corresponding to $minterm_1$ is applied, the resulting network values are shown.

than the minterm's output (2 in this example), it has no effect. If it is less than or equal to the minterm's output, it forces the minterm to ∞ (there is no match).

The final *min* function produces output *y* by combining the *lt* outputs from all the $minterm_j$, corresponding to all table rows *j*. In Fig. 9, the applied input is [0, 1, 2], and the corresponding values are shown. All minterms except [0, 1, 2] evaluate to ∞, and [0, 1, 2] evaluates to 3.

IV. CONSTRUCTING TNNS

Theorem 1 states that all bounded *s-t* functions can be implemented with *min*/*lt*/*inc* primitives. The TNNs studied by theoretical neuroscientists implement bounded *s-t* functions. So, an immediate corollary is that all the TNNs can be implemented using only the *s-t* primitives. To emphasize this point, in this section commonly-used TNN components are constructed using only the primitives. Included are implementations of SRM0 neurons, WTA inhibition, and programmable synaptic weights.

These implementations serve two purposes: to illustrate *s-t* computation as a general proposition and to provide a basis for TNN simulation and/or direct hardware implementation. However, it should be kept in mind that there is no requirement that these specific implementations be used. Any spiking neuron functions that satisfy the *s-t* constraints can be used for constructing TNNs. Of course, this includes the models used in the neuroscience literature.

A. SRM0 Neurons

1) Bitonic Sorting Networks

Generally speaking, *sort* is an important function on its own, and, as is shown here, sort may be used as a building block network for constructing *s-t* neuron models.

First observe that sort is causal and invariant. When numbers are sorted from smallest to largest, the position of any given number in the sorted list depends only on the locations of smaller or equal numbers. Any larger numbers are irrelevant in determining its location in the sorted list. Sort is invariant because adding a constant to all the inputs does not change the sorted order, and the sorted outputs have the same added constant.

A bitonic sorting network [3] is constructed using two-input, two-output compare elements, each consisting of a *min* function and a *max* function (see Fig. 10). Because it uses only *min* and *max* elements, which are both causal and invariant, the entire sorting network must be causal and invariant (Lemma 1).

2) Modeling Response Functions

A *response function* R(*t*) maps the non-negative integers onto the integers. Informally, the only constraints are 1) after finite time (t_{max}) the response function reaches a fixed value c and never changes, and 2) R ranges between fixed, finite maxima and minima, r_{min} and r_{max} . These very broadly defined response functions include discretized versions of all proposed response functions of which the author is aware.

As an example, Fig. 11 (left) is a discretized low resolution version of the biexponential response function. For this response function, $t_{max} = 12$, c = 0, $r_{min} = 0$, $r_{max} = 5$.

R(*t*) may be defined to follow a positive curve (for excitation) or negative curve (for inhibition). R(*t*) serves as the starting point for constructing an *s-t* network that takes a value (spike time) as input and generates a set of incremented (delayed) values as outputs. These output values capture the specified behavior of the response function in *s-t* form, as described in the following paragraphs.

Although a neuron has many inputs, first consider the response function for a single input, *x*. Due to its discrete nature, the response function's amplitude at any given time can be expressed as an integer number of *amplitude units* (either positive or negative). By sequencing from $t = 0$ to $t = t_{max}$, the response function can be represented as a sequence of *up steps* and *down steps*. Each "step" is a single discrete amplitude unit.

