CHAPTER 9

Deep Learning

Any fool can know. The point is to understand.

—Albert Einstein

Artificial neural networks (ANNs) have had a history riddled with highs and lows since their inception. At a nodal level, ANNs started with highly simplified neural models, such as McCulloch-Pitts neurons (McCulloch and Pitts 1943), and then evolved into Rosenblatt's perceptrons (Rosenblatt 1957) and a variety of more complex and sophisticated computational units. From single- and multilayer networks, to self-recurrent Hopfield networks (Tank and Hopfield 1986), to self-organizing maps (also called Kohonen networks) (Kohonen 1982), adaptive resonance theory and time delay neural networks among other recommendations, ANNs have witnessed many structural iterations. These generations carried incremental enhancements that promised to address predecessors' limitations and achieve higher levels of intelligence. Nonetheless, the compounded effect of these "intelligent" networks has not been able to capture the true human intelligence (Guerriere and Detsky 1991; Becker and Hinton 1992). Thus, Deep learning is on the rise in the machine learning community, because the traditional shallow learning architectures have proved unfit for the more challenging tasks of machine learning and strong artificial intelligence (AI). The surge in and wide availability of increased computing power (Misra and Saha 2010), coupled with the creation of efficient training algorithms and advances in neuroscience, have enabled the implementation, hitherto impossible, of deep learning principles. These developments have led to the formation of deep architecture algorithms that look in to cognitive neuroscience to suggest biologically inspired learning solutions. This chapter presents the concepts of *spiking neural networks* (SNNs) and *hierarchical temporal memory* (HTM), whose associated techniques are the least mature of the techniques covered in this book.

Overview of Hierarchical Temporal Memory

HTM aims at replicating the functional and structural properties of the neocortex. HTM incorporates a number of insights from Hawkins's book *On Intelligence* (2007), which postulates that the key to intelligence is the ability to predict. Its framework was designed as a biomimetic model of the neocortex that seeks to replicate the brain's structural and algorithmic properties, albeit in a simplified, functionally oriented manner. HTM is therefore organized hierarchically, as depicted generically in Figure 9-1. All levels of hierarchy and their subcomponents perform a common computational algorithm.

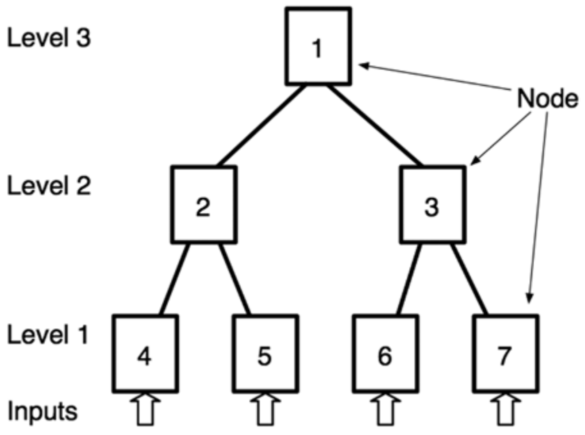

Figure 9-1. *An HTM network's hierarchical structure*

Deep architectures adopt the hierarchical structure of the human neocortex, given the evident existence of a common computational algorithm in the brain that is pervasive throughout the neocortical regions and that makes the brain deal with sensory information—visual, auditory, olfactory, and so on—in very similar ways. Different regions in the brain connect in a hierarchy, such that information flowing up coalesces, building higher and more complex abstractions and representations of the sensory stimuli at each successive level. The brain's structure, specifically the neocortex, evolved to gain the ability to model the structure of the world it senses. At its simplest abstraction the brain can be viewed as a biological data processing black box that discovers external causes in an environment that imposes massive amounts of data on its inputs (senses). The causes of this continuous stream of information are by nature hierarchical, in both space and time. These causes serve as a collection of smaller building blocks that combine to form a larger picture of the world. For instance, speech can be broken down into sentences, sentences into word utterances, word utterances into phonemes, and so on. With digital imagery, pixels combine into edges, edges into contours, contours into shapes, and, finally, shapes into objects. Every sensed object in the world reveals a similar structure perceived at varying levels of granularity. It is this hierarchically organized world that the neocortex and therefore HTM, by imitation, aim at modeling. This modeling happens in HTM at every level.

In HTM the lowest-level nodes of the network are fed sensory information. This information can be raw or preprocessed, depending on the task the network is performing. The nodes learn the most basic features of the data stream by discerning repeatable patterns and the sequences in which these patterns occur and storing them either via local memory structures or via connectivity configurations. These basic patterns and sequences are then used as building blocks at higher levels to form more complex representations of sensory causes. As information travels up the hierarchy, the same learning mechanics are used as higher and higher abstractions of the input patterns are formed. Information can also flow down the hierarchy. This enables the network to act as a generative model, in which higher levels bias lower levels by communicating their internal states to fill in missing input data or resolve ambiguity, or both (Hawkins 2007).

Hierarchical Temporal Memory Generations

HTM has seen so far two generations during its evolution. The underlying implementation in the first generation of computational algorithms, called *Zeta 1*, is strongly rooted in the *Bayesian belief propagation* (BBP) and borrows many of its computations and rules of convergence from that theory. In this earlier version, which is now sunset, HTM used a variation of BBP. BBP is used in Bayesian networks, whereby, under certain topological constraints, the network is ensured to reach an optimal state in the time it takes a message to traverse the network along the maximum path length. Thus, BBP forces all nodes in a network to reach mutually consistent beliefs. The state of these nodes is encoded in probabilistic terms, and Bayesian theory is used to process and fuse information. HTM can be thought of as a Bayesian network with some additions to allow for handling time, self-training, and discovery of causes.

The second-generation algorithms were created to make the framework more biologically feasible. Functionally, many of the concepts of invariant representation and spatial and temporal pooling were carried over by reference to principles of *sparse distributed representation* (SDRs) and structural change. Nodes were replaced with closer analogues of cortical columns with biologically realistic neuron models, and connectivity was altered to allow strong lateral inhibition (Edelman and Mountcastle 1978). Cortical columns are a collection of cells characterized by common feedforward connections and strong inhibitory interconnections (Edelman and Mountcastle, 1978).

Second-generation algorithms—initially referred to as *fixed-density distributed representations* (FDRs) and now simply called HTM *cortical learning algorithms* (CLAs)—replace Zeta 1. In lieu of the “barrel” hierarchy, with its clean-cut receptive fields, each level in the updated framework is a continuous region of cells stacked into columns that act as a simplified model of Edelman and Mountcastle’s (1978) cortical columns. Figure 9-2 depicts the structure of HTM, with cells organized into columns, columns into levels, and levels into a hierarchy of cortical regions.

