2015

Spiking Neural Networks: Neuron Models, Plasticity, and Graph Applications

Shaun Donachy
Virginia Commonwealth University

SPIKING NEURAL NETWORKS: NEURON MODELS, PLASTICITY, AND GRAPH APPLICATIONS

A Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University.

by

SHAUN DONACHY

B.S. Computer Science, Virginia Commonwealth University, May 2013

Director: Krzysztof J. Cios,

Professor and Chair, Department of Computer Science

Virginia Commonwealth University

Richmond, Virginia

July, 2015

TABLE OF CONTENTS

Chapter — Page

Table of Contents i

List of Figures ii

Abstract v

1 Introduction 1

2 Models of a Single Neuron 3
- 2.1 McCulloch-Pitts Model 3
- 2.2 Hodgkin-Huxley Model 4
- 2.3 Integrate and Fire Model 6
- 2.4 Izhikevich Model 8

3 Neural Coding Techniques 14
- 3.1 Input Encoding 14
  - 3.1.1 Grandmother Cell and Distributed Representations 14
  - 3.1.2 Rate Coding 15
  - 3.1.3 Sine Wave Encoding 15
  - 3.1.4 Spike Density Encoding 15
  - 3.1.5 Temporal Encoding 16
  - 3.1.6 Synaptic Propagation Delay Encoding 16
  - 3.1.7 Rank Order Encoding 17
- 3.2 Output Decoding 17
  - 3.2.1 Temporal Decoding 17
  - 3.2.2 Race-Based Decoding 18

4 Konorski/Hebbian Learning Rules 20
- 4.1 Synaptic weight modification 20
- 4.2 Spike Timing-Dependent Plasticity 20
- 4.3 Synaptic Activity Plasticity Rule 22

5 Topology 24

6 Applications 25

6.1 Shortest Path 25

6.2 Clustering 28

7 Discussion 32

8 Conclusions 34

9 Future Work 35

Appendix A Abbreviations 36

References 37

Vita 40

LIST OF FIGURES

Figure Page

1 As seen in (Anderson, 1995) a McCulloch-Pitts neuron with threshold theta = 1 is capable of implementing an inclusive OR logic gate. 4

2 Model parameters for the Hodgkin-Huxley single neuron model as seen in (Gerstner and Kistler, 2002). 6

3 A diagram of the integrate-and-fire neuron model as seen in (Gerstner and Kistler, 2002). 7

4 An explanation of the Izhikevich neuron model. The mathematical model and parameter values for producing various neuron spiking behavior can be seen. (Electronic version of the figure and reproduction permissions are freely available at www.izhikevich.com) 9

5 Simulations of the Izhikevich neuron model with different parameters as seen in Izhikevich, 2004 to exhibit some of the spiking behaviors that are possible with the model. Each horizontal bar represents 20 ms of simulation. (Electronic version of the figure and reproduction permissions are freely available at www.izhikevich.com) 10

6 Comparison of different neuron models with their computational cost and biological plausibility as seen in (Izhikevich, 2004). 12

7 An illustration of temporal encoding and decoding as seen in (Paugam-Moisy and Bohte, 2012). 16

8 SNN network architecture used for MNIST handwritten digit classification as seen in (Beyeler, Dutt, and Krichmar, 2013). 19

9 STDP value vs Δt. 21

10 Example shapes of STDP windows as seen in (Paugam-Moisy and Bohte, 2012). Long term potentiation shown in red and long term depression shown in blue for excitatory synapses. In (4) a standard Hebbian learning rule is commonly applied to inhibitory synapses. 22

11 SAPR function values vs Δt. 23

12 (a). An example shortest path input graph with 9 nodes and 14 edges. The node with label '1' acts as the source node. (b). The shortest path solution from using either STDP or SAPR learning rules. The edges which do not lie on the shortest path from the source node, 1, to any other node are removed. 26

13 (a). An example clustering input graph with 8 nodes and 10 edges. (b). The clustering solution from using either STDP or SAPR learning rules. The edges which do not participate in cluster formation are removed. 29

14 (a). Clustering solution using either STDP or SAPR learning rules for the Iris data set reduced to two dimensions (b). The output solution where the data points are given different shapes based on class labels. The edges which do not participate in cluster formation are removed. . . 31

Abstract

SPIKING NEURAL NETWORKS: NEURON MODELS, PLASTICITY, AND GRAPH APPLICATIONS

By Shaun Donachy

A Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University.

Virginia Commonwealth University, 2015.

Director: Krzysztof J. Cios,

Professor and Chair, Department of Computer Science

Networks of spiking neurons can be used not only for brain modeling but also to solve graph problems (Sala and Cios, 1999). With the use of a computationally efficient Izhikevich neuron model combined with plasticity rules, the networks possess self-organizing characteristics. Two different time-based synaptic plasticity rules are used to adjust weights among nodes in a graph resulting in solutions to graph problems such as finding the shortest path and clustering.

CHAPTER 1

INTRODUCTION

A neural network consists of neurons and the connections among them. This view of the network is easily transformed into a graph with the neurons as nodes and the connections as edges, and is often represented in this way. Looking at the neural network from the perspective of a graph allows us to treat the network as a problem to be solved. In this setting, the resulting graph itself is the solution to a problem. This is a different view, since neural networks are typically used to find a function which maps known inputs to corresponding known outputs for finding a solution to a classification problem. Spiking Neural Networks (SNN) are considered the third generation of neural networks. Taking advantage of the precise timing of spikes generated by neurons they have greater computational power than the preceding forms (Gerstner and Kistler, 2002). Spikes are produced when a neuron receives sufficient input to excite the neuron and generate an output current.

An SNN can be used to solve graph problems by exploiting their ability to self-organize. This self-organization is formed from local learning based on information from the timing of spikes between neurons. Although there are many different neuron models formulated and used within SNN, the Izhikevich neuron model has been shown to be efficient computationally Izhikevich, 2003. Here we will show how to use the Izhikevich neuron model and two synaptic plasticity rules to produce solutions to two graph problems: that of finding the shortest path and clustering of data points (the nodes of a graph).

The human brain, particularly the neo-cortex, is proof of the computational power of a vast network of neurons. The complexity of neuron structure and connections has made it difficult to discover the inner workings of such huge networks. Neuroscientists, computer scientists, and scientists among other disciplines have been working for many years to uncover the unifying algorithms of the neocortex. While much progress has been made towards understanding neuronal systems, the exact methods employed to perform such incredible computation, memorization, and learning capabilities of the neocortex remain elusive. Thus, further investigations which intend to uncover the secrets of the brain are well-founded.

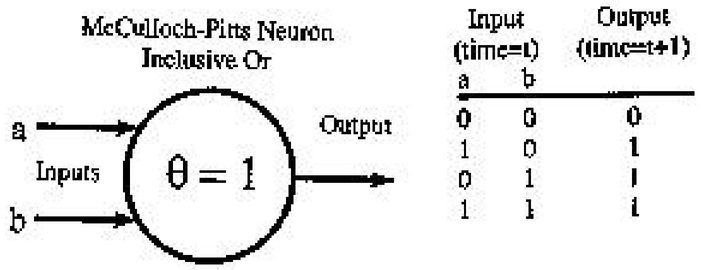
CHAPTER 2

MODELS OF A SINGLE NEURON

A spiking neural network consists of many neurons and the connections among them. This chapter discusses individual neuron models. Multiple neuron models have been introduced based on findings from neuroscience research with varying degrees of biological plausibility. Here, a selection of important models will be described and their inherent attributes discussed.

2.1 McCulloch-Pitts Model

McCulloch and Pitts were interested in investigating the components of the nervous system to determine how it might be working. The computational capabilities of their two-state neuron model were reported in (McCulloch and Pitts, 1943). This neuron model has become known as the McCulloch-Pitts neuron model, or threshold logic unit. In such a model the neuron is only capable of two states, active and inactive. Each neuron has a fixed threshold for which it transfers from an inactive state to an active state, which is equivalent to the Heaviside step function. Some of the synapses between neurons may be inhibitory such that if a neuron is connected to an active inhibitory synapse it may not, itself, become active. The most dramatic result of the work of McCulloch and Pitts was that any finite logical expression can be reified in a network of McCulloch-Pitts neurons. This was particularly interesting to computer scientists and played a role in the development of what is now called the Von Neumann architecture (Anderson, 1995). The McCulloch-Pitts neuron model does not attempt to exhibit biologically plausible properties. Significant advance-



Input (time=t)		Output (time=t+1)
a	b	
0	0	0
1	0	1
0	1	1
1	1	1

Fig. 1. As seen in (Anderson, 1995) a McCulloch-Pitts neuron with threshold theta = 1 is capable of implementing an inclusive OR logic gate.

ments in neuroscience suggest that human brains and the individual neurons which constitute them do not perform formal logic and symbolic operations as the modern digital computer does. Thus, to simulate neuronal activity observed in the human neocortex, more complex neuron models are required.