Accordingly, to construct the response function for input *x*, the input *x* is fanned out, with increment values placed on

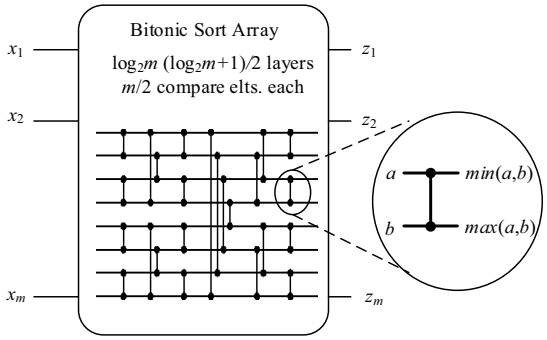

Figure 10. A bitonic sorting network consists of interconnected min/max comparator elements.

the fanned-out lines. The fanouts are divided into *up* fanouts and *down* fanouts. For each t, $1 \le t \le t_{max}$, define $n = R(t) - R(t\text{-}1)$; for the case $t = 0$, define $n = R(0)$. If n is positive, then fan x out n times to the *up* fanout network; each of the fanouts is assigned an associated *inc* value of t. If n is negative, then fanout n times to the *down* fanout network, each of the fanouts is assigned an associated *inc* value t.

Fig. 11 (right) illustrates the fanout/increment network for a biexponential response function. The function takes two *up* steps at $t = 1$, two more *up* steps at $t = 2$, a single *up* step at $t = 5$, then a series of *down* steps at $t = 5, 7, 8, 10, 12$.

3) Modeling Integration and Firing

Given all the input response functions implemented as *s-t* fanout networks, the remainder of the SRM0 neuron model is constructed as shown in Fig. 12.

The inputs to two sort networks are the fanned-out up/down increment values (denoted generically in the figure as "u" and "d") for all x_i. Collectively, these specify the response behavior for all the primary inputs. The *up* steps are sent to one sort network, and *down* steps are sent to the other. After sorting, the two sets of sort network outputs are combined via a series of *lt* blocks. The *lt* blocks determine whether the time of the θ + *i*th *up* step occurs before the *i*th *down* step. If so, the non-∞ value of the θ + *i*th *up* step is the input to the final *min* function. Otherwise, the input value to the final *min* function is ∞. The *min* function's output is therefore the first time the number of *up* steps exceeds the number of *down* steps by the amount θ, i.e., first time the threshold is crossed. If it is never crossed, the output is ∞ .

As described above, a bitonic sort network can be constructed using only interconnected *min* and *max* functions, so the neuron model as a whole is constructed using only *s-t* primitives.

B. Implementing Synaptic Weights

In a typical neuron model, synaptic weights determine response functions, which determine an individual neuron's function, and, consequently, the overall neural network's function. In TNNs, synaptic weights are typically established via a process where inputs from a training set are applied and weights are adjusted to reflect patterns inherent in the training inputs. Training methods used in TNNs may either be biologically plausible, based on spike timing relationships (e.g., STDP[28][37]), or they may be transferred from conventional machine learning (e.g., error back propagation[7]).

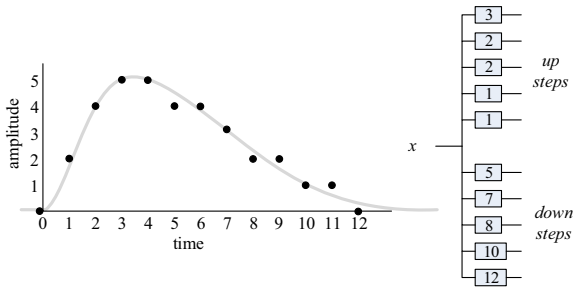

Figure 11. Biexponential response function and corresponding s-t fanout network.

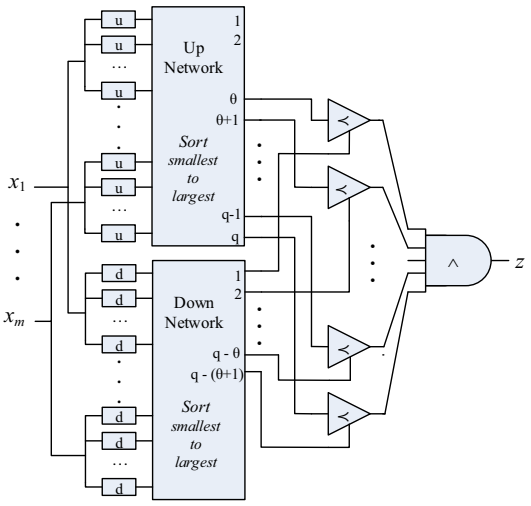

Figure 12. SRM0 implementation using s-t primitives.