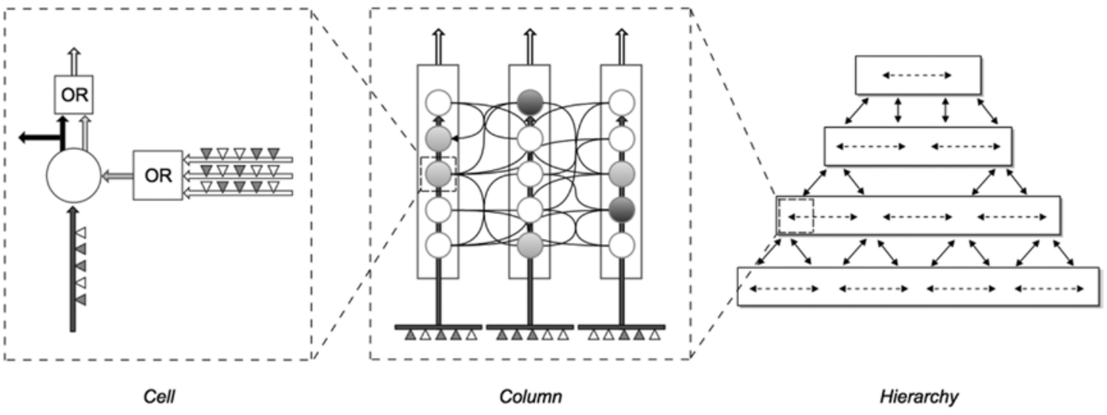

Figure 9-2. *HTM structure*

Whereas Zeta 1 was strongly rooted in Bayesian theory—specifically, in belief propagation—CLA is founded on the principles of SDR. To understand the underlying implementation of CLA, a discussion of SDR and how it is fundamental to HTM theory is necessary.

The architecture suggested for the first-generation HTM model was strongly influenced by the Bayesian rules it implemented. It benefited from the structural characteristics of the neocortex, namely, its hierarchical organization; however, nodes diverged from their biological counterparts. In short, functional modeling was the emphasis. In CLA, HTM abandons these roots and adheres more strictly to neocortical structural guidelines. The result is a neuron model, known in this context as an *HTM cell*. Figure 9-3 depicts the model put forward by Hawkins, Ahmad, and Dubinsky (2011). These cells are more realistic and biologically faithful than those used in traditional ANNs.

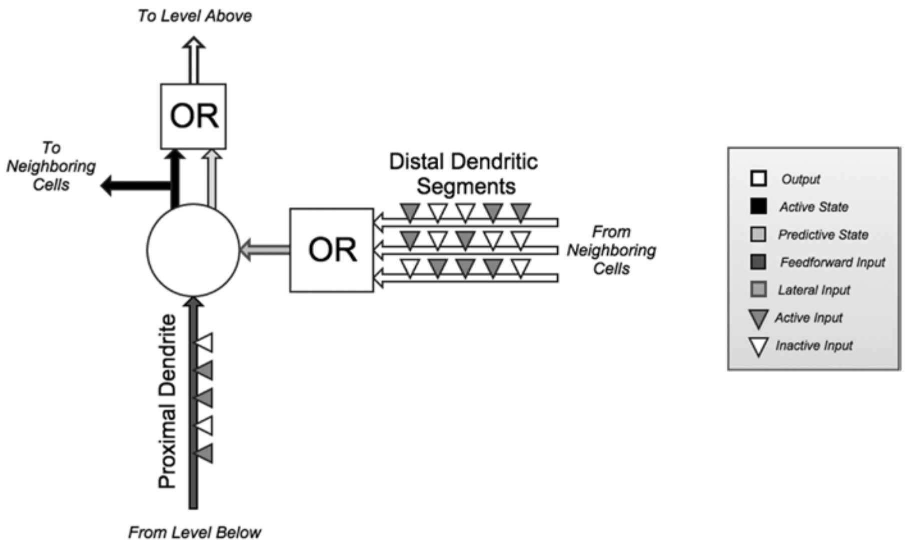

Figure 9-3. *An HTM cell/neuron model*

HTM cells have two types of incoming connection structures: *proximal* dendrites and *distal* dendrites. Dendritic segments of both types are populated by synapses that connect them to other neighboring cells. These synaptic connections are binary, owing to the stochastic nature of real neurons. Because Hawkins, Ahmad, and Dubinsky (2011) posited that any algorithm that aims at emulating the brain cannot rely on the precision or fidelity of individual neurons, HTM cells were modeled to have binary, nonweighted synapses; meaning, they are either connected or not. To account for a real neuron's ability to retract and extend to form connections, HTM cell synapses are assigned a parameter, called *permanence*. Permanence is a scalar value between 0 and 1 that is incremented or decremented, based on a synapse's contribution to activity. When permanence is above a predefined threshold, a synapse becomes connected. Therefore, all synapses in CLA are potential synapses. They are dynamic elements of connectivity.

Proximal dendrites are responsible for feedforward connectivity between regions. These dendrites are populated by a set of potential synapses that are associated with a subset of an input to an HTM region. The dendrites are shared by all the cells of a column (shared feedforward connectivity) and act as linear summation units. Distal dendrites are responsible for lateral connections across a single region. Several segments are associated with a distal dendrite. These segments act as a set of threshold coincidence detectors, meaning that, when enough connections are active at one time, they trigger a response in the receiving cell. It is enough for one segment to be active to trigger a response in the cell (OR gate). Each segment connects an HTM cell to a different subset of neighboring cells. The activity of those cells is monitored and allows the receiving cell to enter a predictive state. This is essentially the root of prediction in an HTM network: every cell continuously monitors the activity of surrounding cells to predict its own.

Finally, an HTM cell has two binary outputs. The first output, owing to proximal connections, forces the cell into an active state if enough activation is present. The second output, owing to distal connections, forces the cell into a predictive state if enough neighbor activity is present; the cell expects to be activated soon. This enables HTM to act in a predictive manner and react to sequences of input. Finally, the output of the HTM cell is the OR of these two outputs. This is what regions higher up in the hierarchy receive as input. Figure 9-4 shows an example of activation of cells in a set of HTM columns: at any point, some cells will be active, as a result of feedforward input (dark gray), whereas other cells, receiving lateral input from active cells, will be in a predictive state (light gray).