2.2 Hodgkin-Huxley Model

On the opposite end of the spectrum of biological plausibility lies the well-known Hodgkin-Huxley neuron model. This model represents a relatively high degree of biological accuracy describing neuron function. The price for this level of biological accuracy is a high computational cost. This model has been widely and successfully used in neuroscience research. The implementation of large numbers of neurons of this type for computational simulation quickly becomes intractable, though.

Hodgkin and Huxley performed experiments on the giant axon of the squid in 1952. They found three different ion channels, one for sodium ions, one for potassium ions and a third which handles other types of channels and is given the name leakage channel (Gerstner and Kistler, 2002). The cell membrane of the neuron is semi-permeable and the flow of ions across the membrane determines potential internal to the cell with respect to the fluid external to the cell. This membrane therefore acts as a capacitor in an electrical circuit. The term u refers to the potential across this membrane. Hodgkin and Huxley formulated three current components of their model as shown in equation 2.1. The channels are characterized by their respective resistances. When the sodium and potassium channels are open they have maximum conductance of g_{Na} and g_K. The terms m and h control the sodium channels while the potassium channels are controlled by the n term. Reversal potentials are given by E_{Na}, E_K, and E_L.

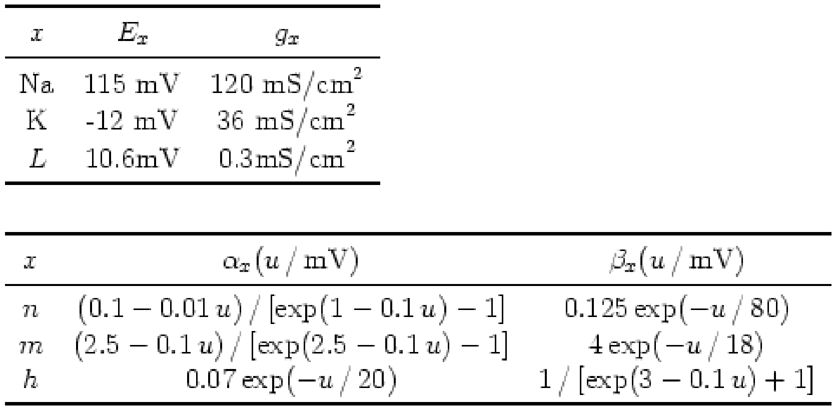
$$\sum_k I_k = g_{Na} m^3 h (u - E_{Na}) + g_K n^4 (u - E_k) + g_L (u - E_L) \tag{2.1}$$

Gating variables m, n, and h are modified according to the differential equations of equation 2.2. These gating variables describe the probability that a particular channel is open, since normally some of the channels are blocked.

$$\begin{aligned} \dot{m} &= \alpha_m(u)(1-m) - \beta_m(u)m \\ \dot{n} &= \alpha_n(u)(1-n) - \beta_n(u)n \\ \dot{h} &= \alpha_h(u)(1-h) - \beta_h(u)h \end{aligned} \tag{2.2}$$

The parameters of the model are given in Figure 2. It should be noted that the parameters shown expect the neuron resting potential to be 0 V. The resting potential is now understood to be -65 mV, thus the model parameters should be shifted by -65 mV to achieve this resting potential.

This model is biologically accurate for the subject of Hodgkin and Huxley's work, the squid. However, there are many more electro-physiological properties in cortical neurons of vertebrates. Detailed models for these types of neurons have been developed that describe additional channels. Such models are, of course, even more computationally demanding than the Hodgkin-Huxley model for simulation purposes.



x	E_x	g_x
Na	115 mV	120 mS/cm^2
K	-12 mV	36 mS/cm^2
L	10.6mV	0.3mS/cm^2

x	$\alpha_x(u\,/\,\mathrm{mV})$	$\beta_x(u\,/\,\mathrm{mV})$
n	$(0.1 - 0.01\,u)\,/\,[\exp(1 - 0.1\,u) - 1]$	$0.125\exp(-u\,/\,80)$
m	$(2.5 - 0.1\,u)\,/\,[\exp(2.5 - 0.1\,u) - 1]$	$4\exp(-u\,/\,18)$
h	$0.07\exp(-u\,/\,20)$	$1\,/\,[\exp(3 - 0.1\,u) + 1]$

Fig. 2. Model parameters for the Hodgkin-Huxley single neuron model as seen in (Gerstner and Kistler, 2002).

2.3 Integrate and Fire Model

An integrate and fire neuron model originally presented by Lapicque in 1907 retains some neuron properties and the membrane potential but simplifies spike generation (Abbott, 1999). The best known model is named the leaky integrate and fire model. This model is essentially a capacitor C in parallel with a resistor R driven by a current I(t); see Figure 3. The addition of the leak is an improvement by permitting the neuron's membrane voltage to leak and therefore decay to an equilibrium state in the absence of stimulation. The neuron model can be seen in equation 2.3 where V_m, C_m, and R_m refer to membrane voltage, capacitance, and resistance, respectively. The addition of the $V_m(t)/R_m$ term forces the input current to overcome a threshold in order to produce a spike as seen in equation 2.4. If this threshold is not met and the input current ceases, then the potential will simply leak out.

$$I(t) - \frac{V_m(t)}{R_m} = C_m \frac{dV_m(t)}{dt} \tag{2.3}$$

$$I_{th} = \frac{V_{th}}{R_m} \tag{2.4}$$

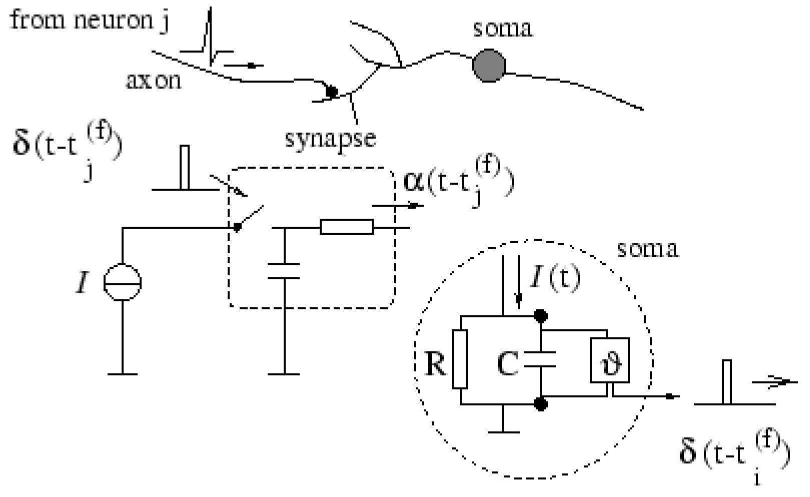


Fig. 3. A diagram of the integrate-and-fire neuron model as seen in (Gerstner and Kistler, 2002).

Multiple variations of the integrate and fire model have been developed. MacGregor increased the biological accuracy of the model including an absolute and relative refractory period for each neuron (MacGregor, 1987). An absolute refractory period is a phenomenon of naturally occurring neurons which prevents them from firing for some period after their last fire event. Similarly, a relative refractory period occurs after the absolute refractory period. During the relative refractory period the neuron is only capable of firing under very high stimulation conditions. MacGregor's model accounts for membrane potential and the potassium channel response.

The modified MacGregor model is described by equations 2.5 - 2.8. Equation 2.5 shows how spikes are generated where E is membrane potential and T_h is the threshold. Equation 2.6 shows the refractory properties of the neuron; G_K is the potassium channel conductance. Equation 2.7 describes the threshold accommodation and equation 2.8 describes the transmembrane potential.

$$S = \begin{cases} 1 & \text{if } E \geq T_h \\ 0 & \text{if } E < T_h \end{cases} \tag{2.5}$$

$$\frac{dG_k}{dt} = \frac{-G_k + B \cdot S}{T_{GK}} \tag{2.6}$$

$$\frac{dT_h}{dt} = \frac{-(T_h - T_{h0}) + c \cdot E}{T_{th}} \tag{2.7}$$

$$\frac{dE}{dt} = \frac{-E + G_k \cdot (E_K - E) + G_e \cdot (E_e - E) + G_i \cdot (E_i - E) + SCN + N}{T_{mem}} \tag{2.8}$$

2.4 Izhikevich Model

The Izhikevich neuron model was developed by Eugene Izhikevich (Izhikevich, 2003). It is among the class of neuron models which seek to reduce the complexity of the highly accurate Hodgkin-Huxley model from four differential equations down to two. Izhikevich has shown that his neuron model is capable of reproducing many of the output behaviors observed with biological neurons, thus similar in this regard to the Hodgkin-Huxley model (Izhikevich, 2004). As can be seen in Figure 6, the number of floating point operations (flops) required to produce these behaviors with the Izhikevich neuron model is 13 whereas the Hodgkin-Huxley neuron model requires 1200.