After training, the learned synaptic weights become part of a neuron's function definition. In a sense, an untrained network is configured, or programmed, depending on the weight settings.

Considering potential applications beyond TNNs, programmability or configurability may be desirable in more general *s-t* networks. Hence, following is a primitive mechanism that supports general configurability as well as implementing synaptic weights in TNNs. The primitive configuration/programming mechanism is conceptually an enable/disable switch. In Fig. 13, a *micro-weight* μ input to an *lt* function is configured to either 0 or ∞ prior to a *s-t* computation. If input $\mu = 0$, then the output z of its associated *lt* is ∞, regardless of the input x. If input $\mu = \infty$, then the x input value passes through to the output of the *lt*.

In the context of spiking neuron models with trained synaptic weights, response functions can be implemented with micro-weight-enabled fanout/increment networks. Say we have a spiking neuron with n possible response functions which are identified via n different synaptic weights. In the low resolution models envisaged here, n may be 8 or fewer. We can then design an input fanout/increment network controlled via micro-weights so that each synaptic weight maps to specific micro-weight settings which, in turn, yield the desired response function. Determining the micro-weight set-

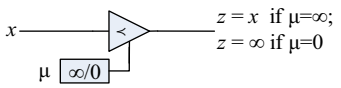

Figure 13. A micro-weight μ either enables input x (by setting $\mu = \infty$) or it disables input x (by setting $\mu = 0$).

tings may be done via a training process, or, in general, the settings may be determined by any other means.

For example, the network shown in Fig. 14 implements a set of four response functions, having four different amplitudes corresponding to a range of synaptic weights. Synaptic weights are set via micro-weights μ_1 through μ_4. To set the synaptic weight to 3, for example, μ_1, μ_2, $\mu_3 = \infty$, and $\mu_4 = 0$.

C. Inhibition: Winner-Take-All Networks

The models discussed thus far are used primarily for excitatory neurons. Inhibitory neurons are typically modeled differently: as tightly-connected networks that operate *en masse* [27]. In the neuroscience literature, inhibitory networks often implement a form of WTA lateral inhibition. In the case of TNNs, the "winners" are the first spikes in a volley, so winner-take-all inhibits all but the first spikes. In general, what is meant by "first" may be parameterized. It may the first *k* spikes, or the spikes that appear within some time window beginning with the first, or some combination.

Fig. 15 is the implementation of a simple 1-WTA network where only the spikes occurring at relative time 0, are allowed to pass; all the others are inhibited. The *min* gate finds the time of the first spike(s), and that time, delayed by 1 time unit, inhibits all the others. In general, a designer can widen the time window of uninhibited spikes by increasing the delay value to τ, thereby implementing τ -WTA.

D. Section Summary

In this section, 1) SRM0 neurons with arbitrary response functions were constructed from the *s-t* primitives. 2) Synaptic weights were implemented through micro-weight networks. 3) An implementation of WTA lateral inhibition was given. Together, these three basic components underpin essentially all TNNs described in the neuroscience literature.

V. Generalized Race Logic

TNNs implement the *s-t* algebra via spikes that mark points in time. Race Logic (RL) [31] is an alternative *s-t* implementation that uses transitions in voltage levels (edges). RL is implemented with conventional digital CMOS and uses the times of logic level transitions for encoding and processing information. RL, as proposed by Madhavan et al. [31], is a feedforward interconnection of logical AND gates, OR gates, and delay elements. Information is encoded as the times of logic level transitions. Among other things, Madhavan et al. illustrate an RL method for finding the shortest path in an acyclic, weighted graph.

