Effects Of Real-time Synaptic Plasticity Using Spiking Neural Network Architecture

Matthew Frazier Nishant Shukla Worthy Martin*

***Abstract*—Artificial Neural Networks is a promising approach to study human brain computation in hopes of achieving similar learning by artificial agents. Recent architecture design of a low-power supercomputer by the University of Manchester, the SpiNNaker, has made it easier to design highly parallel brain-inspired algorithms. We used the SpiNNaker machine to implement a neural network capable of rewiring its connection in real-time while trying to minimize information loss and maximize the decrease in statistical dependence. We believe this is the first use of synaptogenesis on a spiking neural network architecture, laying the framework for future effecient brain-like neural networks. Additionally, we explored scalability issues and unintended pitfalls with this approach.**

***Keywords*—Neural Networks, Synaptogenesis, SpiNNaker**

I. Introduction

ARTIFICIAL neural networks often require vast amounts of computation memory and space [1]. Some of the most effective algorithms such as back propogation do not scale well for a network consisting of a billion or more neurons [Cite]. Furthermore, most neural network models do not fully utilize the machine on which they run. The ones that do fully utilize their machine are typically implemented on Graphical Processing Units (GPUs) which disproportionately drain energy.

Finding a solution to revive neural networks to the same computational caliber as other paradigms such as Support Vector Machines (SVM) would better adhere to biologically based design. Studying biologically inspired structures as opposed to statistically based ones allows us to grasp a better understanding of the human brain. By designing a neural network experiment on a new kind of computer architecture, this paper aims to demonstrate the high performing nature of neural networks.

Recently, deep feedforward neural networks have arisen in popularity [Cite]. Multiple deep-learning approaches have proven to be practical. However, a major issue with deep-learning is the computation time for large networks. Deep-learning does not scale well due to the innate use of back-propagation in the algorithm.

Our approach to neural network learning is through an unsupervised model on a spiking neural network architecture. By utilizing every core of the 72-core energy-efficient embedded system, we are capible of reproducing complex neural network patterns in less than half the energy-cost. The network creates and removes synapses in real-time, so not only are weights learned and adjusted simultaneously, but so are the actual connections between the neurons. We demonstrate that introducing synaptic modification on a neural network leads to a biologically accurate model that outperforms previous previous methods in both performance and energy costs.

II. Background

THE subfield of AI known as neural network computation has recently received a large and growing amount of attention. Put simply; neural networks (NN) are computational models inspired by biological brains that are capable of machine learning. A NN usually consists of interconnected "neurons" (in quotation marks for they are artificially simulated to varying degrees of biological accuracy – but that topic is outside the scope of this paper) which compute from their inputs and produce various outputs, depending on the model.

This paper focuses on types of spiking neural network (SNN) models, in which the neuronal communication (and therefore, the computation of the system as a whole) is achieved via message spikes, or action potentials, from one neuron that is synapsed onto another. In this introduction, we first describe how a SNN is a good computational representation of the brain. We then describe various different spike-dependent spike-time learning methods used on a SNN, followed by an exploration of various types of SNN. We discuss why the neuromimetic hardware we use is a good platform for experimentation with SNN, and then describe our approach to the problem of simulating synaptogenesis on such a device and propose a solution. Our design section explores this solution in detail, and our implementation section describes our results. Furthermore, we discuss future work that remains to be conducted, including interesting implications such as computability beyond Von Neuman architecture and Turing machines.

A. The Brain And Other Spiking Neural Networks

The brain is a spiking neural network. That is to say; biological neurons pass information around the brain via action potentials which travel from a neurons' soma (cell body) down the axon and generate synaptic events which are received by the dendrites of all other connected neurons. Figure 1 diagrams the relevant biological prerequisites.

The process has been simplified greatly as neurobiology is not the focus of this paper, nor is the biological process yet completely understood. What's important are the overarching properties exhibited by brains. Namely; that brains are fast, power-efficient, and capable of obtaining, representing, and integrating complex multi-dimensional information sets into useful knowledge (and from noisy sensory inputs, at that). Moreover, the property of being able to fairly reliably solve highly complex problems is something we want computers to

have. [Include TSP example illustrating that a human could do in a day what a procedural program would take hundreds of thousands of man-years to do?] Progress is however being made towards the goal of making machines more like minds; indeed, the bus in a computers' hardware may operate at tens of MHz, while an axon may carry only tens to hundreds of action potentials per second, about five orders of magnitude slower than a machine (Furber 2007). The power efficiency of the brain is still far ahead of even the best neuromimetic hardware but the gap is closing quickly, considering that biological evolution had several hundred million years of a head start.