The Izhikevich model is described by equations 2.9 - 2.11. Here, v is the membrane potential variable and u is the membrane recovery variable that provides negative feedback to v, both of which are dimensionless; $v' = \frac{dv}{dt}$ and $u' = \frac{du}{dt}$ and t is time. In this model there are four dimensionless parameters: a, b, c, and d. The parameter a describes the scale of time that u operates on. The parameter b describes how sensitive u is to fluctuations in v below the firing threshold. The parameter c is used to define the reset potential of v after a spike is generated. The parameter d describes the reset of the variable u after a spike is generated. The section $0.04v^2 + 5v + 140$

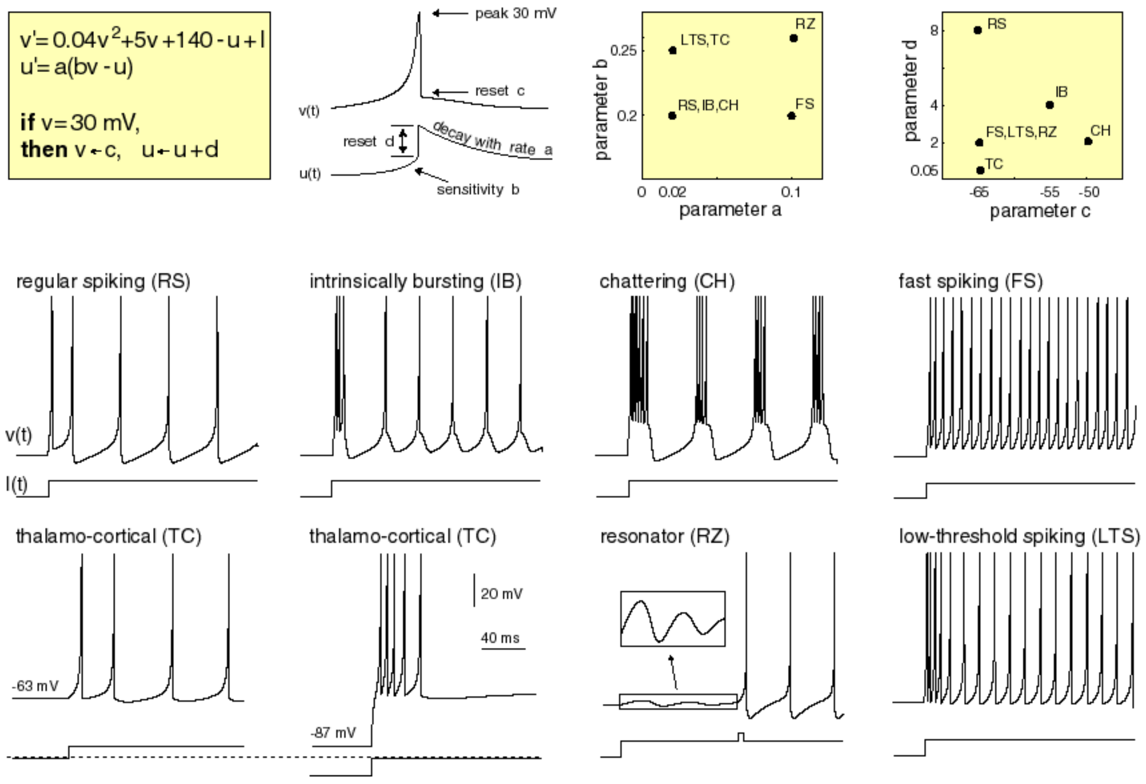


Fig. 4. An explanation of the Izhikevich neuron model. The mathematical model and parameter values for producing various neuron spiking behavior can be seen. (Electronic version of the figure and reproduction permissions are freely available at www.izhikevich.com)

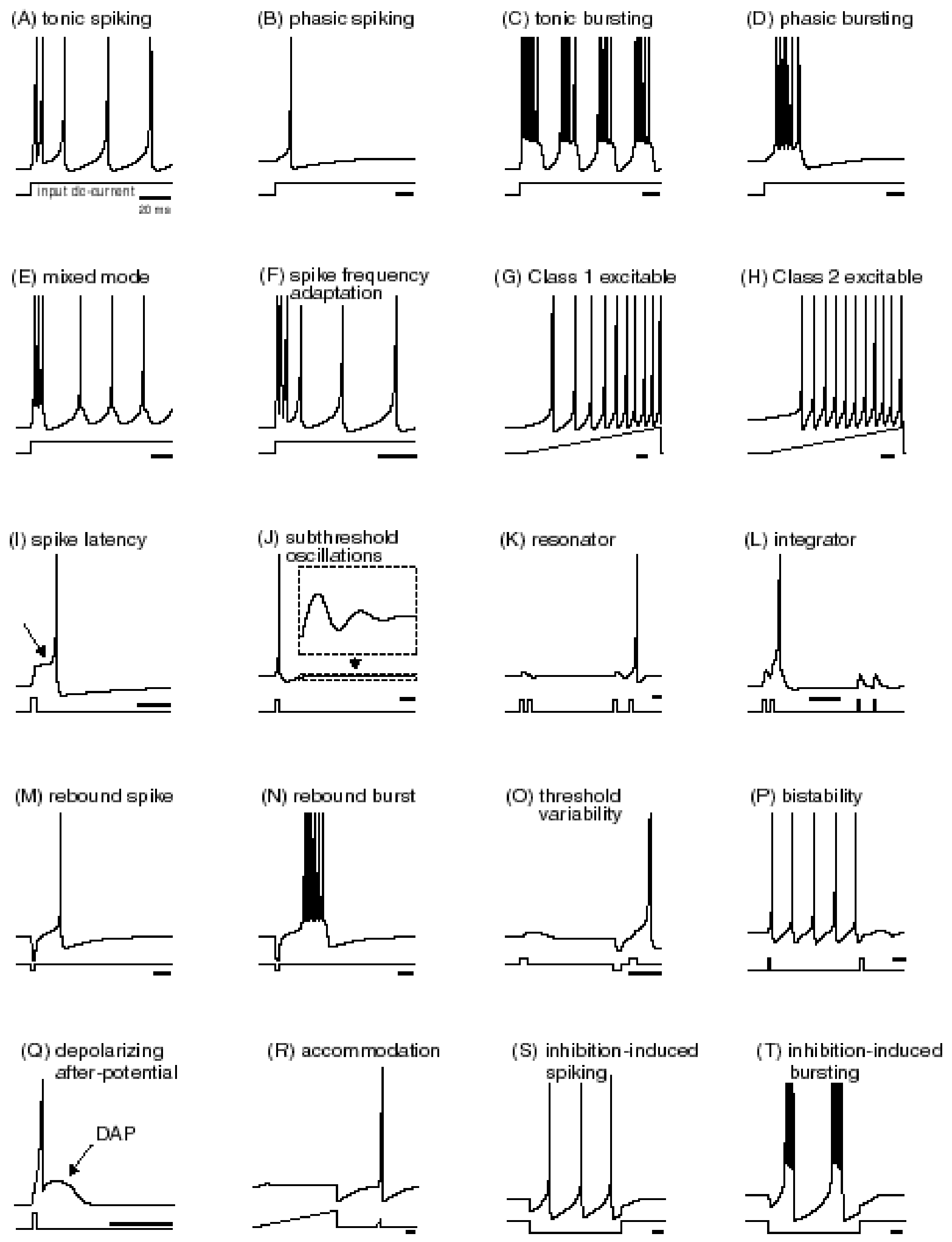


Fig. 5. Simulations of the Izhikevich neuron model with different parameters as seen in Izhikevich, 2004 to exhibit some of the spiking behaviors that are possible with the model. Each horizontal bar represents 20 ms of simulation. (Electronic version of the figure and reproduction permissions are freely available at www.izhikevich.com)

was determined by fitting the model dynamics to operate on the voltage scale of mv and the time scale of ms. This aspect of the model may be modified to fit other scales Izhikevich, 2003. The model incorporates the input current to the system using parameter I. By modifying the model parameters a, b, c, and d it is possible to produce different kinds of neuron spiking behavior as demonstrated in Figure 4 and Figure 5.