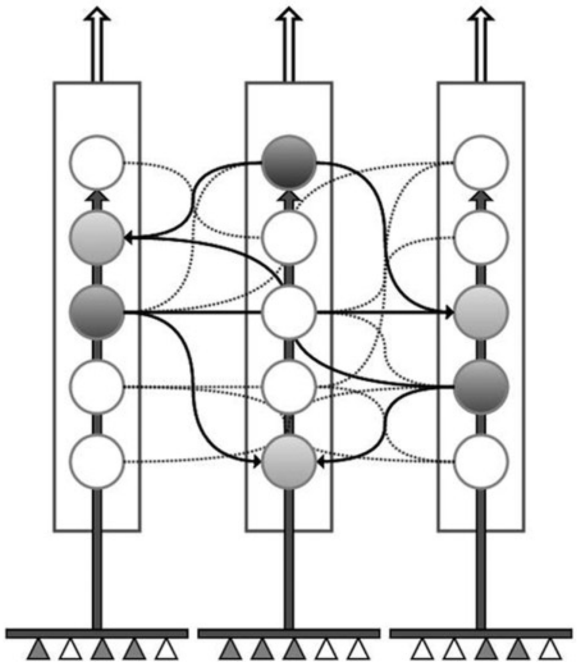

Figure 9-4. *HTM columns*

Sparse Distributed Representation

CLA borrows many of its principles of operation for its biological analogue. The neocortex is made up of more than 10^{11} highly interconnected neurons. Yet, it is still capable of reacting to stimuli with relatively sparse activation levels. This is made possible by the vast amount of inhibitory connections. Inhibition guarantees that only a small number of neurons are activated at any one time. Furthermore, CLA implements the same encoding strategy as SDR. Using lateral synaptic connections, strongly stimulated neural columns inhibit nearby activity, thus reducing the number of active columns and yielding a sparse internal representation of the input pattern or stimuli. This internal representation is also *distributed*, that is, spread out across a region.

Because active bits are sparse, knowledge of a subset of them still carries information about the input pattern, in contrast to other representations, such as ASCII code. With ASCII code an individual bit is meaningless; SDR, therefore, injects a representational quality into individual activations. Because only a tiny fraction of possibly a large number of neurons are active, whatever semantic meaning a single neuron gains becomes specific to a limited number of similar patterns. Consequently, even a subset of the active neurons of a pattern can be a good indicator of it. The theoretical loss of information that results from enforcing this kind of sparseness does not have a practical effect (Hawkins, Ahmad, and Dubinsky 2011).

Algorithmic Implementation

Contrary to the first generation, separation of the learning phase from the inference phase does not offer much insight into the rules of operation of HTM. In CLA, learning, inference, and—most important—prediction occur harmoniously. Each level of the hierarchy is always predicting. Learning by the lower nodes can be turned off when they stabilize; it occurs online in tandem with prediction, when activated.

Therefore, it is best to consider the operation of CLA in terms of its pooling functions. The following two sections discuss the theory behind spatial and temporal pooling in second-generation algorithms and show how they are implemented in the cortical network, as suggested by Hawkins, Ahmad, and Dubinsky (2011).

Spatial Pooler

The role of the spatial pooler is the same in CLA as in Zeta 1. Input patterns that are spatially similar should have a common internal representation. The representation should be not only robust to noise, but also sparse to abide by the principles of SDR. These goals are achieved by enforcing competition between cortical columns. When presented with input data, all columns in an HTM region will compute their feedforward activation. Columns that become active are allowed to inhibit neighboring columns. In this way, only a small set of strongly active columns can represent a cluster of similar inputs. To give other columns a fair chance at activation and ensure that all columns are used, a boosting factor is added. This enables weak columns to better compete.

For each of the active columns, permanence values of all the potential synapses are adjusted, based on Hebbian learning rules. The permanence values of synapses aligned with active input bits are increased, whereas permanence values of synapses aligned with inactive input bits are decreased. Figure 9-5 shows a flow chart of the phases involved.

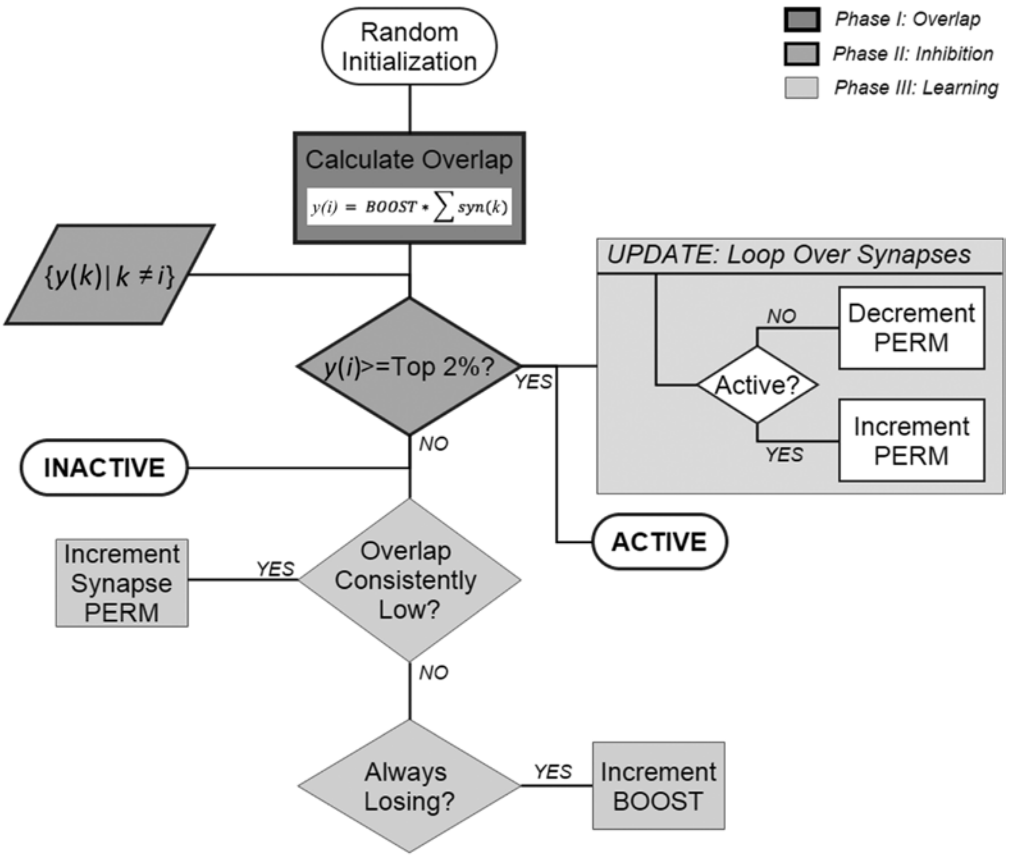

Figure 9-5. *Spatial pooler flowchart*

The spatial pooling operations are as follows:

- *Phase 0 (corresponding to initialization)*: Each column is randomly assigned a random set of inputs (50 percent of the input vector), which is referred to as the *potential pool* of the column. Each input within this pool is represented by a potential synapse and assigned a random permanence value. The choice of permanence value is decided according to the following criteria:
    - Values are chosen from within a small range around the permanence threshold. This enables potential synapses to become connected (or disconnected) after a small number of training iterations.
    - Each column has a natural center over the input region, and the permanence values have a bias toward this center, with higher values near the center.
- *Phase 1 (corresponding to overlap)*: The overlap for each column is computed as the number of connected synapses with active inputs multiplied by its boost. If this value is below a predefined threshold ("minOverlap"), the overlap score is set to 0.
- *Phase 2 (corresponding to inhibition)*: The number of winning columns in a local area of inhibition (neighborhood of a column) is set to a predefined value, *N*. A column is a winner if its overlap score is greater than the score of the *N*th highest column within its inhibition radius. A variation of this inhibition strategy that is significantly less computationally demanding is picking the columns with the highest overlap scores for every level of the hierarchy.
- *Phase 3 (corresponding to learning)*: During this phase, updates to the permanence values of all synapses are performed as necessary as well as to the parameters, such as boost and inhibition radius. For winning columns, if a synapse is active, its permanence value is incremented; if inactive, it is decremented. There are two separate boosting mechanisms in place to help a column learn connections. If a column does not win often enough, its overall boost value is increased; alternatively, if a column's connected synapses do not overlap well with any inputs often enough, its permanence values are boosted. This phase terminates with updating the inhibition.