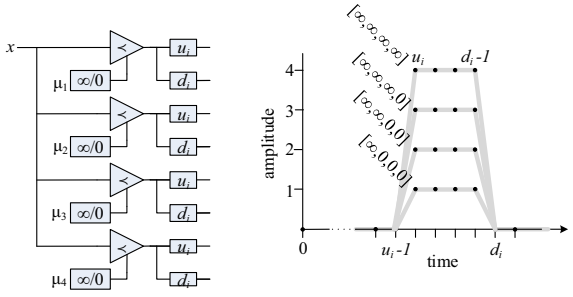

Figure 14. Modeling a range of synaptic weights. In this example, the range is 0 to 4. Synaptic weights are determined via a vector of micro-weights [μ_1, μ_2, μ_3, μ_4].

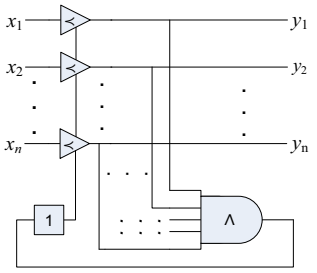

Figure 15. Winner-take-all network. Only the first spikes pass through uninhibited.

In the next section, it is shown that generalized race logic (GRL) yields a complete implementation of *s-t* algebra. The implication is that the functionality of TNNs can be implemented directly with off-the-shelf CMOS circuits.

A. GRL Primitives Defined

Communication in GRL is via 1→ 0 transitions in logic levels. The *s-t* primitives, implemented with conventional logic gates, are shown in Fig. 16. Note that the *lt* implementation contains a simple latch at its output. The latch assures that once the output transitions to 0, it never transitions back to 1 if/when the *lt* input (*b* in this case) transitions at some time after the *a* input transitions. The *reset* input initializes the *lt* latch to 1 prior to each feedforward computation.

B. Modeling the Flow of Time

Following the method of Madhavan et al., GRL delays (*s-t increment* functions) are implemented by inserting series of clocked flipflops (forming shift registers) so that a single clock cycle demarks idealized *unit time*.

Note that the clock signal is not part of the neuron model *per se*; rather, it is used as an ancillary mechanism for implementing precise unit delays associated with the *increment* function. Furthermore, to faithfully model the zero-delay nature of the gate primitives (*min*, *max*, *lt*), the implemented clock cycle may be made long enough to cover all inter-shift-register wire and gate delays.

A disadvantage of shift register delay elements is that

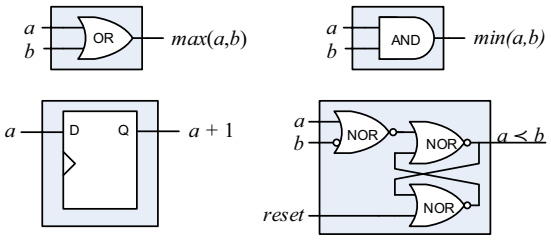

Figure 16. GRL implementations of four *s-t* algebra primitives.

energy consumption may increase significantly due to the clocked shift registers. Further research is required to quantify, and perhaps minimize, this effect.

One might also consider a more direct form of GRL that relies on implementing precise physical delays, say in the wires or intentionally inserted non-clocked delay elements. This approach would have to account for individual gate latencies as well.

Conjecture: Regardless of the way delays are physically implemented, an essential aspect of the *s-t* paradigm is that the passage of physical time (whether measured with a shift register or any other way) must be an integral part of an implementation in order to achieve the full energy efficiency and performance benefits.

C. Direct TNN Implementations

A *direct* TNN implementation, whether in silicon or in biological neurons, literally uses transient temporal events for communication and computation. This is essentially a unary encoding method. In contrast, an *indirect* implementation, as one might use in a simulator, may encode times as binary numbers. This is essentially conversion of a direct unary representation to an indirect binary representation. As noted earlier, a direct unary implementation is only practical for low resolution data. An indirect implementation is practical for much higher resolution data, if one is so inclined.