One very important property of the brain that has received little attention by Computer Scientists, which is also believed to be quintessential in mankind's ability to adapt to and solve new and ever-more difficult problems, is neural plasticity involving the dynamic creation or removal of synapses. It would be erroneous to say that no degree of plasticity has been implemented in any type of SNN, but to the best of our knowledge there is yet no model which allows networks to add or remove synapses at runtime. Given that we do not yet fully understand the mechanisms governing neural plasticity in our own heads, it suffices to say that there is yet much work to be done in this field of computational neuroscience in which artificial neural networks dynamically alter their own topology.

B. Overview of Spike-time coding and learning methods

Before discussing different types of artificial spiking neural networks, it is necessary to describe various spike-time coding and learning methods that are relevant to understanding how the different types of spiking neural networks function.

Firstly, the distinction between rate- and rank-order coding must be addressed. Rate order coding is, as it sounds, a way to describe neural responses to stimulus in terms of firing rate, paying attention to the timing between spikes of a single input. Rank order, on the other hand, focuses on the relative timing of spikes from all inputs. Rate order has a number of drawbacks, though the primary one is its inability to transmit and process information in a short period of time; with n neurons being able to transmit over the course of 10 ms only $log_2(nC1)$ bits of information [22]. Rank order, under the same constraints, is capable of achieving $log_2(n!)$ bits of information. [15]

As detailed in [22], rate order can be considered an $analog-to-frequency$ converter, while rank order would be an $analog-to-delay$ converter, as the time an integrate-and-fire neuron takes to reach threshold is dependent upon input strength. During an initial learning phase, in which a new output neuron i is created for each N-dimensional training input pattern, connection weights $w_j, i(j = 1, 2, ..., N)$ are calculated based on the rank order learning rule:

$$w_j, i(t) = \alpha \cdot mod^{order(j,i)} \tag{1}$$

where w_j, i is the connection weight between post-synaptic neuron i and pre-synaptic neuron j, α is a learning parameter, mod is a modulation factor (determining the importance of the order of the first spikes), with $order(j, i)$ representing the order of spikes arriving at neuron i from all synapses to the neuron i (has value 0 for the first spike to neuron i, increases incrementally for each subsequent spike from other pre-synaptic neurons). [15] Some models using rank order coding will also merge output neurons with similar weight vectors based on the Euclidean distance between them, thus reducing the number of redundant neurons that must be simulated. In short, rank order (RO) coding is assumed to be a better basis for model construction as it allows networks to process more information in a shorter amount of time with fewer redundant neurons. One of the only downsides to rank order over rate order is an increased overhead.

Moving on to learning methods, spike time dependent plasticity (STDP) implements plasticity through the use of long-term potentiation (LTP) and depression (LTD). In other words, the connection weight between two neurons increases if the pre-synaptic neuron spikes before the post-synaptic neuron, and the weight decreases in the reverse case. Spike driven synaptic plasticity (SDSP) is a semi-supervised variant of STDP in which a threshold $Vmth$ is given to the post-synaptic neurons' membrane potential. If the membrane potential is above $Vmth$ when an input spike arrives, potentiation occurs. Otherwise, the synapse experiences depression. These two cases are typically either shortly before or shortly after a post-synaptic spike is emitted, respectively.

STDP is the most commonly discussed paradigm for learning in neural networks. Its benefits include its unsupervised nature, and, significantly, the ability for the LTP and LTD weight modifications to be calculated by the post-synaptic neuron upon receipt of a spike. This allows the updates to be performed at a single point in the simulation: at the spike arrival event, which can further increase the efficiency of a simulation. [4], [5]

We will later describe a new learning method referred to as synaptogenetic spike time dependent plasticity (SSTDP), utilizing STDP and governed by biologically inspired formulas taken from (Levy), and influenced by Information Theory from Shannon.