Temporal Pooler

With winning columns calculated by the spatial pooler, the HTM network gains insight into what pattern it may be seeing at its input. What it lacks is context. Any one pattern can occur as part of a large number of sequences, that is, in multiple contexts. Essential to HTM theory is the ability to predict through the learning of sequences. In CLA sequential learning happened using multiple cells per column. All the cells in a column share feedforward activation, but only a subset (usually a single cell) is allowed to be active. This means that the same pattern, represented by the same set of columns, can be represented by different cells in each column, depending on the context in which the pattern occurs. On a cellular level each of a cell's dendritic distal segments has a set of connections to other cells in the same region, which is used to recognize the state of the network at some point in time. Cells can predict when they will become active by looking at their connections. A particular cell may be part of dozens or hundreds of temporal transitions. Therefore, every cell has several dendrite segments, not just one.

There are three phases involved with temporal pooling. In the first phase each cell's active state is computed. Phase 2 computes each cell's predictive state. In Phase 3, synapses are updated by either incrementing or decrementing their permanence values. Following is the general case, in which both inference and learning are taking place, with the caveat that the learning phase in CLA can be switched off.

- *Phase 1*: For every winning column the active state of each of its cells is computed here. Also, a cell is designated a learning cell. If any of the cells is in a predictive state, owing to its lateral connections, it is put in an active state. If a learning cell contributed to its lateral activation, the cell is chosen as a learning cell, too. In contrast, if no cell in the column is in a predictive state, all the cells are turned active to indicate that the context is not clear—a process called *bursting*. Additionally, the best matching cell becomes the learning cell, and a new distal segment is added to that cell.
- *Phase 2*: Once all the cells in the winning columns are updated, their states can be used for prediction in the cells of other columns. Every cell in the region computes its lateral/distal connection. If any of a cell's segments are activated, owing to feedforward activation in other cells, the cell is put in a predictive state. The cell then queues up the following changes:
    - Reinforcement of the currently active segment by incrementing the permanence values for active synapses and decrementing the values for synapses that are inactive
    - Reinforcement of a segment that could have predicted this activation, that is, a segment that has a (potentially weak) match to activity in the previous time step
- *Phase 3*: This is the phase in which learning occurs, by deciding which of the queued-up updates are to be committed. Temporary segment updates are implemented once you have feedforward input and a cell is chosen as a learning cell. Thus, you update the permanence of synapses only if they correctly predicted the feedforward activation of the cell; otherwise, if the cell stops predicting for any reason, the segments are negatively reinforced.

Related Work

Zhituo, Ruan, and Wang (2012) used multiple HTMs in a content-based image retrieval (CBIR) system, which leverages the categorical semantics of a query image, rather than low-level image features, for image indexing and retrieval. Using ten individual HTM networks with some training and testing datasets of size 50 each, recall rates greater than 95 percent were achieved for four of the five categories involved and greater than 70 percent for the fifth category.

Bobier (2007) recreated a handwritten digit recognition experiment on the United States Postal Service database reported by Numenta (Hawkins's company) to have achieved a 95 percent accuracy rate. The digit images were binarized and fed to the network at varying parameters to reach a maximum rate of 96.26 percent—which, the authors noted, was not up to par, compared with other classifiers, such as support vector machine (SVM), which delivered higher rates in a fraction of the computational time.

Kostavelis, Nalpantidis, and Gasteratos (2012) presented a biologically inspired object recognition system. Saliency maps were used to emulate visual fixation and reveal only the relevant parts of the image, thus reducing the amount of redundant information presented to the classifier. The authors chose to substitute the temporal pooler with a correlation-based alternative, using the ETH-80 and supervised learning at the top node. The system outperformed other HTM-based implementations with both SVM and *k*-NN as top-level supervisors.

Sinkevicius, Simutis and Raudonis (2011) explored using HTM for human traffic analysis in public spaces. Two HTM networks were designed: one for human detection, the other for direction of movement detection. An experiment involving use of an overhead camera, mounted on a doorway, was performed, and detection performance was evaluated, using multiple scenarios of varying difficulties. The average accuracy achieved was 80.94 percent for pedestrian detection and 73.44 percent for directional detection.

Boone et al. (2010) used HTM as an alternative to traditional computer vision techniques for diabetic retinopathy. HTM primarily detected the optic nerve on retina images. The images were segmented into fragments the size of an optic nerve and presented with labels (0 or 1) to HTM. Following supervised training the HTM network was able to correctly classify 77.44 percent of the optic nerves presented, leading the authors to conclude that HTM is not competitive with traditional techniques, despite its promise.

Zhuo et al. (2012) supplemented state-of-the-art image classification techniques with locality-constrained linear coding (LLC), spatial pyramid matching (SPM), and HTM for feature pooling. Image descriptors were extracted and encoded using LLC. The LLC codes were then fed to HTM and multiscale SPM to form an image vector. The system was evaluated using a Caltech 101 dataset and UIUC-Sport dataset, with linear SVM as the classifier. Results showed an increase in accuracy for both datasets (73.5 percent versus 71.2 percent and 86.7 percent versus 84.2, respectively), compared with the original LLC model.

Gabrielsson, Konig, and Johansson (2012) aimed at leveraging HTM to create a profitable software agent for trading financial markets. A supervised training scheme was used for HTM, with intraday tick data for the E-mini Standard and Poor's 500 futures markets. The tuned model was used as a predictor of market trends and showed at least comparable results when evaluated against ANNs.

■ **Note** Most of the work making use of HTM has been carried out by the developer community established by Numenta. Annual "hackathons" have produced multiple demos, in which CLA was employed for traffic prediction, human movement prediction, tic-tac-toe simulation, infrared (IR) sensor prediction, and so on. One of the more impressive demos, on music analysis and prediction, shows the use of MIDI note sequencing for training; a melody was learned after 25 epochs (`http://numenta.org/blog/2013/06/25/hackathon-outcome.html#jin-danny-stan`). Currently, research is being undertaken on a CLA model for use in *natural language processing* (NLP). An example is available on the GitHub web site (`https://github.com/chetan51/linguist`). However, more research and validation are needed to compare HTM performance with the state-of-the-art machine learning approaches the model claims to be superior or similar to. With HTM reported performances, the community has yet to see where HTM has the upper hand.