$$v' = 0.04v^2 + 5v + 140 - u + I \tag{2.9}$$

$$u' = a(bv - u) \tag{2.10}$$

$$\text{if} \quad v \geq 30 \quad \text{mV, then} \begin{cases} v \leftarrow & c \\ u \leftarrow & u + d \end{cases} \tag{2.11}$$

The Izhikevich neuron model does not model the absolute or relative refractory periods, which reduces the biological plausibility of the model and may lead to unrealistic neuron behaviors under certain conditions. A solution to this problem was given by (Strack, Jacobs, and Cios, 2013) by preventing the neuron from generating an output spike until a specified time after a spike has occurred. To incorporate an absolute refractory period, when v reaches the threshold at time $t = t^f$, the dynamics are interrupted (equations 2.9-2.11) until time $t^f + \Delta_{(abs)}$. Therefore, equation 2.11 was modified to have an additional constraint as shown in equation 2.12.

$$\text{if} \quad v \geq 30 \quad \text{mV and } t - t_{prev} \geq \Delta_{(abs)}, \text{ then} \begin{cases} v \leftarrow & c \\ u \leftarrow & u + d \end{cases}$$

$$\text{else if } v \geq 30 \text{ mV then, } v \leftarrow 30 \tag{2.12}$$

Since the Izhikevich neuron model is capable of biological realism and computa-

Models	biophysically meaningful	tonic spiking	phasic spiking	tonic bursting	phasic bursting	mixed mode	spike frequency adaptation	class 1 excitable	class 2 excitable	spike latency	subthreshold oscillations	resonator	integrator	rebound spike	rebound burst	threshold variability	bistability	DAP	accommodation	inhibition-indiced spiking	inhibition-induced bursting	chaos	# of FLOPS
integrate-and-fire	-	+	-	-	-	-	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	5
integrate-and-fire with adapt.	-	+	-	-	-	-	+	+	-	-	-	-	+	-	-	-	-	+	-	-	-	-	10
integrate-and-fire-or-burst	-	+	+		+	-	+	+	-	-	-	-	+	+	+	-	+	+	-	-	-		13
resonate-and-fire	-	+	+	-	-	-	-	+	+	-	+	+	+	+	-	-	+	+	+	-	-	+	10
quadratic integrate-and-fire	-	+	-	-	-	-	-	+	-	+	-	-	+	-	-	+	+	-	-	-	-	-	7
Izhikevich (2003)	-	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	13
FitzHugh-Nagumo	-	+	+	-		-	-	+	-	+	+	+	-	+	-	+	+	-	+	+	-	-	72
Hindmarsh-Rose	-	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	120
Morris-Lecar	+	+	+	-		-	-	+	+	+	+	+	+	+		+	+	-	+	+	-	-	600
Wilson	-	+	+	+			+	+	+	+	+	+	+	+	+	+		+	+				180
Hodgkin-Huxley	+	+	+	+			+	+	+	+	+	+	+	+	+	+	+	+	+	+		+	1200

Fig. 6. Comparison of different neuron models with their computational cost and biological plausibility as seen in (Izhikevich, 2004).

tional efficiency it stands out as a candidate for large computational simulations. For example, Izhikevich implemented a network of 10,000 neurons with 1,000,000 random synapse connections using a 1GHz desktop PC in real-time (Izhikevich, 2003).

CHAPTER 3

NEURAL CODING TECHNIQUES

3.1 Input Encoding

Spiking neural networks differ significantly from early neural network paradigms. The presence and precise timing of spikes encapsulates meaning. Different techniques are therefore required to submit a stimulus to the network. This chapter discusses techniques of transforming data into a suitable form for network submission.

3.1.1 Grandmother Cell and Distributed Representations

An important consideration is how to represent information in a system(Anderson, 1995). One type of representation is called grandmother cell. Grandmother cell representation gets its name from an assumption that a single cell becomes active whenever a person sees their grandmother. This example is reinforced by neuroscience studies of place cells (Martig and Mizumori, 2010). Place cells become active in the hippocampus when the subject navigates through a particular area.

A second type is termed distributed representation. The semantic meaning of a distributed representation by definition cannot be interpreted by observing a single neuron. Extending the grandmother cell representation to a pool of neurons produces a distributed representation. Thus the data can be represented in a spatial fashion. Similar inputs could be represented as spatially similar using such a technique. The challenge becomes determining a way in which similar inputs can be transformed to spatially similar representations and therefore this may only be relevant for ordinal data types.

3.1.2 Rate Coding

The notion of rate coding assumes that a significant portion of information is encoded in the firing rate or frequency of neurons (Meftah, Lezoray, and Chaturvedi, 2012). Probabilistic firing rates can be used to encode information. This technique of encoding has been criticized for several reasons (Gerstner and Kistler, 2002). In particular, behavioral experiments show that human response times to visual stimuli are very short, which would not leave enough time for an average firing rate to be determined by the system. Thus, an additional information source is likely.

3.1.3 Sine Wave Encoding

In the supervised classification problem there exists input features which must be transformed to an acceptable format for the SNN. One method of transformation as seen in (Shin et al., 2010) is sine wave encoding. The raw feature values are normalized and then the amplitude of the sine wave is adjusted based on the normalized feature value. This signal is presented to the network for some portion of the total simulation time. Since the amplitude of the signal is encoding the information this technique is very similar to the continuous inputs of traditional neural networks.

3.1.4 Spike Density Encoding

A spike density code is a form of population coding that measures how many neurons are firing. So, a pool of neurons could be set up such that neurons fire stochastically relative to the size of the input value. Therefore, the density of the spikes generated by the pool as an entire unit encodes the input information (Paugam-Moisy and Bohte, 2012). One issue with this method is the apparent inefficiency of using such a large number of neurons to encode a relatively few number of inputs.

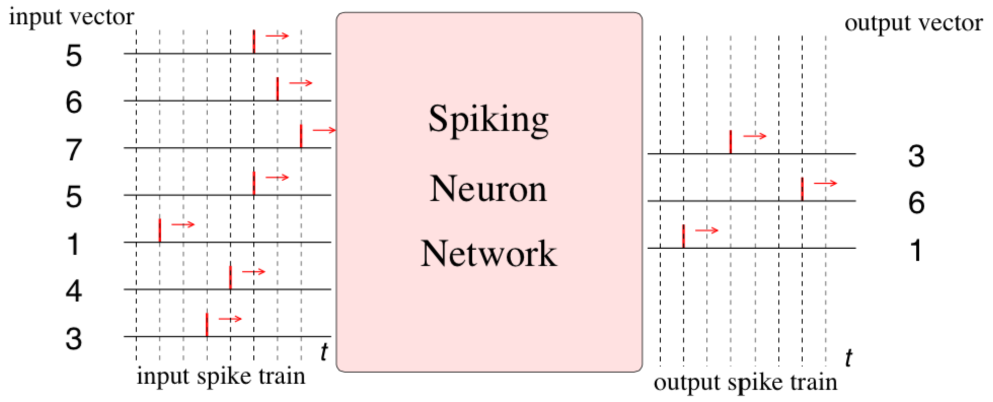


Fig. 7. An illustration of temporal encoding and decoding as seen in (Paugam-Moisy and Bohte, 2012).

The increased number of neurons implies increased amounts of synaptic connections and there for energy within the system to represent the signal.

3.1.5 Temporal Encoding

Temporal coding is a way of encoding input information as time differentials. This technique may also be called latency coding or time-to-first-spike coding (Paugam-Moisy and Bohte, 2012; Gerstner and Kistler, 2002). As can be seen in Figure 7 the timing of the spikes is varied by some delay relative to the strength of the inputs. This technique takes advantage of the SNN's ability to encode information temporally. The biological relevance of this technique is well-founded by the observations made in the visual system. More intense signals are seen as spike transmissions earlier than less intense inputs.

3.1.6 Synaptic Propagation Delay Encoding

Encoding the edge weights of a graph as synaptic propagation delays encodes information temporally. The timing of spikes is influenced heavily by introducing a delay between neurons. This method is based on biological networks which possess a variable delay between neurons based on the length of the dendrites connecting neurons.

3.1.7 Rank Order Encoding

Coding by rank order is a technique where the order of the spikes is used to encode information. Such a coding scheme would require the mapping of input values to a rank order over n neurons. The spike emissions are among one of the $n!$ possible orderings of n neurons. Therefore, $\log_2(n!)$ bits may be used to represent such an ordering. Such a capacity is optimistic as using this method within computer simulations necessitates the ability to differentiate between two spike timings (Paugam-Moisy and Bohte, 2012).

3.2 Output Decoding

In addition to using special techniques to encode information to a usable form for the SNN, specific techniques will be required for decoding the output from an SNN. This section will look at some techniques that are used to decode such information into a form suitable for classification decision making.