C. Types of Spiking Neural Networks

(a) **A spiking neural network (SNN)** is the most basic form, in which synaptic weights are static and neurons communicate via message spikes. While they have some drawbacks, an SNN is still a fairly powerful biologically-inspired model.

(b) **An evolving spiking neural network (eSNN)** evolves its functionality and structure based on incoming information using the RO rule, but only during a learning phase. Once a neurons' weight has been set, however, this model does not allow for any further tuning of synaptic weights to reflect on other incoming spikes at the same synapse. That is to say; these synapses can capture some long-term memory during the learning phase, but have little ability to capture any short term memory, which is detrimental to the computational capabilities of the network as a whole. [15]

(c) **A dynamically evolving spiking neural network (deSNN)** represents an improvement over eSNN in that it implements not only RO learning during an initial

learning phase, but also a dynamic synaptic plasticity mechanism so that it will continue to learn and improve performance during a recall phase; the deSNN proposed in [15] uses the SDSP mechanism. This fixes the primary shortcoming of the eSNN model; allowing for short term memory to be stored easily, and to be potentiated as long as it remains useful [15].

D. SpiNNaker

The Spiking Neural Network Architecture (SpiNNaker) board is a neuromimetic hardware system designed by the University of Manchester with the eventual intent of simulating the human brain. Each SpiNNaker processing node is a multi-core system-on-chip (SoC) with 1GB SDRAM, containing 18 ARM968 processor cores, connected in a toroidal triangular mesh with passed messages being managed by an on-chip router for an asynchronous packet-switched network-on-chip (NoC). [18] It is the brainchild of Steve Furber, and has a number of properties that make it useful for neural computation:

Fast
: The SpiNNaker system has a bisection bandwidth of over 5 billion packets/s [7]. It is, by design, meant to model spiking neural networks in real time. However, it can be said to outperform biology in terms of raw speed of neural activity, and must be slowed down to run real-time simulations [7], [19].

Power efficient
: A simulation run on the SpiNNaker system has been documented taking 100 nJ per neuron per millisecond and 43 nJ per postsynaptic potential, a smaller power consumption than any other recorded digital simulation [20].

Massively parallelized
: The brain is a massively parallel system populated with many low-performance asynchronous components known as neurons [8]. It follows that a digital system meant to effectively and efficiently emulate neural activity would also be massively parallelized, though each ARM core is substantially more complex than a single neuron, and can efficiently model many neurons in real time [7]. Each NoC is extended seamlessly to surrounding chips, creating a power-efficient system-wide interprocessor communication network. [18]

Globally asynchronous, locally synchronous (GALS)
: In a GALS system, individual processing cores have their own clock signals, potentially with different frequencies, and requiring no interprocessor phase-alignment [18]. Each SpiNNaker chip is its own GALS system, with an independently-clocked router managing intercore and interchip communications, leading to the SpiNNaker system as a whole also being a GALS system [8]. Major benefits of a GALS approach include that it eliminates any top-level system constraints (i.e. it is decentralized), and any timing closure issues [18]. It also offers increased flexibility regarding process variability [8].

Fault Tolerant
: In any large-scale systems fault tolerance is a major concern, and is increases in importance with the number of fallible subsystems. Fault tolerance was paid a great deal of attention by the SpiNNaker development team, and listing all the built-in mechanisms at various levels of abstraction is outside the scope of this paper. To summarize, fault tolerance mechanisms exist at nearly every level of the system, including those related to processors, the interrupt controller, timers, and packet communications, allowing subsystems to generally fail gracefully without greatly affecting the performance of the system as a whole [7].

Scalable
: Due to the nature inherent to its GALS design, it is arbitrarily scalable. The only cost associated with scaling up the system is power consumption. The full million-core machine has an expected power budget of around 90 kW [7].

To summarize, the SpiNNaker specifically disregards three significant axioms of conventional supercomputing (memory coherence, synchronization, determinism) in a way that makes it well-suited for a wide range of biological and non-biological applications [7].