Overview of Spiking Neural Networks

SNNs are biologically inspired networks and belong to the third generation of ANNs. It seems for ANNs that any improvement on the performance should be based on the neuron model. The neuron model in the second generation of ANNs is based on a simplified model of the actual neuron which ignores the actual way of encoding the information between neurons and the type of this information. SNNs are similar to the ANNs architecture which consists of one or more layers of connected neurons, but differ in the neuron's model and the type of the activation function. In contrast with the second generation of ANNs which utilize time-missing continuous activation functions, SNNs rely on the spike timing in their learning and activation phases. SNNs strive to mimic human neurons, in using spikes to transmit and learn the *spatio-* and *spectrotemporal data* (SSTD) that are encoded with the location of the synapses, for the spatial data, and with the spiking-time activities, for the temporal data.

SNNs and their variants have been used in many applications, such as character recognition (Gupta and Long 2007), sign language recognition (Schliebs, Hamed, and Kasabov 2011), visual and auditory pattern recognition (Wysoski, Benuskova, and Kasabov 2006, 2007), image clustering (Meftah et al. 2008), car crash identification (Kasabov et al. 2013), human behavior recognition (Meng, Jin, and Yin 2011), breast cancer classification (O'Halloran et al. 2011), human localization in sensor networks (Obo et al. 2011), intrusion detection (Budjade 2014; Demertzis and Illiadis 2014), electroencephalography (EEG) spatio-/spectrotemporal pattern recognition (Kasabov et al. 2013), and taste recognition (Soltic and Kasabov 2010). Generally, SNNs are not as popular as other methods of machine learning, owing to their high computational cost.

To understand how SNNs differ from ANNs, it is necessary to examine the most common models of human neurons: the Hodgkin-Huxley model, the integrate-and-fire model, the leaky integrate-and-fire model, the Izhikevich model, and Thorpe's model. These models are covered in the following sections.

Hodgkin-Huxley Model

The *Hodgkin-Huxley model* formulates the propagation of action potential in neurons and may be considered the basis of the other models. Hodgkin and Huxley modeled the electrochemical information of natural neurons after the giant axon of the squid. The model consists of four differential equations describing the change in electrical charge on the part of the neuron's membrane capacitance as functions of voltage (V_m) and current ($I(t)$),

$$C\frac{du}{dt} = -g_{Na}m^3h(u - E_{Na}) - g_K n^4(u - E_K) - g_L(u - E_L) + I(t)$$

$$\tau_n\frac{dn}{dt} = -\left[n - n_0(u)\right], \quad \tau_m\frac{dm}{dt} = -\left[m - m_0(u)\right], \quad \tau_h\frac{dh}{dt} = -\left[h - h_0(u)\right],$$

where

$I(t)$ is the input current caused by the presynaptic potentials

g_{Na}, g_K, g_L are the conductance parameters for the sodium and potassium ion channels and the leak per unit area, respectively

E_{Na}, E_K, E_L are the equilibrium potentials

m, n, h are dimentionless variables that are governed by three other differential equations

Because of the complexity of these equations, caused by the nonlinearity and four-dimensionality of the data, several simpler forms were proposed for practical implementation. We discuss some of these proposals in the following sections.

Integrate-and-Fire Model

The *integrate-and-fire model* is derived from the Hodgkin-Huxley model but neglects the shape of the potential actions. This model assumes that all potential actions are uniform but differ in the time of occurrence. As a result for the previous simplification, all the spikes have the same characteristics such as shape, width, and amplitude. The membrane capacitance and *postsynaptic potential* (PSP) are given by the equations

$$C\frac{du}{dt} = -\frac{1}{R}(u(t) - u_{rest}) + I(t)$$

$$u\left(t^{(f)}\right) = \vartheta \;\; with \;\; u'\left(t^{(f)}\right) > 0,$$

where

u_{rest} is the membrane potential of the neuron at the initial state

ϑ is the threshold value at which the neuron fires

$t^{(f)}$ is the spike firing time

$I(t)$ is *the* input current, caused by the presynaptic potentials

Leaky Integrate-and-Fire Model

The *leaky integrate-and-fire model* differs from the integrate-and-fire model, in that the membrane potential of the neuron decays over time if no potentials reach the neuron. When the membrane potential $u(t)$ of the neuron reaches a specific threshold ϑ at time t, called the spiking time $t^{(f)}$, and the $u(t)$ satisfies the $u`(t^{(f)}) > 0$ condition, the neuron emits a spike immediately. Then, the neuron goes under an absolute refractory period u_{abs} which means that the neuron will neglect any effect of the arriving spikes during this period. The refractory period lasts for a specific time d_{abs}; the membrane potential of the neuron during this period is

$$u(t) = -u_{abs},$$

where u_{abs} is the refractoriness potential.

When d_{abs} expires, the membrane potential returns to the u_{rest} value. The membrane potential is given by:

$$\tau_m \frac{du}{dt} = u_{rest} - u(t) + RI(t),$$

where

τ_m is the time constant of the neuron membrane

u_{rest} is the membrane potential of the neuron at the initial state

$I(t)$ is the input current, caused by the presynaptic potentials

R is the equivalent resistance of the neuron model.

Izhikevich Model

The *Izhikevich model* is a tradeoff between biological plausibility and computational efficiency. This model uses the following two differential equations to represent the activities of the membrane potential:

$$\frac{du}{dt} = 0.04\, u(t)^2 + 5u(t) + 140 - w(t) + I(t)$$

$$\frac{dw}{dt} = a\big(bu(t) - w(t)\big).$$

The after-spiking action is described by the term below where membrane potential and recovery variable are reset

$$\text{if } u \geq \vartheta, \text{ then } u \leftarrow c, \text{ and } w \leftarrow w + d.$$

Here, u represents the membrane potential, and w represents a membrane recovery variable that provides –ve feedback to u. a, b, c, and d are the dimensionless parameters. Due to the simplicity of this model, large number of neurons can be simulated of a compute machine.

Thorpe's Model

Thorpe's model is a variation of the integrate-and-fire model that takes into consideration the order of the spikes as they reach the neuron. Thorpe's model is suitable for many applications, because it uses the simple mathematical representation:

$$PSP_i = \sum w_{ji} * mod^{order_j},$$

where

w_{ji} is the weight or efficiency of synapsis between neuron *j* and neuron *i*

mod is a modulation factor $\in [0,1]$

$order_j$ is the firing order of the presynaptic neuron *j*, where $j \in [1, n-1]$, and *n* is the number of presynaptic neurons connected to neuron *i*

The weights in this model are updated according to

$$\Delta w_{ji} = mod^{order_j}.$$

Thorpe's model makes stronger connections with the connected neurons that fire and reach the current neuron early. Spiking occurs whenever PSP_i reaches a threshold value $PSP_{\theta i}$. After the spiking, PSP_i is immediately set to 0, such that:

$$PSP_i = \begin{cases} PSP_i + P_{ji} \; when \;\; PSP_i < PSP_{\theta i} \\ 0 \qquad\qquad when \;\; PSP_i \geq PSP_{\theta i} \end{cases}.$$

This method allows a fast and real-time simulation of large networks.