3.2.1 Temporal Decoding

Similar to temporal encoding as seen in section 3.1.5, temporal decoding transforms the first spike time information of individual neurons into an output vector (see Figure 7). To transform this output into a format capable of decision making, the output vector can be used to form signatures as in (Shin et al., 2010). The signatures are formed by calculating an average of spike outputs generated by all inputs of a selected class. Once signatures are computed a distance based comparison may be used to generate a decision.

Alternatively, a system may use such an output vector as input to an optimization technique which minimizes classification error mapping the vector to a class label. In this scenario the SNN becomes a feature extraction system.

3.2.2 Race-Based Decoding

Another method which considers spike timing as essential information is a race-based decoding mechanism. In this paradigm a single neuron or a pool of neurons is set up for each class label. The role of these neurons is to identify which class type the input belongs to by firing before the members of the other neuron pools.

To achieve this there is some form of feedback to the decision nodes about the class label of input being submitted to the network. It may come in the form of supervised training of weights leading into the decision pools.

In (Beyeler, Dutt, and Krichmar, 2013) a clever method of feedback is used, called supervised Hebbian learning (Kasinski and Ponulak, 2006). The pool of decision neurons have recurrent excitatory connections among themselves as well as inhibitory connections to other neuron decision pools (see Figure 8). When a stimulus is presented to the network a single neuron in the pool corresponding to the correct label is also stimulated. Combined with Hebbian plasticity (see section 4.2 and 4.3) this teacher signal causes increased excitement, thus generating faster and more frequent spikes among the correct pool of neurons.

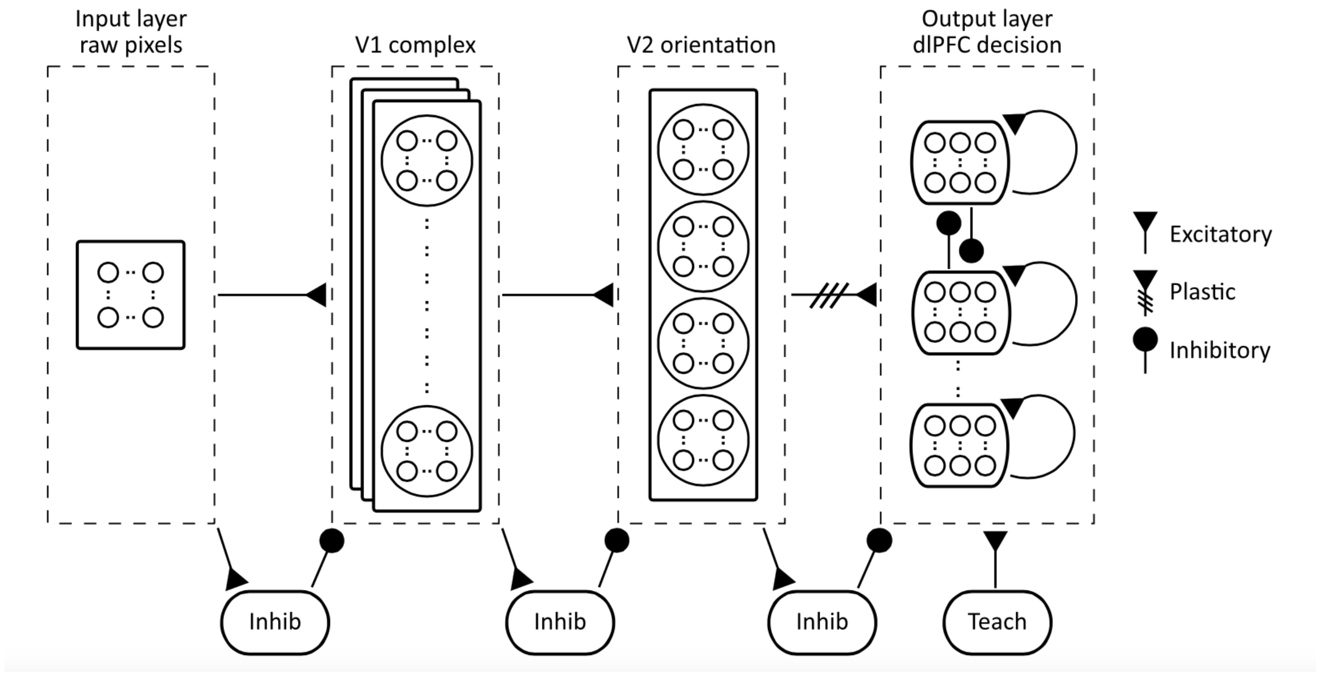


Fig. 8. SNN network architecture used for MNIST handwritten digit classification as seen in (Beyeler, Dutt, and Krichmar, 2013).

CHAPTER 4

KONORSKI/HEBBIAN LEARNING RULES

Biologically founded neural networks like SNN are capable of self-learning from their input. The network's behavior is shaped by inputs that it has received over time. The concept of synaptic plasticity was first presented by Konorski (Konorski, 1948) in 1948 and later by Hebb (Hebb, 1949) in 1949 and has come to be known as Hebbian style learning. Here we take a closer look at two different synaptic plasticity rules that may be used for networks of spiking neurons.

4.1 Synaptic weight modification

A common method used to modify the connection weights is based on the observations of Konorski and Hebb (Hebb, 1949; Konorski, 1948). Such synaptic weight modifications are termed *plasticity*. The Konorski/Hebbian learning rule changes the weight of a synaptic connection based upon the pre- and post-synaptic neuron activity. When the firing of a pre-synaptic neuron regularly participates in the firing of a post-synaptic neuron, the strength of the action of the pre-synaptic neuron onto the post-synaptic neuron increases. Conversely, if the pre-synaptic neuron regularly fires after the post-synaptic neuron, the strength of the action from the pre-synaptic neuron onto the post-synaptic neuron decreases.

4.2 Spike Timing-Dependent Plasticity

Spike timing-dependent plasticity (STDP) is a temporally asymmetric form of Konorski/Hebbian learning that modifies synaptic connections between pre- and post-

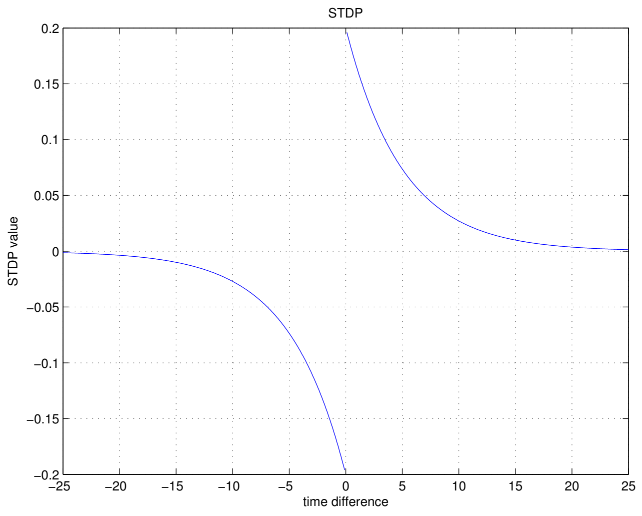


Fig. 9. STDP value vs Δt.

synaptic neurons that are temporally correlated. In addition, a winner takes all approach is often used where only the weight of the first post-synaptic neuron to fire is updated. Such plasticity is believed to be an underlying learning and information storage mechanism and possibly contributes to the development of neuronal circuits during brain development (Sjostrom and Gerstner, 2010). For these reasons it is relevant to many SNN implementations.

Equations 4.2 and 4.3 define the STDP weight update rule. The Δt term in equation 4.1 is the difference in firing times between pre- and post-synaptic neurons. To emphasize that weight updates occur only when a post-synaptic neuron fires $w(new)$ has a $t^{f_{post}}$ subscript. Separate τ time constants (τ_+, τ_-) are used for $\Delta t > 0$ and $\Delta t \leq 0$, respectively, and α is the learning rate.