Neuromorphic neural network implementations [24] are direct methods based on silicon circuits that communicate via voltage spikes, literally. The motivation is that such neuromorphic implementations may achieve some of the same efficiencies as biological neurons. As proposed in this section, however, spiking silicon circuits are not the only direct method for implementing *s-t* functions (and TNNs): GRL is another. The only difference is that GRL uses edges rather than spikes. And, more importantly, GRL can be implemented with off-the-shelf digital circuits. In effect, one *may* be able achieve the advantages of a direct neuromorphic implementation without the need for specialized silicon circuits. It remains a topic for future research to quantify the merits of spike-based vs. edge-based neuromorphic circuits.

VI. Concluding Remarks

In this paper, a simple computing model and algebra are proposed. The space-time model includes all TNNs, including the biological one that we all possess – assuming the temporal coding hypothesis! Its simplicity is evident in that *min*/*inc*/*lt* are sufficient for implementing all the *s-t* functions. Although simple, this model runs counter to the way we normally think about computing in the following ways.

1) The *s-t* model deals with time in a completely different way than we normally do. When we construct computers, we remove the passage of physical time from the computing model. Conventional synchronous networks and delay-independent asynchronous networks are two ways of doing this. In contrast, with *s-t* computing, the passage of time is an essential part of the model. In effect, the time it takes to communicate a value *is* the value, and the time it takes to compute a value *is* the value.

Evolution has shown itself to be a very clever engineer across a broad range of biological systems. And, it seems compelling that a clever engineer might use the flow of physical time as a resource because it possesses some ultimate engineering advantages: the flow of physical time is free (it happens whether we like it or not), it requires no space, and it consumes no energy.

2) We humans appear to have a strong affinity to addition and multiplication as primitives for thinking about and describing almost any algorithm (including conventional machine learning algorithms). However, neither of these preferred arithmetic primitives is invariant; $(a+1) + (b+1) \neq (a+b+1)$ and $(a+1)*(b+1) \neq (a*b+1)$

It is the author's experience that reasoning about *s-t* functions does not come naturally. Look no further than the non-obvious proof of Lemma 2. Logical reasoning about space seems to be much easier for us than reasoning about time [10]. It may take some effort to develop an intuition that is useful for guiding *s-t* design.

3) The *s-t* primitives are incomplete with respect to all multi-valued functions. They are complete only with respect to *s-t* functions. In contrast, when we develop a discrete algebra, we seem to prefer to make it complete with respect to all possible functions, not just a proper subset of possible functions (as in the *s-t* case).

Observe that in most algebras, some unary operation akin to inversion, complementation, or negation is present. In an *s-t* algebra, however, such operations are tantamount to time flowing backwards. Hence, such operations are not present in the *s-t* algebra, and, consequently, the algebra is not complete for all functions.

4) Space-time computing networks are only practical for very low resolution direct implementations.

Contrast this with the development of digital computer systems. The numerical operations (addition and multiplication) are ubiquitous, and there has always been a push toward supporting higher resolution data; e.g. from 32-bit to 64-bit floating point. If we were forced to use 3-bit or 4-bit data values (always), it would upend most of the algorithms currently in use.

There are two important, fundamental aspects of *s-t* computing that have only been hinted at in this paper.

1) *Energy Efficiency*: It is conjectured that direct *s-t* implementations possess intrinsically high energy efficiency. When GRL is implemented in CMOS, transistors undergo either a single switch or none at all (however, they must be reset prior to the next computation). Furthermore, if one uses sparse spike codings, many of the signals undergo *zero* transitions during any given computation. This minimal-transition property appears to be intrinsic to direct implementations of *s-t* functions, and the minimal-transition property strongly suggests high energy efficiency.

