Analysis of a simple object oriented simulation of STDP in memristor synapse arrays for potential use in event-driven contrastive divergence

Vladimir Jovanovic
Dept. of Neurosciences
U.C. San Diego
La Jolla, CA 92093
vjovanovic@ucsd.edu

Vy A. Vo
Dept. of Neurosciences
U.C. San Diego
La Jolla, CA 92093
vav001@ucsd.edu

Abstract

Memristors are electrical devices whose conductance can be modulated by the charge and voltage flux through the two elements. Previous work has shown that memristors are good models of synapses, even reproducing learning behavior such as spike-time dependent plasticity (STDP). As such, there is widespread interest in understanding how large networks of memristor synapses might be trained to perform tasks such as visual perception or categorization (for review, see [1]). Our project aim is to simulate a small network of memristor synapses using an object-oriented programming language (Java) to create each element of the network, following previously published models. The main aim will be to demonstrate and characterize STDP behavior in our modeled cells. We expect that the results of this simulation will be similar to those reported by other groups [1]. The secondary aim will be to use this form of plasticity to train weights in a deep learning network. This aim will be mainly exploratory.

1 Memristor networks and spike-time dependent plasticity

1.1 Memristors and neuromorphic systems

Large-scale hardware simulations of neural systems may provide significant insight into emergent properties that arise from fairly simple biological principles. One promising neuromorphic system depends on an electrical device called a memristor. The conductance of a memristor can be modulated by the charge and voltage flux through the two elements of the device, which makes it an ideal candidate for a hardware implementation of an electrical synapse. Moreover, these devices demonstrate complex learning behavior similar to a characteristic property of real neural systems: spike-time dependent plasticity (STDP).

STDP follows the basic principle of Hebbian learning, but further postulates that a pre-synaptic spike which precedes a post-synaptic potential increases the strength of that synapse, whereas a post-synaptic potential which precedes a pre-synaptic spike decreases the strength of that synapse [2]-[3]. These recent demonstrations using small arrays of memristors indicate that networks of memristors may have similar properties to networks of cells.

1.2 Event-based simulations of spike-time dependent plasticity

Arrays of memristors with STDP behavior have successfully learned fairly complex stimulus properties. For example, Linares-Barranco and colleagues have shown that a simple array with a layer of excitatory synapses and a layer of inhibitory synapses connected to an output layer can inherit receptive field properties similar to those in the early visual system [4]. However, building such devices requires both resources and technical skill, and exploring the full extent of memristor array capabilities requires some degree of software simulation. Most simulations are currently dependent on ordinary differential equation (ODE) solvers which describe the behavior of each memristor at every time step of the simulation. Such ODE solvers are effective and fairly accurate, but generally limit the scale of the simulation, as these calculations are computationally expensive.

Instead, event-based simulations of neural spiking only simulate relevant events, such as spikes and synaptic weight updating. Furthermore, since each cell or synapse is represented as an independent object, the state of the entire system does not need to be updated at every time step, further reducing the computational expense of the simulation.

Our goal was to build an event-driven simulation of spiking cells in arrays of neurons. Instead of modeling individual electrical properties of the memristors, we directly implemented STDP in all synapse objects. Our final exploratory aim was to investigate how our simulation performed on basic learning tasks, such as categorization.

2 Java simulation with a graphical interface

2.1 Basic structure

We used the object-oriented language Java to build our simulation. Each neuron was represented as an individual object with properties such as voltage, threshold, and refractory period. A synapse class maintained information about which neurons were connected, whether the connection was excitatory or inhibitory, and how the weight of the synapse would be updated as its pre- or post-synaptic neuron fired. For simplicity, all spiking events were modeled as delta functions, and voltage updates were discrete step functions. Event-timing was maintained by a priority queue.

2.2 Program Flow

Three classes interacted with each other to create the graphical user interface (GUI). First, NetworkUI created the graphical components and maintained the interaction between the user and the program by processing button clicks. It also created the neural network that would be displayed to the user. The actual network was stored in a separate class named NeuronSystem. This class created the Neuron objects and connected them with Synapse objects as needed by NetworkUI. It maintained the list of neurons and how to interact with them. Finally, NetworkGraphDisplay was used to paint the network as needed. To cut processing time, only key aspects of the screen would be repainted. Whenever a neuron changed color due to an event, only the relevant portion of the screen that includes the changed neuron would be repainted. (Figure 1).