For computer simulations taking advantage of any STDP-like rule it is necessary to prevent the synapse weights from exceeding a maximum value or minimum value. Otherwise, weight modifications will diverge to $\pm\infty$ (Paugam-Moisy and Bohte, 2012). It is known that using STDP to modify weights will result in a bimodal

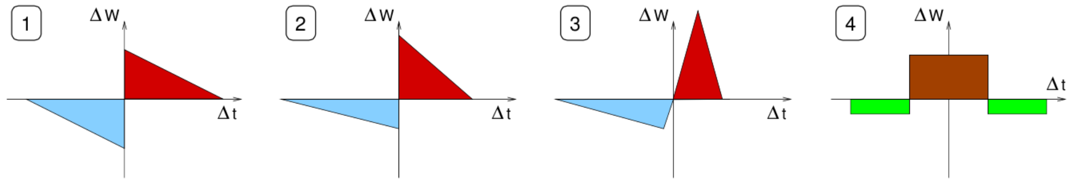


Fig. 10. Example shapes of STDP windows as seen in (Paugam-Moisy and Bohte, 2012). Long term potentiation shown in red and long term depression shown in blue for excitatory synapses. In (4) a standard Hebbian learning rule is commonly applied to inhibitory synapses.

distribution of weights (Legenstein, Naeger, and Maass, 2005). The weights will tend toward either the maximum value or the minimum value.

$$\Delta t = t^{f_{post}} - t^{f_{pre}} \tag{4.1}$$

$$w_{t^{f_{post}}}(new) = w(old) + STDP(\Delta t) \tag{4.2}$$

$$STDP(\Delta t) = \begin{cases} \alpha_+ e^{-\Delta t/\tau_+} & \text{if} \Delta t > 0 \\ -\alpha_- e^{\Delta t/\tau_-} & \text{if} \Delta t \leq 0 \end{cases} \tag{4.3}$$

4.3 Synaptic Activity Plasticity Rule

The Synaptic Activity Plasticity Rule (SAPR) is a temporally symmetric form of Konorski/Hebbian learning. The synaptic connection strength in SAPR is modified using an update function that takes advantage of the membrane potential of the post-synaptic neuron (Swiercz et al., 2006). In contrast to the STDP function, SAPR is continuous when the time difference between pre- and post-synaptic firing times is zero, where STDP is undefined. Values for STDP approach ±∞ as the time difference nears zero, whereas SAPR is bounded to a finite range.

Equation 4.5 is used to calculate Excitatory Post-Synaptic Potential (EPSP),

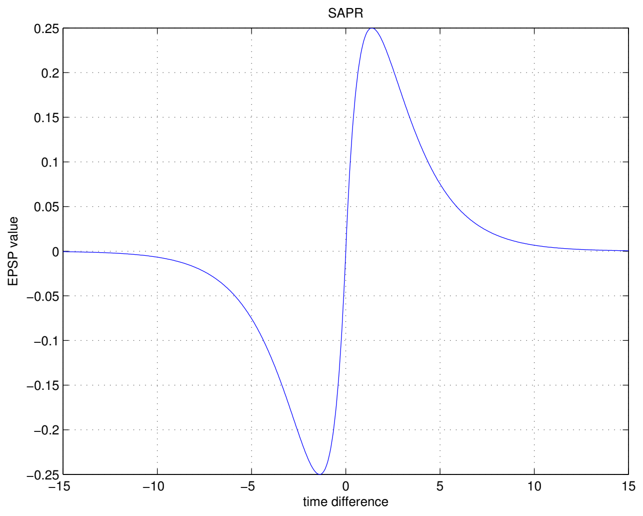


Fig. 11. SAPR function values vs Δt.

given the learning rate α , and synapse time constants τ_d and τ_r. With the time difference between when the neuron fired (t^f) and the current time step (t) the membrane potential can be computed. This value is used to update the synaptic weights between two neurons by equation 4.4. If the post-synaptic neuron fires after the pre-synaptic neuron fired then the pre-synaptic neuron contributes positively to the firing of the post-synaptic neuron and its weight is increased, otherwise the weight is decreased.

$$w_{t^{f_{post}}}(new) = \begin{cases} w(old) + SAPR(\Delta t) & \text{if} \Delta t > 0 \\ w(old) - SAPR(abs(\Delta t)) & \text{if} \Delta t \leq 0 \end{cases} \tag{4.4}$$

$$SAPR(\Delta t) = \alpha \left[e^{-\left(\frac{\Delta t}{\tau_d}\right)} - e^{-\left(\frac{\Delta t}{\tau_r}\right)} \right] (t - t^f) = EPSP \tag{4.5}$$

CHAPTER 5

TOPOLOGY

The way in which the neurons within a neural network are connected constitutes a neural network topology (Fiesler, 1996). Every neuron will have a connection to at least one other neuron, and the way these connections are determined is one of the fundamental factors of how the network will behave. The neurons may be organized into layers and have intra- and inter-layer connections.

For the applications presented in this work, the translation from an input graph to an SNN topology is straight-forward. There will only be one layer of neurons and each node in the input graph becomes a neuron in the SNN. The edges of the input graph become connections (synapses) among the neurons in the SNN as explained in the following section.

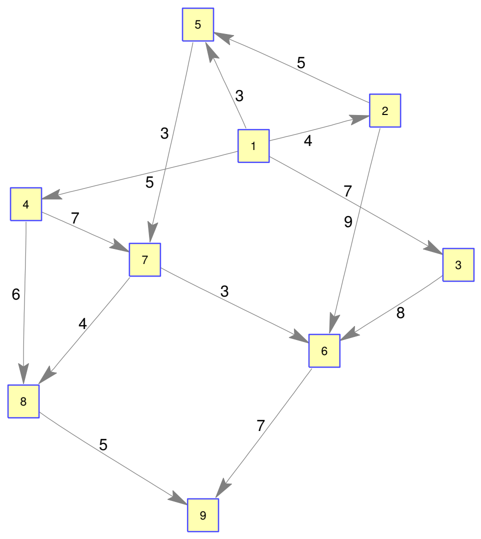
CHAPTER 6

APPLICATIONS

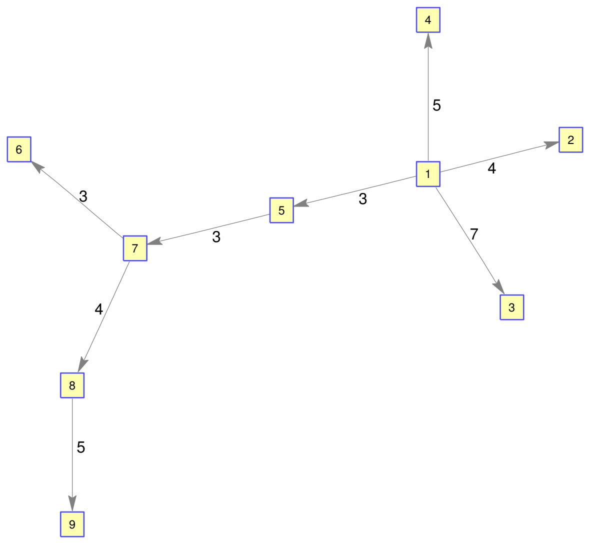
6.1 Shortest Path

This investigation looks at the one-to-all shortest path problem, that is: given a source node, find the shortest paths to all other nodes. Using the Izhikevich neuron model and both STDP and SAPR, we will describe how to accomplish this task using an SNN.

We begin by assigning the nodes in the graph to neurons in the SNN and the edges as synapses. The edge values in the graph become respective synaptic propagation delays between neurons in the SNN. All of the synaptic weights are initialized to 0.5 and restricted to the range [0.01, 1]. In the case of STDP this is a hard limit imposed by a step function, and in the case of SAPR a sigmoid function is used to meet this constraint. The source node is then stimulated with current I in equation 2.11 that is large enough to elicit a spike response. The initial weights and spike currents must combine in such a way that a spike response is created in each and all post-synaptic neurons. Weight values may increase or decrease but it is known that they will eventually diverge to the maximum or minimum value (Gerstner and Kistler, 2002). All neurons are inhibited from spiking a second time after they have spiked once and produced an action potential. Once all neurons have fired, a cycle is concluded. Since it is known that the weight values will diverge to minimum or maximum, or in some cases remain the same, the simulation stops if the the weights have climbed sufficiently high, fallen sufficiently low, or remain unchanged. Often, this will take only one cycle.



(a)



(b)

Fig. 12. (a). An example shortest path input graph with 9 nodes and 14 edges. The node with label '1' acts as the source node. (b). The shortest path solution from using either STDP or SAPR learning rules. The edges which do not lie on the shortest path from the source node, 1, to any other node are removed.

If the pre-synaptic neuron fires before the post synaptic neuron, then the synaptic strength between the neurons is increased. But, because winner-takes-all is being used, only the neuron whose action potential arrives first at the post-synaptic neuron will have the synaptic strength between itself and the post-synaptic neuron increased. All others' synaptic connection strength to the post synaptic neuron will remain unchanged. If the pre-synaptic neuron fires after the post-synaptic neuron fires then the synaptic strength between the neurons is decreased. In this case all neurons whose action potentials arrive after the post-synaptic neuron after it has fired have their synaptic strengths decreased.

Upon conclusion of the simulation the synaptic connections whose weights are near the maximum weight value represent the set of edges that constitute the shortest paths in the graph from the source node.