E. Synaptic Plasticity

A difficult problem in designing neural networks is configuring the initial parameters to optimize the network. For example, defining how the neurons are connected together greatly affects the efficiency and performance in learning. [3] Perceptrons are often designed as bipartite graphs, where every neuron in the pre-synaptic layer is connected to every neuron in the post-synaptic layer. This brute-force approach to constructing a network topology may achieve satisfactory results if the weights converge to the optimal values within an acceptable time. However, a bipartite graph scales exponentially to the number of nodes in each layer. In our solution, we let the neural network modify itself, providing a "hands-off" approach to designing a network.

Other than updating synaptic weights to achieve learning, we introduce two additional forms of synaptic plasticity. In our study, we allow new synapses to be formed (synaptogenesis), or existing synapses to be entirely disconnected. The three types of plasticity undergo the following principles (Levy 2004):

(a) **Synaptogenesis** - Unlike most networks, we allow ours to form new connections between neurons while learning occurs. The creation of a new synapse depends on whether the post-synaptic neuron is at an optimal activation. If not, the network considers forming a new synapse. Additionally, synaptogenesis only occurs between neighboring neurons, ensuring a constraint on locality. Synapses are formed by taking into account both receptivity and avidity (Levy 2004).

Avidity is the measure of ability for each neuron to participate in a new synaptic connection, defined as

$$A_i(t) = \frac{a}{(a + \sum_j \sum_k W_{i_k j}(t))}$$

$$a = \begin{cases} 1.0 * 10^{33} & \text{for unlimited avidity} \\ 1.0 & \text{for moderate avidity} \\ 1.0 * 10^{-3} & \text{for limiting avidity} \end{cases}$$

The receptivity of new synaptic connections is inversely proportional to the running average of the neuron's activation. We use the following equation for receptivity,

$$R_j(t) = \frac{r_1}{r_1 + \bar{y}_j(t)^{r_2}}$$

$$\bar{y}(t) = 0.99\bar{y}_j(t-1) + 0.01Y_j(t)$$

(r_1 and r_2 are experimental constants)

Finally, the probability of forming a new synapse between two neurons i and j depends on both avidity and receptivity.

$$\text{Prob(of new synapse } ij) \propto A_i(t) * R_j(t)$$

(b) **Weight Modification** - Similar to most unsupervised weight modification rules, we use a deterministic process depending on the pre- and post-synaptic activities. This type of weight adjustment is based off the Hebbian rule, where links between nodes that fire together strengthens. Specifically, weights are adjusted using the following formula,

$$\triangle w(t+1) \propto f(\text{post}_j(t)) * g(\text{pre}_j(t), w_{ij}(t))$$

(c) **Synapse Removal** - Lastly, we allow our network to undergo synaptic removal, a stochastic process where the probability of removal becomes non-zero if excitatory synapses fall bellow a specified threshold. Given some constant $\delta > 0$, probability of synaptic removal is defined as

$$\text{Prob(removal } ij) = \begin{cases} 0 & \text{if } w_{ij}(t) + \triangle w_{ij}(t+1) > \delta \\ > 0 & \text{otherwise} \end{cases}$$

Synaptic connections are slowly modified through additions or removals until optimal output firing levels are obtained.

Levy asserts that "information theory has gained popularity in recent years as a tool for understanding brain recordings."

III. Design

NEURAL networks consist of biologically inspired relationships between individual nodes (neurons). Learning in such a network occurs by intelligently adjusting weights on the links between the nodes. We define the relationship between nodes into the following three categories:

(a) **Static** - where the network topology is fixed, and can only change by manually adjusting the relationship between nodes. Most neural networks fall into this category because regardless of the number of nodes n, only one fixed relationship forms between them:

$$\text{NumberOfTopologies}(n) = 1$$

(b) **Semi-Dynamic** - in which the network is not static, and there is some freedom for nodes to rewire with other nodes in real time. The complexity grows exponentially. Each node has non-zero probability to be rewired with a constant c number of other nodes. Given n such nodes, the number of possible topologies can be up to

$$\text{NumberOfTopologies}(n) = c^n$$

(c) **Dynamic** - where each node has non-zero probability to be wired with any other neuron in the entire network. Given n nodes, the number of possible topologies becomes

$$\text{NumberOfTopologies}(n) = n^{n-1}$$

A biological neural network such as the human brain does not follow the static network topology. Connections between neurons in the brain are regularly formed and removed over time [2], [3]. Moreover, such a neural network is not fully dynamic either, since locality is a physical constraint. For example, a neuron on the far end of the left hemisphere of a brain might never directly connect with a neuron on the right hemisphere.