Information Coding in SNN

Information coding in neurons has long been the subject of lively debate, in terms of whether the information in the neuron is coded as rate coding or spike coding. Recent studies have shown that the information is encoded as spike coding, because rate coding is insufficient as a representation of the ability of neurons to process information rapidly.

Rank coding is very efficient and can achieve the highest information coding capacity. Rank coding starts by converting the input values into a sequence of spikes, using the Gaussian receptive fields. The Gaussian receptive fields consist of *m* receptive fields that are used to represent the input values *n* as spikes. Assuming that *n* takes values from the range $[I^n_{min}, \; I^n_{max}]$, the Gaussian receptive field for the neuron *i* is given by its center u_i,

$$u_i = I^n_{min} + \frac{2i-3}{2} * \frac{I^n_{max} - I^n_{min}}{M-2},$$

and the width σ,

$$\sigma = \frac{1}{\beta} * \frac{I^n_{max} - I^n_{min}}{M-2},$$

where β is a parameter that controls the width of the receptive field with $1 \leq \beta \leq 2$.

Unlike the common learning methods, which depend on the rate of spiking, SNNs use a variant of Hebb's rule to emphasize the effect of the spikes' timing. The weight-updating mechanism is based on the interval between the firing time of the presynaptic and postsynaptic neurons. Two types of neurons are involved in the weights updating process; the neurons before the synapses of the current neuron (the presynaptic neurons) and the neuron after the synapses (the postsynaptic neuron). If the postsynaptic neuron fires right after the postsynaptic neuron, then the connection between these two neurons is strengthened, such that the neuron's weight is given by:

$$\textit{if } \Delta t \geq 0, \textit{ then } w_{new} \leftarrow w_{old} + \Delta w, \textit{ where}$$
$$\Delta t \textit{ is the difference in firing time and it is equal to} t_{post} - t_{pre}.$$

If the presynaptic neuron fires right after postsynaptic neuron, then the connection between these two neurons is weakened, such that

$$\textit{if } \Delta t < 0, \textit{ then } w_{new} \leftarrow w_{old} - \Delta w.$$

When the firing time of the postsynaptic neuron does not occur immediately after the firing time of the presynaptic neuron, the weights are not updated.

The preceding discussion addresses the excitatory connection. The inhibitory connection uses a simple process, as it does not take into account the interval between the firing time of the presynaptic and postsynaptic neurons.

Learning in SNN

The most popular algorithms developed for SNN are the *SpikeProp* and the *Theta learning* rule. SpikeProp is similar to the backpropagation algorithm that was designed to adjust the weights in the second generation of the neural networks. The Theta learning rule uses the *quadratic integrate and fire* (QIF) neuron model. Both of these algorithms are very sensitive to the parameters of the neuron model, and sometimes these algorithms suffer spike-loss problems. Spike loss occurs when the neuron does not fire for any patterns and hence cannot be recovered by the gradient method. Another approach for training SNNs is using evolutionary strategies that do not suffer from the tuning sensitivity, but these are computationally intensive and costly.

Assume **H**, **I** and **J** are the input layer, the hidden layer and the output layer respectively. Each neuron from a specific layer is represented by the lower cases **i**, **h** and **j**. The set of neurons that are preceding the neuron **i** are denoted by Γ_i while the neurons that are succeeding the neuron i are denoted by Γ^i. Each connection between two neurons in the consecutive layers is consisted of $\boldsymbol{m} \in \{1..m\}$ subconnections where each has a constant incremental delay d_k and a weight w_{ij}^k with $k \in \{1..m\}$. t_i, t_h and t_j represent the spike time of the neuron in the respective layers, while $\hat{t}_i, \hat{t}_h$ and $\hat{t}_j$ represent the actual spike time in the respective layers.

The response function of the neuron *i* is given by:

$$y_i^k = \varepsilon\left(t - t_i - d_k\right)|_{t_i = \hat{t}_i}$$

with:

$$\varepsilon\left(t\right) = \frac{t}{\tau} e^{1 - \frac{t}{\tau}}$$

$$x_j\left(t\right) = \sum_{i \in \Gamma_j} \sum_{k=1}^{m} w_{ij}^k y_i^k(t)\,|_{t_i = \hat{t}_i}$$

Where τ is the membrane constant.

Weights updating from the output layer to the hidden layer is given by

$$\Delta w_{ij}^{k} = -\eta.\delta_j.y_i^k \,|_{t_i=\hat{t}_i\,,\,t_j=\hat{t}_j}$$

where

$$\delta_j = \left.\frac{\partial E}{\partial t_j}\right|_{t_j=\hat{t}_j} \cdot \left.\frac{\partial t_j}{\partial x_j}\right|_{x_j=\hat{x}_j}$$

$$= \frac{T_j - \hat{t}_j}{\sum_{i\in\Gamma_j}\sum_{l=1}^{m} w_{ij}^{l} \frac{\partial}{\partial t}(y_i^l)\Big|_{t_i=\hat{t}_i\,,\,t_j=\hat{t}_j}}$$

Weights updating from the hidden layers to the input layer is given by

$$\Delta w_{hi}^{k} = -\eta.\delta_i.y_h^k \,|_{t_i=\hat{t}_i\,,\,t_h=\hat{t}_h}$$

Where

$$\delta_i = \left.\frac{\partial E}{\partial t_i}\right|_{t_i=\hat{t}_i} \cdot \left.\frac{\partial t_i}{\partial x_i}\right|_{x_i=\hat{x}_i}$$

$$= \frac{\sum_{j\in\Gamma^i} \delta_j \sum_{l=1}^{m} w_{ij}^{l} \frac{\partial}{\partial t}(y_i^l)\Big|_{t_i=\hat{t}_i\,,\,t_j=\hat{t}_j}}{\sum_{h\in\Gamma_i}\sum_{l=1}^{m} w_{hi}^{l} \frac{\partial}{\partial t}(y_h^l)\Big|_{t_i=\hat{t}_i\,,\,t_h=\hat{t}_h}}$$

The weaknesses of the SpikeProp algorithm include the following:

- The membrane potential of neurons is calculated at fixed time-step intervals.
- There is no method for selecting the initial weights and the thresholds.
- A reference neuron that spikes at t=0 is required.
- Convergence is compromised owing to the insufficient spike response function.

SNN Variants and Extensions

Variants and extensions of SNNs are reviewed in the following sections.