As an example, Figure 12(a) shows a graph with 9 nodes and 14 edges. The edge weights shown become propagation delays between synapses in an SNN. This means that an action potential produced at time t will be supplied as input current to the post-synaptic neuron at time $t + d$ where d is the propagation delay. The source neuron, 1, is stimulated and produces an action potential to be received by its four connected neurons which receive input stimulation after their respective propagation delays have passed. Upon receipt of the input stimulation, the edge weights will be positively modified based on which neuron's action potential arrived first using equation 4.3 in the case of STDP and equation 4.4 for SAPR. After iterating over individual time-steps until each neuron fires, forming a cycle, the neurons are reset to a resting state. The cycles continue until all of the edge weights have settled. The result is shown in Figure 12(b) where it can be seen that only the edges along the shortest paths from the source node, in this case node 1, to all other nodes remain.

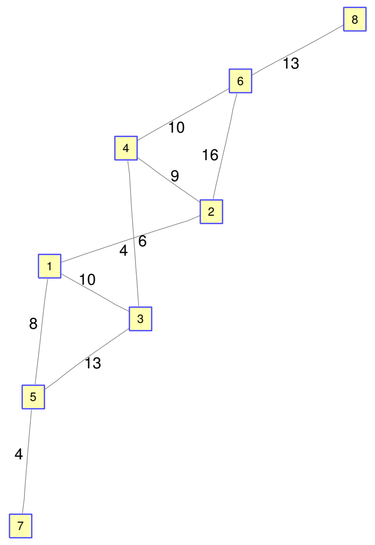
6.2 Clustering

The goal of clustering within a graph is to group together similar nodes based on their edge distances to the adjacent nodes. To solve this problem in an unsupervised fashion (without a priori specifying the number of clusters) a network of spiking neurons using the Izhikevich neuron model, and STDP and SAPR plasticity rules are used.

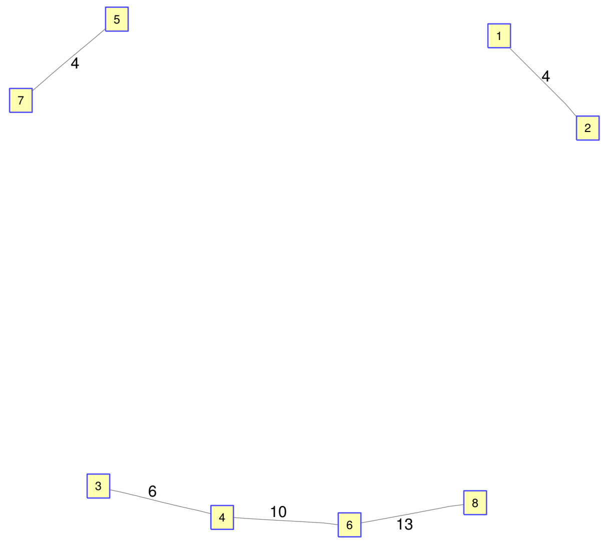
The initial translation of the graph to the SNN is similar to that of the shortest path explained above. Synaptic weights are again initialized to 0.5 and restricted to the range [0.01, 1]. However instead of stimulating a single node during the first time step, all neurons are stimulated with the exception of one. All neurons take a turn as the non-stimulated neuron in a round-robin fashion. Time advances forward in one millisecond steps until the non-stimulated node fires, thus completing a cycle. All neurons are inhibited from firing more than once during a cycle. Once all neurons have taken their turn as the dormant node for a cycle the simulation completes and the remaining edges form clusters.

Unlike the shortest path simulation in which all edges are directed, graph clustering requires all edges in the input graph to be bi-directional. This becomes an issue because the synaptic weights are updated independently of one another, yet they logically represent the same edge in the associated graph. To address this issue, only incoming edges are updated when a post-synaptic neuron fires and upon conclusion of the simulation, if the pair of synaptic weights differ, then the higher weight is used to determine which edges survive.

Figure 13(a) shows an example graph consisting of 8 nodes and 10 edges. The graph is translated into an SNN structure as in the shortest path example where the edge weights become propagation delays. Neuron number 1 is initially dormant at



(a)

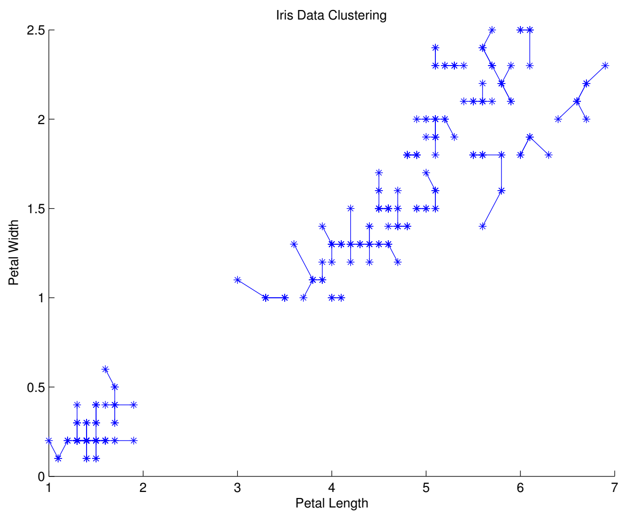


(b)

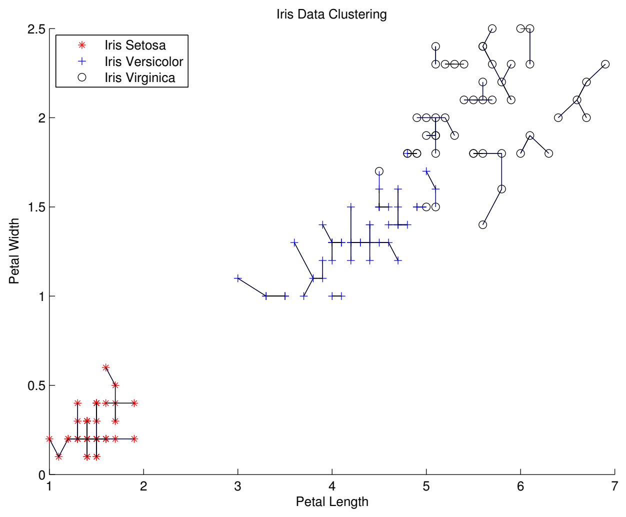
Fig. 13. (a). An example clustering input graph with 8 nodes and 10 edges. (b). The clustering solution from using either STDP or SAPR learning rules. The edges which do not participate in cluster formation are removed.

time step 1 while all other nodes are stimulated. When an action potential arrives and elicits a spike response at neuron 1 then the incoming synapses to neuron 1, in this case synapses originating at neurons 2, 3, and 5 have their weights updated. The action potential from node 2 should arrive first and thus the synaptic weight between neuron 2 and neuron 1 will be positively modified. Neurons 2-8 will take their turn as the dormant node and synapses will be updated accordingly. After 8 cycles are completed the synapses which are near the maximum weight value are kept. These represent the edges that constitute the clustering output as seen in Figure 13(b).

Next, we show how clustering is performed on real data. Fisher's original data set consists of 50 samples each of three different species of Iris flowers and is described by four features: sepal length, sepal width, petal length, and petal width Fisher, 1936. The Iris Setosa class is linearly separable from the other classes, but Iris Virginica and Iris Versicolor have overlapping data points and thus are not linearly separable. It has been shown, for example in (Valko, Marques, and Castellani, 2005), that petal length and width features are sufficient for correct classification. Here we use the data described by these two features, which enables visualization of clustering performance. A matrix representing the complete graph's edge weights is computed by calculating the Euclidean distances from one observation to all others. This graph becomes the input and is translated into an SNN. The clustering result is shown in Figure 14(a). To provide a better sense of the clustering performance, the data points are assigned three shapes according to their class labels shown in Figure 14(b). Since Iris Virginica and Iris Versicolor classes overlap there are a few data points which are clustered together. Biologists have confirmed that the species are co-mingled, so this clustering result was expected.



(a)



(b)

Fig. 14. (a). Clustering solution using either STDP or SAPR learning rules for the Iris data set reduced to two dimensions (b). The output solution where the data points are given different shapes based on class labels. The edges which do not participate in cluster formation are removed.

CHAPTER 7

DISCUSSION

The choice of neuron model, learning rule, neural encoding scheme, and topology must all be deliberately made to address the specific nature of the problem to be solved with SNN. Studies have shown that use of all of the spiking neuron models presented in this work are viable different computing tasks. Many studies in the literature take advantage of the computationally simpler integrate-and-fire type of neuron models. Such neurons are a good choice when the acceptable level of biological plausibility does not necessitate the use of a complex model such as the Hodgkin-Huxley model.