We designed a programming framework on SpiNNaker's interface to enable a semi-dynamic network topology. Each neuron has the flexibility to rewire with none, all, or some of the fixed number of other neurons, following our synaptic plasticity rules. To test performance, the network analyzed the MNIST database of handwritten digits. We compared information loss (see appendix A) and statistical dependency (see appendix B) from a network following our synaptic plasticity model, to that without synaptogensis and synapse removal.

In order to implement a spiking neural network, we choose an existing neural model. There exists multiple neural models for implementing a spiking neural network. Some of the most popular models are briefly covered below.

(a) **Integrate-and-fire (IF)** is one such model which increments the voltage while a current is present, and finally resets the voltage once a threshold V_{th} is reached. Every voltage reset produces a spike for that neuron. While computationaly effecient, the IF model does not exhibit properties of the cortical spiking neurons.

(b) **Leaky integrate-and-fire (LIF)** models are a biologically revised version of the previous model. This model

allows for "forgetting" isolated input currents. However, it also is insufficient for modeling cortical spiking neurons.

(c) **Izhikevich** model is an adaptation of the Integrate-and-fire model that uses a quadratic non-linearity when adjusting the voltage. This model is known for its applicability to large-scale simulations of cortical neural networks [CITE].

We use the Izhikevich model since it our best candidate for a biologically inspired neuron that is simple enough to implement on a new computer architecture. By running an ensemble of neurons on the SpiNNaker, we wish to see both a more energy and time effecient approach for emulating neural networks. In direct comparison to Izhikevich's paper on polychronization, we present a spiking neural network on the SpiNNaker that can also exhibit reproducible time-locked but not synchronous firing patterns. This time-lock pattern is referred to as polychronization [Cite]. These polygroups are possible due to introducing delays in the network. The SpiNNaker machine produces natural delays as packets are sent to and from chips. We take advantages of the delays to discover these polygroups.

IV. Implementation

BY taking advantage of the multiple independent cores on the SpiNNaker, we were able to emulate efficient real-time synaptogenesis. The neural network consisted of 39 excitatory input neurons, 9 inhibitory neurons, and 16 output neurons.

A. Input Spikes

The SpiNNaker neural network was given a set of letters as input. Each letter was represented by a 48-dimensional binary vector of $1s$ or $0s$. Intuitively, a 1 represents a black pixel, and a 0 represents a white pixel in a visual 8 x 6 image representation of the letter.

Each coordinate of the 48-dimensional input spikes its cooresponding input neuron. The letters are randomly generated by a Python script which sends a spike to the SpiNNaker machine one by one.

B. Floating Point Arithmetic

The SpiNNaker machine does not support floating point arithmetic. However, the Izhikevich neural model adjusts voltage variables based off the following equations.

$\triangle v = 0.04v^2 + 5v + 140 - u + I$

$\triangle u = a(bv - u)$

In order to alleviate the rounding errors from only using integer arithmic on the SpiNNaker machine, we scale the equations up to remove all floating points values. Then we use integer division in the final step to obtain the actual result. The rounding errors of implementing the Izhikevich model are minute as demonstrated in the following diagrams.

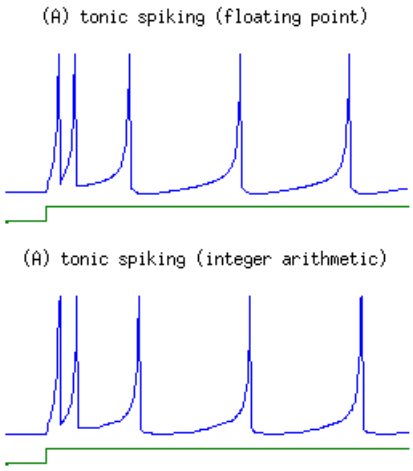

Fig. 1. Tonic spiking of a neuron.