2) *Emergence:* It is conjectured that the uniform passage of global time is a key element of the highly emergent behavior present in TNNs using STDP-trained neurons [28][29]. All training and evaluation is done locally at each neuron. Yet, large assemblies of neurons work collectively

to compute coherent global results. The passage of global time may serve as a wire-less, energy-less synchronization mechanism. Causality and invariance ensure that the uniform passage of global time is properly modeled.

Whether or not the neocortex operates according to *s-t* computing principles is yet to be definitively determined. However, if the neocortex does operate according to *s-t* principles (and a significant group of neuroscientists believe so), then developing *s-t* computing methods, although challenging, may be a significant step toward eventually "reverse-engineering the brain".

VII. Acknowledgments

What started out as an interesting pastime has turned into a fairly intense multi-year project, and I thank my wife Raquel for her patience and support. A number of people have given advice and valuable feedback: Mario Nemirovsky, Ravi Nair, Joel Emer, Tim Sherwood, Advait Madhavan, Abhishek Bhattacharjee, Cristóbal Camarero, Shlomo Weiss, Ido Guy, Quinn Jacobson.

VIII. References

[1] Abeles, M., Corticonics: Neural Circuits of the Cerebral Cortex, Cambridge University Press, NewYork, 1991.

[2] Arthur, J.V., et al., "Building block of a programmable neuromorphic substrate: A digital neurosynaptic core," 2012 International Joint Conference on Neural Networks, pp. 1-8, 2012.

[3] Batcher, K. E., "Sorting networks and their applications," Proceedings of the Spring Joint Computer Conference, pp. 307-314, 1968.

[4] Bi, G.-Q., and M.-M. Poo, "Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type," The Journal of Neuroscience 18, no. 24, pp. 10464-10472, 1998.

[5] Bichler, O., D. Querlioz, S. J. Thorpe, J.-P. Bourgoin, and C. Gamrat, "Extraction of temporally correlated features from dynamic vision sensors with spike-timing-dependent plasticity," Neural Networks 32, pp. 339-348, 2012.

[6] Bohte, S. M., H. La Poutré, and J. N. Kok, "Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer RBF networks," IEEE Transactions on Neural Networks, 13, no. 2, pp. 426-435, 2002.

[7] Bohte, S. M., J. N. Kok, and H. La Poutré, "Error-backpropagation in temporally encoded networks of spiking neurons," Neurocomputing 48, no. 1 pp.17-37, 2002.

[8] Brette, R., "Philosophy of the spike: rate-based vs. spike-based theories of the brain," Frontiers in Systems Neuroscience 9, 2015.

[9] Broomhead, D. S., and D. Lowe, "Radial basis functions, multi-variable functional interpolation and adaptive networks," No. RSRE-MEMO-4148, Royal Signals and Radar Test Establishment, Malvern, United Kingdon, 1988.

[10] Casasanto, D., and L. Boroditsky, "Time in the mind: Using space to think about time," Cognition 106, no. 2, pp. 579-593, 2008.

[11] Delorme, A., J. Gautrais, R. van Rullen, and S. Thorpe, "SpikeNet: A simulator for modeling large networks of integrate and fire neurons," Neurocomputing, 26-27, pp. 989-996, 1999.

[12] Diehl, P. U., et al., "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," 2015 International Joint Conference on Neural Networks, pp. 1-8, 2015.

[13] Deiss, S. R., R. J. Douglas, and A. M. Whatley, "A pulse-coded communications infrastructure for neuromorphic systems," Pulsed Neural Networks , pp. 157-178, 1999.

[14] Eliasmith, C., et al., "A large-scale model of the functioning brain," Science 338, no. 6111, pp.1202-1205, 2012.

[15] Fries, P., D. Nikolić, and W. Singer, "The gamma cycle," Trends in Neurosciences 30, no. 7 pp. 309-316, 2007.

[16] Fusi, S., and M. Mattia, "Collective behavior of networks with linear (VLSI) integrate-and-fire neurons," Neural Computation 11.3, pp. 633-652, 1998.