Evolving Spiking Neural Networks

Evolving spiking neural networks (eSNNs) are a variant class of SNNs that have a dynamic architecture (Schliebs and Kasabov 2013). eSNNs use Thorpe's model and a population rank coding to encode information. eSNNs combine *evolving connectionist system* (ECoS) (Kasabov 2007) architecture and SNNs. Compared with SNNs, eSNNs have three advantages. First, eSNNs have a lower computational cost, as they are dependent on a light neuron model, namely, Thorpe's model. Second, the learning algorithm in eSNN is more effective than those in SNNs, which are unable to converge 20 percent of the time (Thiruvarudchelvan, Crane, and Bossomaier 2013). Third, eSNNs are online learning models, which gives them a clear advantage over other techniques.

Reservoir-Based Evolving Spiking Neural Networks

Reservoir computing (RC) is a framework of randomly and sparsely connected nodes (neurons) used to solve complex dynamic systems. RC is divided into *echo state machine* (ESM) (Jaeger 2007) and *liquid state machine* (LSM) (Maass, Natschläger, and Markram 2002; Maass 2010) types.

The LSM is a real-time computation model that accepts continuous streams of data and that generates a high-dimensional continuous output stream. The LSM can be seen as a dynamic SVM kernel function. The architecture of the LSM consists of a liquid and a readout function. A liquid may be seen as a filter that has a trained architecture and that is used for general-purpose problem solving, such as recognizing different kinds of objects from the same video stream. In contrast, a readout function is specific purpose, such that different readout functions are used to recognize different objects from the same liquid. The readout function should be a linear-discriminant and memory-less function.

Dynamic Synaptic Evolving Spiking Neural Networks

Dynamic synaptic evolving spiking neural networks (deSNNs) are a class of eSNNs that use dynamic weight adjustments throughout the learning process (Kasabov et al. 2013). The dynamic updating of weights makes the model more efficient at capturing the complex patterns of a problem. In deSNN the weights continue to change slightly, according to the spikes arriving at the connection, as opposed to eSNN, in which the weights are updated once.

Probabilistic Spiking Neural Networks

The neural model of the *probabilistic spiking neural network* (pSNN) (see Figure 9-6) uses three types of probabilities (Kasabov 2010):

$P_{cj,i}(t)$, the probability that a spike emitted from neuron n_j reaches neuron n_i at a time moment t through the connection cj,i between n_j and n_i.

$P_{sj,i}(t)$, the probability that synapse $s_{j,i}$ contributes to the $PSP_i(t)$ after receiving a spike from neuron n_j.

$P_i(t)$, the probability that neuron n_i emits a spike after its PSP reaches the emitting thresholds.

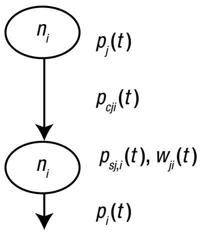

Figure 9-6. *The pSNN model*

The PSP of neuron n_i is given by:

$$PSP_i(t) = \sum_{p=t_0,\ldots,t} \sum_{j=1,\ldots,m} e_j g\left(P_{cj,i}(t-p)\right) f\left(P_{sj,i}(t-p)\right) w_{j,i}(t),$$

where

e_j is 1 if a spike has been emitted from neuron n_j, and 0 otherwise

$g(P_{cj,i}(t - p))$ is 1 with a probability of $P_{cj,i}(t)$, and 0 otherwise

$g(P_{sj,i}(t - p))$ is 1 with a probability of $P_{sj,i}(t)$, and 0 otherwise

Conclusion

This and the preceding two chapters have covered four learning algorithms that relate to deep learning: deep neural networks (DNNs), cortical algorithms (CAs), hierarchical temporal memory (HTM), and spiking neural networks (SNNs). DNN is an established technique developed from a traditional AI perspective and has shown robust results in many applications. CA, SNN, and HTM are biologically inspired techniques that are less mature but that are regarded by advocates of biologically inspired computing as highly promising. Traditional AI exponents, however, argue that DNN-related approaches will be the future winners in the learning area. Which vision will prevail is a hotly contested issue.

But, instead of asking who is right and who is wrong, we might do better to frame the question in terms of the context and aim of the learning sought. Are we seeking to create a universal form of machine intelligence that replicates the capability of human intelligence to learn by inference from a few instances for a wide variety of applications? Or, are we after computationally efficient learning frameworks that can crush with brute force the big data collected from the Internet of Things (IOT) to develop models capable of predictive and descriptive tasks?

As researchers have yet to agree on a unique definition of AI or on a golden metric for evaluating learning algorithms, the intention of the present work is to provide readers with a general background and a snapshot in time of this rapidly expanding subfield of deep learning.

References

Bobier, Bruce. “Handwritten Digit Recognition Using Hierarchical Temporal Memory,” 2007.

Boone, Aidan R. W., T. P. Karnowski, E. Chaum, L. Giancardo, Y. Li, and K. W. Tobin Jr. “Image Processing and Hierarchical Temporal Memories for Automated Retina Analysis.” In *Proceedings of the 2010 Biomedical Sciences and Engineering Conference*. Piscataway, NJ: Institute of Electrical and Electronic Engineers, 2010.

Budjade, Gaurav. “Intrusion Detection Using Spiking Neural Networks.” Master’s thesis, Rochester Institute of Technology, 2014.

Demertzis, Konstantinos, and Lazaros Iliadis. “A Hybrid Network Anomaly and Intrusion Detection Approach Based on Evolving Spiking Neural Network Classification.” In *E-Democracy, Security, Privacy and Trust in a Digital World: 5th International Conference, E-Democracy 2013, Athens, Greece, December 5–6, 2013, Revised Selected Papers*, edited by Alexander B. Sideridis, Zoe Kardasiadou, Constantine P. Yialouris, and Vasilios Zorkadis, 11–23. Cham, Switzerland: Springer, 2014.

Dileep, George. “How the Brain Might Work: A Hierarchical and Temporal Model for Learning and Recognition.” PhD diss., Stanford University, 2008.

Dileep, George, and Jeff Hawkins. "Towards a Mathematical Theory of Cortical Micro-Circuits." *PLoS Computational Biology* 5, no. 10 (2009). http://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1000532.

Edelman, Gerald M., and Vernon B. Mountcastle. *The Mindful Brain: Cortical Organization and the Group-Selective Theory of Higher Brain Function*. Cambridge, MA: Massachusetts Institute of Technology Press, 1978.

Gabrielsson, Patrick, R. Konig, and Ulf Johansson. "Hierarchical Temporal Memory-Based Algorithmic Trading of Financial Markets." In *Proceedings of the 2012 IEEE Conference on Computational Intelligence for Financial Engineering and Economics*, 1–8. Piscataway, NJ: Institute of Electrical and Electronic Engineers, 2012.

Guerriere, Michael R. J., and Allan S. Detsky. "Neural Networks: What Are They?" *Annals of Internal Medicine* 115, no. 11 (1991): 906–907.

Gupta, Ankur, and Lyle N. Long, "Character Recognition Using Spiking Neural Networks." In *IJCNN 2007: Proceedings of the 2007 International Joint Conference on Neural Networks*, 53–58. Piscataway, NJ: Institute of Electrical and Electronic Engineers.