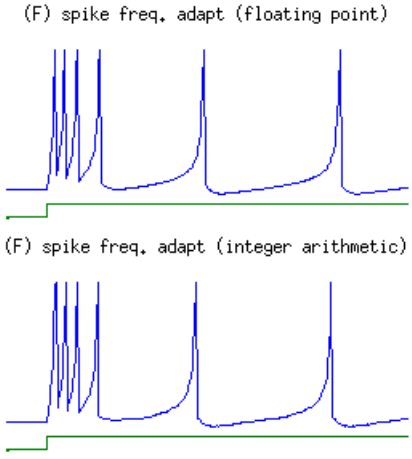

Fig. 2. Spike frequency adaptation of a neuron.

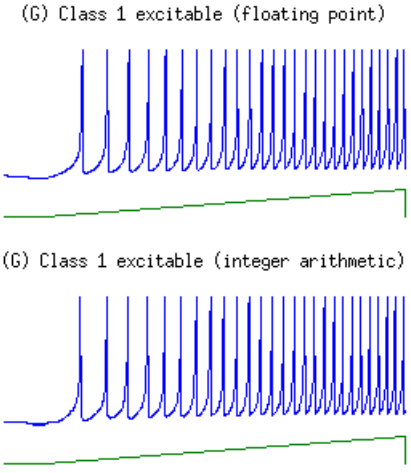

Fig. 3. Class 1 excitable neuron.

C. Parallel Design

The algorithm written on the multi-core parallel SpiNNaker system is a translation of sequential MATLAB code. Instead of dealing with lists of neurons, we were able to simple use a variable per each neuron core. Likewise, instead of using a two-dimensional array, where one dimension was the index of a neuron, we were able to simply use a one-dimensional array per each neuron core. Arguably, the parallel implementation on the SpiNNaker was easier to design due to the code only ever acting locally on one core.

We used the weight modification rule proposed by Levy to categorize an input set of letters.

[Delta W rule]

We implemented a single-layer perception as a proof of concept to demonstrate synaptogenesis on the SpiNNaker board. In our network, every input node broadcasts a data packet to six of the output nodes. To simulate the creation and rewiring of synapses, not all messages are registered by the output neurons.

Each packet received by an output node is looked up in the output node's local list of incoming connections. If the address from the packet is not listed in the incoming connections list, it will be ignored, and no further computation will be done.

Synaptic connectivity is broken when weights fall below some negative threshold. More specifically, when an average running weight dropped below $-\delta$ for some $\delta > 0$, the link is considered disconnected.

New connections are established between nodes when an average rate of activity drops below 25%. In this case, the next data packet received from a muted neuron becomes unmuted, in hopes to raise the average rate of activity.

V. Conclusion

THrough emperical evidence, we have shown that by taking advantage of a spiking neural network architecture such as the SpiNNaker, we can implement a high performance energy effecient neural network system. Without the computation drawbacks of back-propagation, or the growing space complexity of global variables, we are able to train a network on a machine that uses less than 5 Watts of energy [Cite]. Moreover, we have futher demonstrated that a neural network can produce outstanding results by having its input be fed in series as opposed to in parallel.

VI. Pitfalls

SOME disadvantages are present when considering our approach for synaptogenesis on the SpiNNaker. Programming parallel applications is a challenge in its own right, but a lack of floating point support on the SpiNNaker system [7] compounds the difficulty, especially when the neural dynamics parameters and coefficients in the differential equations of Izhikevich's simple neural model are floating point numbers [13]. To get around this, we rationalized all coefficients in the equations and multiplied the entire set of equations by the lowest common multiple of the denominators of the rationalized coefficients. This incurred a slight loss of precision, but not so much that the models we used exhibited different neural activity behavior than their original counterparts with the same input. Source code illustrating this is available upon request.

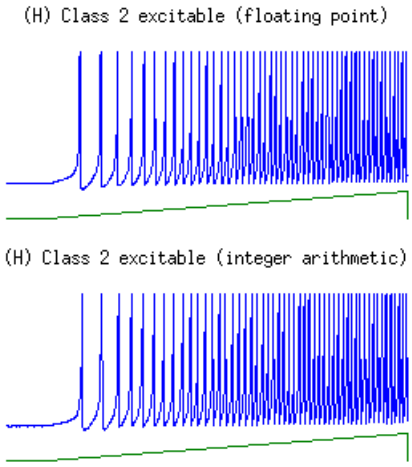

Fig. 4. Class 2 excitable neuron.