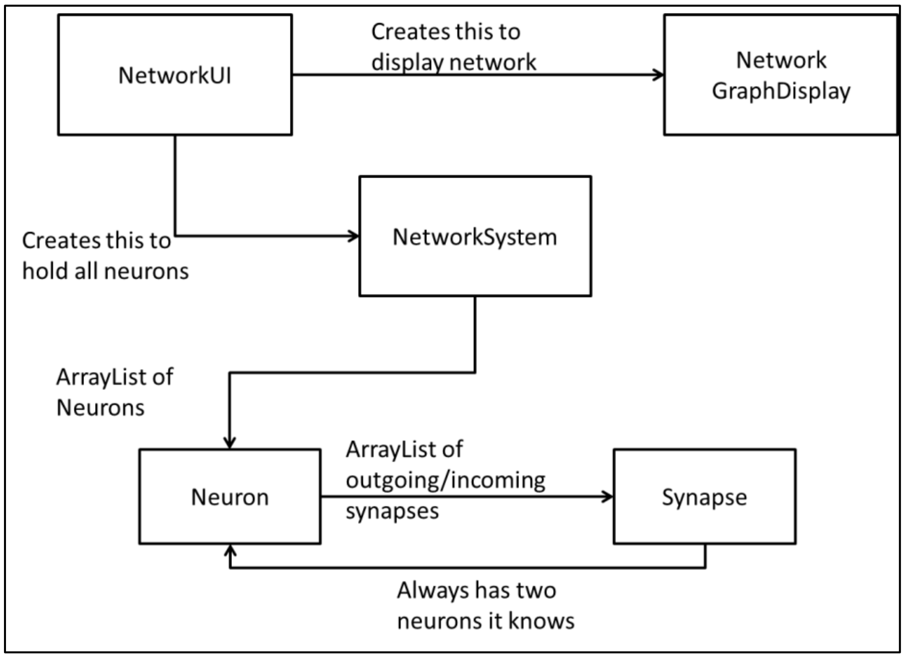
NetworkUI
Creates this to display network
Network GraphDisplay
Creates this to hold all neurons
NetworkSystem
ArrayList of Neurons
Neuron
ArrayList of outgoing/incoming synapses
Synapse
Always has two neurons it knows

Figure 1. General outline of Java program for event-driven STDP. Each arrow indicates that the pointer contains an object of the pointed class (e.g. NetworkUI has an instance of NetworkSystem). Not shown: NEventQ that demonstrates how the event queue behaves.

2.3 Neural Event Queue

The neural event queue refers to the priority queue that maintained a list of all the interactions between neurons. A spike event and refractory-end event were added to the priority queue each time a neuron fired. These events would be separated in time according to the refractory period, and the priority queue added them according to their time stamp. This ensured that all events that were processed happened sequentially.

2.4 Spike Behavior

A neuron had three distinct events to deal with: the spike event when its voltage surpassed the threshold, a receiving event to update its internal state when a presynaptic neuron fired, and a refractory period event. The spike event prompts an update in weights to all synapses going in to the neuron and out of the neuron. It also forces the update of neurons connected to the outgoing synapses of the spiking neuron. The neuron that is updated has to take the weighted voltage input (given the delta function of 0 or 1, it was simply the synaptic weight) and add it to its current voltage. If the voltage surpassed the threshold, then a new spike event is added to the event queue along with the refractory event. During the refractory period, the neuron would not update its voltage. The refractory-end event would then reset the current voltage of the neuron and allow it to accept new input.

2.5 GUI Walkthrough

The user has to first press a button to create the network, and then press Run Network. The run network command causes a series of spike events to be added to the event queue that correspond to the network. An example of a randomly generated network is shown in Figure 2. These neurons are set to fire spontaneously until the user chooses to end the program. A second example network is shown in Figure 4A, where an input causes specific spiking behavior in the top layer of the network. In this version, the user can also run a simple learning rule to force the synaptic weights to increase or decrease depending on whether the network has correctly classified the input (for more detail, see section 3.2).

The user can also choose to print a log of spike times and synapse changes. However, the user does not have the ability to directly interact with the neural network.