[17] Gerstner, W., R. Kempter, J. L. van Hemmen, and H. Wagner, "A neuronal learning rule for sub-millisecond temporal coding," Nature 383, no. 6595, pp. 76-78, 1996.

[18] Gütig, R., and Haim S., "The tempotron: a neuron that learns spike timing–based decisions," Nature Neuroscience 9, no. 3, pp. 420-428, 2006.

[19] Gütig, R., Tim G., H. Sompolinsky, and M. Meister, "Computing complex visual features with retinal spike times," PloS One 8, no. 1, e53063, 2013.

[20] Guyonneau, R., R. VanRullen, and S. J. Thorpe, "Temporal codes and sparse representations: A key to understanding rapid processing in the visual system," Journal of Physiology-Paris 98, pp. 487-497, 2004.

[21] Guyonneau, R., R. VanRullen, and S. J. Thorpe, "Neurons tune to the earliest spikes through STDP," Neural Computation 17, no. 4, pp. 859-879, 2005.

[22] Hoare, T., Milner, R., "Grand challenges for computing research", The Computer Journal, 48, no. 1, pp. 49-52, 2005.

[23] Hopfield, J. J., "Pattern recognition computation using action potential timing for stimulus representation," Nature 376, pp. 33-36, 1995.

[24] Indiveri, G., Linares-Barranco, et al., "Neuromorphic silicon neuron circuits", Frontiers in Neuroscience 5, 2011.

[25] Institute of Medicine (US) Forum on Neuroscience and Nervous System Disorders. From Molecules to Minds: Challenges for the 21st Century: Workshop Summary. Washington (DC): National Academies Press (US), 2008.

[26] Irwin, M. J., and J. P. Shen, "Revitalizing computer architecture research," Computing Research Association, 2005.

[27] Karnani, M. M., M. Agetsuma, and R. Yuste, "A blanket of inhibition: functional inferences from dense inhibitory

connectivity," Current opinion in Neurobiology 26, pp. 96-102, 2014.

[28] Kheradpisheh, S. R., M. Ganjtabesh, and T. Masquelier, "Bio-inspired unsupervised learning of visual features leads to robust invariant object recognition," Neurocomputing 205, pp. 382-392, 2016.

[29] Kheradpisheh, S. R., M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "STDP-based spiking deep neural networks for object recognition," arXiv preprint arXiv:1611.01421, 2016.

[30] Kistler, W. M., W. Gerstner, and J. L. van Hemmen, "Reduction of the Hodgkin-Huxley equations to a single-variable threshold model", Neural Computation 9.5, pp. 1015-1045, 1997.

[31] Madhavan, A., T. Sherwood, and D. Strukov, "Race logic: abusing hardware race conditions to perform useful computation," IEEE Micro 35, no. 3, pp. 48-57, 2015.

[32] Maass, W., "Networks of spiking neurons: the third generation of neural network models," Neural Networks 10.9, pp.1659-1671, 1997.

[33] Maass, W., T. Natschläger, and H. Markram, "Real-time computing without stable states: a new framework for neural computation based on perturbations," Neural Computation 14.11, pp. 2531-2560, 2002.

[34] Mainen, Z. F., and T. J. Sejnowski, "Reliability of spike timing in neocortical neurons," Science 268, no. 5216 , pp. 1503-1506, 1995.

[35] Markram, H., J. Lübke, M. Frotscher, and B. Sakmann, "Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs," Science 275, no. 5297, pp. 213-215, 1997.

[36] Masquelier, T., and S. J. Thorpe, "Learning to recognize objects using waves of spikes and Spike Timing-Dependent Plasticity," 2010 International Joint Conference on Neural Networks, pp. 1-8, 2010.

[37] Masquelier, T., and S. J. Thorpe, "Unsupervised learning of visual features through spike timing dependent plasticity," PLoS Comput Biol 3, no. 2, e31, 2007.