VII. Further Study

THIS paper reveals multiple questions that still need to be examined further. A key feature of the SpiNNaker machine is its ability to connect to other SpiNNaker chips for scalable performance. Future papers can examine the performance of a network of SpiNNaker chips together. It's been shown that with "quarter of a million neurons, tens of millions of synapses and dynamic activity of over a billion synaptic events per second can be delivered within a 30 W power envelope."

Appendix A
Shannon's Entropy and Mutual Information

Appendix A text goes here.

Appendix B
Calculation of Statistical Dependence

Appendix B text goes here.

Appendix C
Floating point vs. Integer arithmetic

Appendix C.

Acknowledgment

The authors would like to thank Professor Worthy Martin, Associate Professor of Computer Science at the University of Virginia.

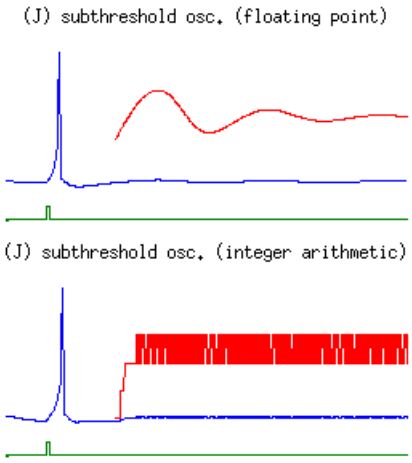

Fig. 5. Subthreshold oscillatory neuron.

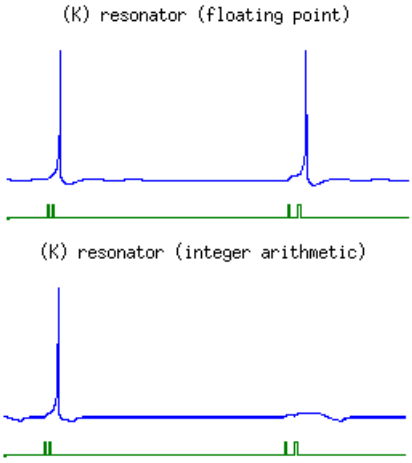

Fig. 6. Spiking resonator neuron.

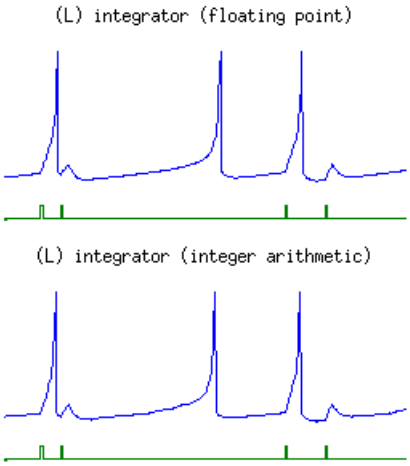

Fig. 7. Integrator neuron.

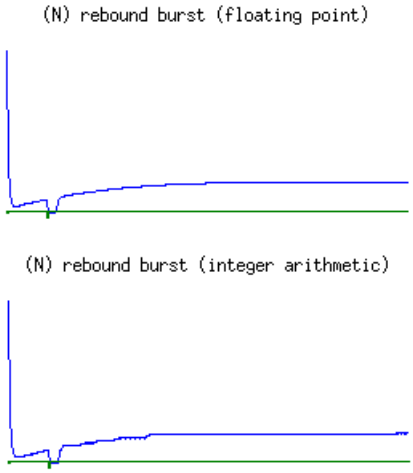

Fig. 8. Rebound burse neuron.

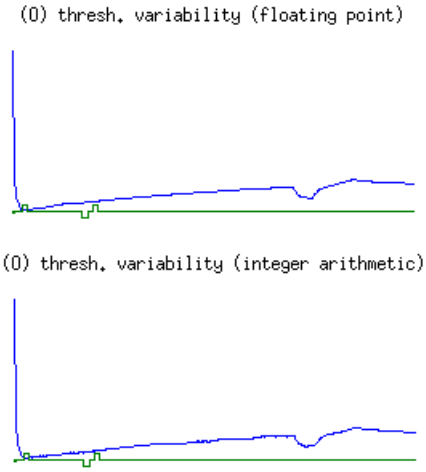

Fig. 9. Threshold variability neuron.