Hawkins, Jeff. *On Intelligence*. New York: Times Books, 2007.

Hawkins, Jeff, Subutai Ahmad, and D. Dubinsky. "Hierarchical Temporal Memory, Including HTM Cortical Learning Algorithms." Technical report, Numenta, 2011.

Jaeger, Herbert. "Echo State Network." *Scholarpedia* 2, no. 9: (2007): 2330.

Johnson, Stephen C. "Hierarchical Clustering Schemes." *Psychometrika* 32, no. 3 (1967): 241–254.

Kasabov, Nikola. *Evolving Connectionist Systems*. London: Springer, 2007.

Kasabov, Nikola. "To Spike or Not to Spike: A Probabilistic Spiking Neuron Model." *Neural Networks* 23, no. 1 (2010): 16–19.

Kasabov, Nikola, Kshitij Dhoble, Nuttapod Nuntalid, and Giacomo Indiveri. "Dynamic Evolving Spiking Neural Networks for On-Line Spatio- and Spectro-Temporal Pattern Recognition." *Neural Networks* 41 (2013): 188–201.

Kohonen, Teuvo. "Self-Organized Formation of Topologically Correct Feature Maps." *Biological Cybernetics* 43.1 (1982): 141–152.

Kostavelis, Ioannis, Lazaros Nalpantidis, and Antonios Gasteratos. "Object Recognition Using Saliency Maps and HTM Learning." In *Proceedings of the 2012 IEEE International Conference on Imaging Systems and Techniques*, 528–532. Piscataway, NJ: Institute of Electrical and Electronic Engineers, 2012.

Maass, Wolfgang. "Liquid State Machines: Motivation, Theory, and Applications." In *Computability in Context: Computation and Logic in the Real World*, edited by S. Barry Cooper and Andrea Sorbi, 275–296. London: Imperial College Press, 2011.

Maass, Wolfgang, Thomas Natschläger, and Henry Markram. "Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations." *Neural Computation* 14, no. 11 (2002): 2531–2560.

McCulloch, W. S., and W. H. Pitts. *A Logical Calculus of the Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics* 5 (1943): 115–133.

Meftah, B., A. Benyettou, O. Lezoray and W. QingXiang. "Image Clustering with Spiking Neuron Network." In *IJCNN 2008: Proceedings of the IEEE International Joint Conference on Neural Networks*, 681–685. Piscataway, NJ: Institute of Electrical and Electronic Engineers, 2008.

Meng, Yan, Yaochu Jin, and Jun Yin. “Modeling Activity-Dependent Plasticity in BCM Spiking Neural Networks with Application to Human Behavior Recognition.” *IEEE Transactions on Neural Networks* 22, no. 12 (2011): 1952–1966.

Misra, Janardan, and Indranil Saha, “Artificial Neural Networks in Hardware: A Survey of Two Decades of Progress.” *Neurocomputing* 74, nos. 1–3 (2010): 239–255.

Obo, Takenori, Naoyuki Kubota, Kazuhiko Taniguchi, and Toshiyuki Sawayama. “Human Localization Based on Spiking Neural Network in Intelligent Sensor Networks.” In *Proceedings of the 2011 IEEE Workshop on Robotic Intelligence in Informationally Structured Space*, 125–130. Piscataway, NJ: Institute of Electrical and Electronic Engineers, 2011.

O’Halloran, Martin, Brian McGinley, Raquel C. Conceição, Fearghal Morgan, Edward Jones, and Martin Glavin. “Spiking Neural Networks for Breast Cancer Classification in a Dielectrically Heterogeneous Breast.” *Progress in Electromagnetics Research* 113 (2011): 413–428.

Rosenblatt, Frank. “The Perceptron: A Perceiving and Recognizing Automaton.” Project Para Report No. 85-460-1, Cornell Aeronautical Laboratory, 1957.

Schliebs, Stefan, Haza Nuzly Abdull Hamed, and Nikola Kasabov. “Reservoir-Based Evolving Spiking Neural Network for Spatio-Temporal Pattern Recognition.” In *Neural Information Processing* (2011): 160–168.

Schliebs, Stefan, and Nikola Kasabov. “Evolving Spiking Neural Network—a Survey.” *Evolving Systems* 4, no. 2 (2013): 87–98.

Sinkevicius, S., R. Simutis, and V. Raudonis. “Monitoring of Humans Traffic Using Hierarchical Temporal Memory Algorithms.” *Electronics and Electrical Engineering* 115, no. 9 (2011): 91–96.

Soltic, Snjezana, and Nikola Kasabov. “Knowledge Extraction from Evolving Spiking Neural Networks with Rank Order Population Coding.” *International Journal of Neural Systems* 20, no. 6 (2010): 437–445.

Tank, D. W. and J. J. Hopfield. “Simple ‘Neural’ Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit.” *IEEE Transactions on Circuits and Systems* 33, no. 5 (1986): 533–541.

Thiruvarudchelvan, Vaenthan, James W. Crane, and Terry R. Bossomaier. “Analysis of Spikeprop Convergence with Alternative Spike Response Functions.” In *Proceedings of the 2013 IEEE Symposium on Foundations of Computational Intelligence*, 98–105. Piscataway, NJ: Institute of Electrical and Electronic Engineers, 2013.

Wysoski, Simei Gomes, Lubica Benuskova, and Nikola Kasabov. “On-Line Learning with Structural Adaptation in a Network of Spiking Neurons for Visual Pattern Recognition.” In *Artificial Neural Networks–ICANN 2006: Proceedings of the 16th International Conference, Athens, Greece, September 10–14, 2006*, edited by Stefanos D. Kollias, Andreas Stafylopatis, Włodzisław Duch, and Erkki Oja, 61–70. Berlin: Springer, 2006.

Wysoski, Simei Gomes, Lubica Benuskova, and Nikola Kasabov. “Text-Independent Speaker Authentication with Spiking Neural Networks.” In *Artificial Neural Networks–ICANN 2007: Proceedings of the 17th International Conference, Porto, Portugal, September 9–13, 2007*, edited by Joaquim Marques de Sá, Luís A. Alexandre, Włodzisław Duch, and Danilo Mandic, 758–767. Berlin: Springer, 2007.

Zhituo, Xia, Ruan Hao, and Wang Hao. “A Content-Based Image Retrieval System Using Multiple Hierarchical Temporal Memory Classifiers.” In *Proceedings of the Fifth International Symposium on Computational Intelligence and Design*, 438–441. Piscataway, NJ: Institute of Electrical and Electronic Engineers, 2012.

Zhuo, Wen, Zhiguo Cao, Yueming Qin, Zhenghong Yu, and Yang Xiao. “Image Classification Using HTM Cortical Learning Algorithms.” In *Proceedings of the 21st International Conference on Pattern Recognition*, 2452–2455. Piscataway, NJ: Institute of Electrical and Electronic Engineers, 2012.