[38] Morrison, A., M. Diesmann, and W. Gerstner, "Phenomenological models of synaptic plasticity based on spike timing," Biological Cybernetics 98, no. 6, pp. 459-478, 2008.

[39] Mountcastle, V., "The Columnar organization of the neocortex," Brain 120, no. 4, pp. 701-722, 1997.

[40] NAE Grand Challenges for Engineering, "Reverse Engineer the Brain", http://www.engineeringchallenges.org/challenges/9109.aspx

[41] Natschläger, T., and B. Ruf, "Spatial and temporal pattern analysis via spiking neurons," Network: Computation in Neural Systems 9, no. 3, pp. 319-332, 1998.

[42] O'Connor, P., D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, "Real-time classification and sensor fusion with a spiking deep belief network," Frontiers in Neuroscience 7, 2013.

[43] Pfeil, T., et al., "Is a 4-bit synaptic weight resolution enough?–constraints on enabling spike-timing dependent plasticity in neuromorphic hardware," Frontiers in Neuroscience 6, 2012.

[44] Probst, D., W. Maass, H. Markram, and M.-O. Gewaltig, "Liquid computing in a simplified model of cortical layer IV: learning to balance a ball", Artificial Neural Networks and Machine Learning–ICANN, pp. 209-216, 2012.

[45] Querlioz, D., O. Bichler, P. Dollfus, and C. Gamrat, "Immunity to device variations in a spiking neural network with memristive nanodevices," IEEE Transactions on Nanotechnology 12, no. 3, pp. 288-295, 2013.

[46] Schrauwen, B., and J. Van Campenhout, "Extending spikeprop," IEEE International Joint Conference on Neural Networks, pp. 471-475, 2004.

[47] Singer, W., "Neocortical rhythms," Dynamic Coordination in the Brain: From Neurons to Mind, eds. Von der Malsburg, Christoph, William A. Phillips, and Wolf Singer. MIT Press, 2010.

[48] Smith, J. E. "Space-time computing with temporal neural networks," Synthesis Lectures on Computer Architecture, 2017.

[49] Stein, R. B. "A theoretical analysis of neuronal variability," Biophysical Journal 5.2, pp. 173-194, 1965.

[50] Thorpe, S. J., and M. Imbert, "Biological constraints on connectionist modelling," Connectionism in Perspective, pp. 63-92, 1989.

[51] Thorpe, S., A. Delorme, and R. Van Rullen, "Spike-based strategies for rapid processing," Neural Networks 14.6-7, pp. 715-725, 2001.

[52] Tuckwell, H. C. "Synaptic transmission in a model for stochastic neural activity," Journal of Theoretical Biology 77.1, pp. 65-81, 1979.

[53] Weidenbacher, U., and H. Neumann, "Unsupervised learning of head pose through spike-timing dependent plasticity," in Perception in Multimodal Dialogue Systems, ser, Lecture Notes in Computer Science. Springer Berlin 1 Heidelberg, vol. 5078/2008, pp. 123-131, 2008.

[54] VanRullen, R., R. Guyonneau, and S. J. Thorpe, "Spike Times Make Sense", Trends in neurosciences 28.1, pp. 1-4, 2005.

[55] Wysoski, S. G., L. Benuskova, and N. Kasabov,"Evolving spiking neural networks for audiovisual information processing," Neural Networks 23, pp. 819-835, 2010.

[56] Yuste, R., "From the neuron doctrine to neural networks," Nature Reviews Neuroscience 16, no. 8, pp. 487-497, 2015.

[57] Zhao, B., R. Ding, S. Chen, B. Linares-Barranco, and H. Tang, "Feedforward categorization on AER motion events using cortex-like features in a spiking neural network," IEEE Transactions on Neural Networks and Learning Systems 26, no. 9, pp. 1963-1978, 2015.