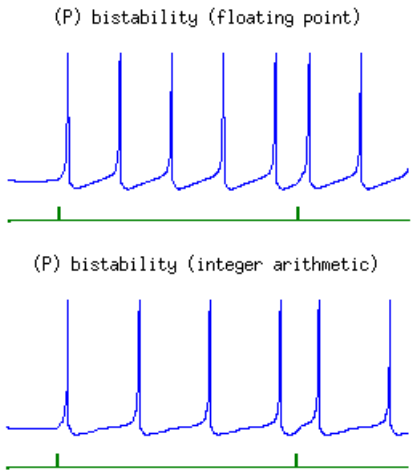

Fig. 10. Bistability neuron.

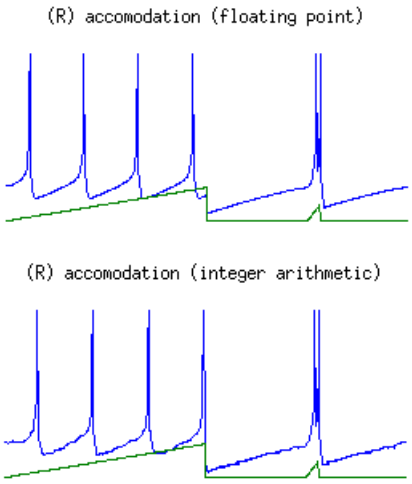

Fig. 11. Accomodation neuron.

REFERENCES

[1] D. Adelsberger-Mangan and W. Levy in *International Joint Conference on Neural Networks 4*, 1992

[2] D. Adelsberger-Mangan and W. Levy in *Biological Cybernetics 67*, 1992, pp. 469-477

[3] D. Adelsberger-Mangan and W. Levy in *Biological Cybernetics 70*, 1993, pp. 81-87

[4] S. Davies et al. in *Proceedings of International Joint Conference on Neural Networks*, San Jose, CA, 2011

[5] S. Davies et al. in *Neural Networks 32* 2012, pp. 314

[6] S. Davies et al. in *Proceedings of the 8th ACM International Conference on Computing Frontiers*, New York, NY, 2011, Article No. 15

[7] S. Furber et al. in *IEEE Transactions on Computers, Vol. 62, No. 12*, 2013, pp. 2454-2467

[8] S. Furber and A. Brown in *Ninth International Conference on Application of Concurrency to System Design*, 2009, pp. 3-12

[9] S. Furber and S. Temple in *Journal of the Royal Society Interface 4*, 2007, pp. 193-206

[10] F. Galluppi et al. in *Proceedings of the 9th conference on Computing Frontiers*, New York, NY, 2012, pp. 183-192

[11] F. Galluppi and S. Furber in *Internation Joint Conference on Neural Networks*, San Jose, CA, 2011, pp. 943-950

[12] J. Humble et al. in *From Brains to Systems*, Springer New York, 2011, pp. 19-31

[13] E. Izhikevich in *IEEE Transactions on Neural Networks 14*, 2003, pp. 1569-1572

[14] E. Izhikevich in *Neural Computation 18*, 2006, pp. 245-282

[15] N. Kasabov et al. in *Neural Networks 41*, 2013, pp. 188-201

[16] J. Navaridas et al. in *Proceedings of the 23rd international conference on Supercomputing*, 2009, pp. 286-295

[17] L. Plana et al. in *IEEE Design and Test archive, Vol. 24 No. 5*, 2007, pp. 454-463

[18] L. Plana et al. in *ACM Journal on Emerging Technologies in Computing Systems, Vol. 7, No. 4*, 2011

[19] A. Rast et al. in *Neural Networks 24*, 2011, pp. 961-978

[20] T. Sharp et al. in *Journal of Neuroscience Methods 210*, 2012, pp. 110-118

[21] K. Stanley and R. Miikkulainen in *Evolutionary Computation, Vol. 10 No. 2*, 2002, pp. 99-127

[22] S. Thorpe and J. Gautrais in *Computational Neuroscience; trends in research*, 1998, pp. 113-118