The selection of a neural coding scheme significantly affects the operation of the network. The minutiae of the problem must be taken into consideration. It is currently believed that a rate coding or frequency based coding scheme is unlikely to contain the data required for generalization. Though the coding techniques mentioned above have been used successfully, others yet exist. It is possible to combine some of these techniques to address the particular needs of a problem. Although it presents an increase in pre-processing, Gaussian receptive fields have been used to distribute the input values throughout a given range (Meftah, Lezoray, and Chaturvedi, 2012).

Konorski/Hebbian learning rules exploit the temporal difference among spikes, making it necessary to encode information in such a way that any underlying pattern within the data is discernible in time. Pure temporal encoding would be a good encoding scheme to take advantage of this characteristic. Using a synaptic propagation delay is also an effective approach for the use of Konorski/Hebbian learning rules as it introduces temporal characteristics to the network.

There are extensive quantities of model parameters to consider when developing an SNN model. Consideration must be given to the amplitude of a generated spike; this value determines how many pre-synaptic spikes must be generated to elevate the post-synaptic potential to a threshold value. The acceptable range of synaptic weights must be determined, since Hebbian-style rules will result in a bi-modal distribution of weights near the minimum and maximum. Synaptic weights are often initialized randomly, but consideration should be given to the range of these initial weights as well. Topological attributes are also important, the number of layers, number of neurons, and probability of connections among them all affect the operation of the network. Experiments in neuroscience have found a biologically plausible ratio of inhibitory to excitatory connections to be about 20%. Which synaptic weights receive updates during learning can vary, for example, most studies reduce the learning ability of inhibitory synapses. Selecting these parameters depends upon the problem being solved, but can pose a considerable challenge to creating a well-tuned network.

Asynchronous message passing systems which do not rely on a global time-step to keep the network synchronized have interesting properties. Computation and communication time for neuron membrane potential updates should remain small. Careful attention to the time required for network maintenance operations could permit the use of artificial delays to maintain consistent spike fire timings. Such a parallel implementation is closer to biological observations. The improvements gained by such an implementation are increased scalability and higher computational performance.

CHAPTER 8

CONCLUSIONS

Applications in this work have shown how a computationally efficient neuron model can be used as the basis of an SNN capable of solving common graph problems. Combining a temporal encoding in the form of synaptic delays with the addition of synaptic plasticity rules like STDP and SAPR, the network exhibited self-organizing properties. Although SAPR was shown to improve performance over STDP when used for face image recognition (Shin et al., 2010) there was no significant difference between using STDP and SAPR for the graph problems presented here. The Izhikevich neuron model and the STDP and SAPR synaptic plasticity rules are biologically realistic, giving an interesting perspective on the applicability of biologically inspired artificial neural networks to graph solutions.

CHAPTER 9

FUTURE WORK

There are many opportunities for further investigations into the use of SNN for problem solving. Some optimizations could be investigated to improve the speed and scalability of SNN-based approaches for solving graph problems. It may be possible to parallelize computations by using a message passing system as mentioned above to achieve improved performance. Significant intricacies arise when spiking neurons interact with each other, such as neuron firing order, and how the synaptic connection strengths should be modified. Since time plays such a crucial part of SNN, it is imperative this be accurate. Simplification or generalization of how to solve this implementation issue could reduce the steepness of the learning curve for working with SNN. This work used the Izhikevich neuron model coupled with two types of Konorski/Hebbian learning. Possible continuations of this work could compare the characteristics of different combinations of neuron models and learning rules with the characteristics of using the Izhikevich neuron model combined with STDP and SAPR.

Appendix A

ABBREVIATIONS

VCU	Virginia Commonwealth University
NN	Neural Network
SNN	Spiking Neural Network
STDP	Spike Timing-Dependent Plasticity
SAPR	Synaptic Activity Plasticity Rule
SRM	Spike Response Model

REFERENCES

Fisher, R.A. (1936). “The use of multiple measurements in taxonomic problems”. In: *Annual Eugenics* 7, pp. 179–188.

McCulloch, W.S. and W. Pitts (1943). “A logical calculus of the ideas immanent in nervous activity”. English. In: *Bulletin of Mathematical Biology* 52.1-2, pp. 99–115. ISSN: 0092-8240. DOI: `10.1007/BF02459570`.

Konorski, J. (1948). *Conditioned Reflexes and Neuron Organization.* Cambridge biological studies. University Press, p. 89.

Hebb, D. (1949). *The Organization of Behavior; a Neuropsychological Theory.* Wiley.

MacGregor, R. (1987). *Neural and brain modeling.* Academic Press.

Anderson, J.A. (1995). *An Introduction to Neural Networks.* A Bradford book. MIT Press. ISBN: 9780262510813.

Fiesler, E. (1996). *Neural Network Topologies.*

Abbott, L.F. (1999). “Lapicque’s introduction of the integrate-and-fire model neuron (1907)”. In: *Brain Research Bulletin* 50, pp. 303–304.

Sala, D.M. and K.J. Cios (1999). In: *IEEE Transactions on Neural Networks* 10.4, pp. 953–957.

Gerstner, W. and W.M. Kistler (2002). *Spiking Neuron Models : Single Neurons, Populations, Plasticity.* Cambridge University Press. ISBN: 9780521813846.

Izhikevich, E.M. (2003). “Simple model of spiking neurons”. In: *Neural Networks, IEEE Transactions on* 14.6, pp. 1569–1572. ISSN: 1045-9227. DOI: `10.1109/TNN.2003.820440`.

— (2004). “Which model to use for cortical spiking neurons?” In: *Neural Networks, IEEE Transactions on* 15.5, pp. 1063–1070.

Legenstein, R., C. Naeger, and W. Maass (2005). “What Can a Neuron Learn with Spike-Timing-Dependent Plasticity?” In: *Neural Computation* 17.11, pp. 2337–2382.

Valko, M., N.C. Marques, and M. Castellani (2005). “Evolutionary feature selection for spiking neural network pattern classifiers”. In: *Artificial intelligence, 2005. epia 2005. portuguese conference on.* IEEE, pp. 181–187.

Kasinski, A. and F. Ponulak (2006). “F.: Comparison of supervised learning methods for spike time coding in spiking neural networks”. In: *International Journal of APplied Mathematics and Computer Science* 16.1, pp. 101–113.

Swiercz, W. et al. (2006). “A New Synaptic Plasticity Rule for Networks of Spiking Neurons”. In: *IEEE Transactions on Neural Networks* 17.1, pp. 94–105.

Martig, A.K. and S.J.Y. Mizumori (2010). “Place Cells”. In: *Encyclopedia of Behavioral Neuroscience.* Ed. by George F. KoobMichel Le MoalRichard F. Thompson. Oxford: Academic Press, pp. 70–78. ISBN: 978-0-08-045396-5. DOI: `http://dx.doi.org/10.1016/B978-0-08-045396-5.00154-8`. URL: `http://www.sciencedirect.com/science/article/pii/B9780080453965001548`.

Shin, J. et al. (2010). “Recognition of partially occluded and rotated images with a network of spiking neurons”. In: *Neural Networks, IEEE Transactions on* 21.11, pp. 1697–1709.

Sjostrom, J. and W. Gerstner (2010). “Spike-timing dependent plasticity”. In: *Scholarpedia* 5.2. revision 142314, p. 1362.

Meftah, B., O. Lezoray, and S. Chaturvedi (2012). “Image Processing with Spiking Neuron Networks”. In: *Artificial Intelligence, Evolutionary Computing and Metaheuristics.* Springer Berlin Heidelberg, pp. 525–544. ISBN: 978-3-642-29693-2. DOI: `10.1007/978-3-642-29694-9_20`.

Paugam-Moisy, H. and S. Bohte (2012). “Computing with spiking neuron networks”. In: *Handbook of natural computing*. Springer, pp. 335–376.

Beyeler, M., N.D. Dutt, and J.L. Krichmar (2013). “Categorization and decision-making in a neurobiologically plausible spiking network using a STDP-like learning rule”. In: *Neural Networks* 48, pp. 109–124. ISSN: 0893-6080. DOI: http://dx.doi.org/10.1016/j.neunet.2013.07.012. URL: http://www.sciencedirect.com/science/article/pii/S0893608013001986.

Strack, B., K. Jacobs, and K.J. Cios (2013). “Biological Restraint on the Izhikevich Neuron Model Essential for Seizure Modeling”. In: *Conf. Proceedings of the* 6^{th} *Int. IEEE EMBS Conference on Neural Engineering*, pp. 395–398.

VITA

Shaun Donachy received his B.S. degree in Computer Science in 2013 from Virginia Commonwealth University. His research interests include data mining and machine learning.