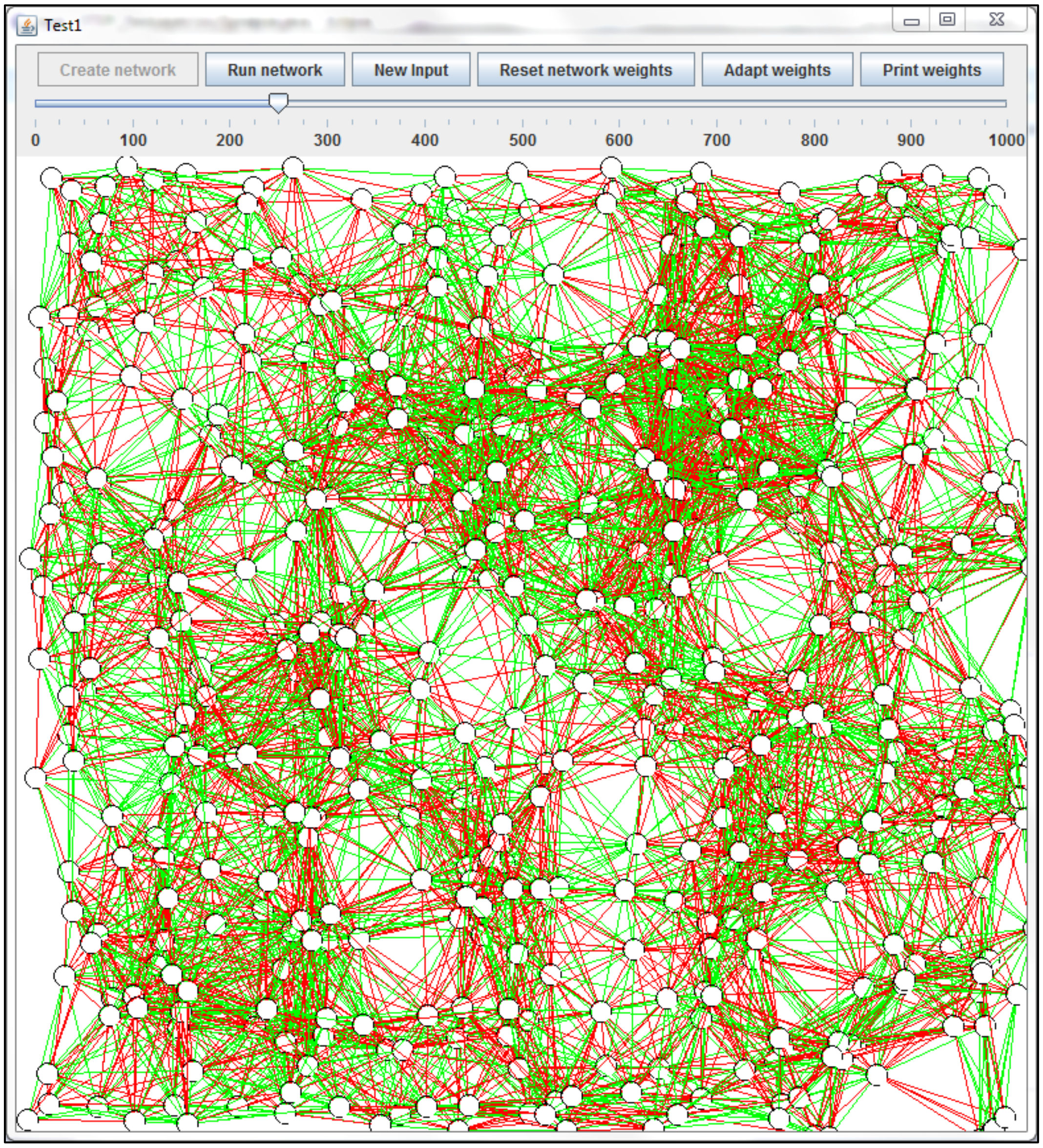

Figure 2. GUI demonstration of the Java program. A randomly generated graph is output to the screen once the user presses “Create network” button. The user can change the speed of event drawing by moving the slider appropriately left or right.

3 Spike-time-dependent plasticity and learning behavior

As mentioned in 2.4, each spike event triggers an adaptation of the synaptic weights. All synapses keep an updated time stamp of the most recent pre- or post-synaptic spikes. The weight adjustment is then calculated according to the time difference ($\Delta t = pre - post$).

3.1 STDP behavior in the simulation

The program used an STDP function derived by Song *et al.* [5]. The equation differentiated between times where the time difference was negative or positive. Accordingly, negative time difference, that is, pre-synaptic neurons firing before post-synaptic neurons led to an increase in synaptic weight. Positive time difference led to a decrease in synaptic weight. The equation is reproduced here along with the appropriate parameters used in our demonstration (Equation 1, Table 1).

Equation 1 was used for excitatory connections, where F was the percent change in the synaptic weight. For inhibitory synapses, the same equation was used except with a -1 multiplication attached to both pieces of the equation (i.e., ‘anti-STDP’).

Equation 1

$$F(\Delta t) = \begin{cases} A_{+} \exp\left({}^{\Delta t}/_{\tau_{+}}\right), & \Delta t < 0 \\ -A_{-} \exp\left({}^{-\Delta t}/_{\tau_{-}}\right), & \Delta t \geq 0 \end{cases}$$

Table 1: Parameter Values

A_{+}	0.005
A_{-}	0.00525
τ_{+}	20
τ_{-}	20

3.1.1 Reconstruction of STDP function

By creating a randomly connected neural network (Figure 2), and allowing spontaneous firing (spike events that randomly occurred at prescribed intervals), we were able to log all changes to the synaptic weight and the associated difference in spike times. This allowed us to show that the implementation of the Equation 1 in the weight update function of the Synapse class was working (Figure 3).

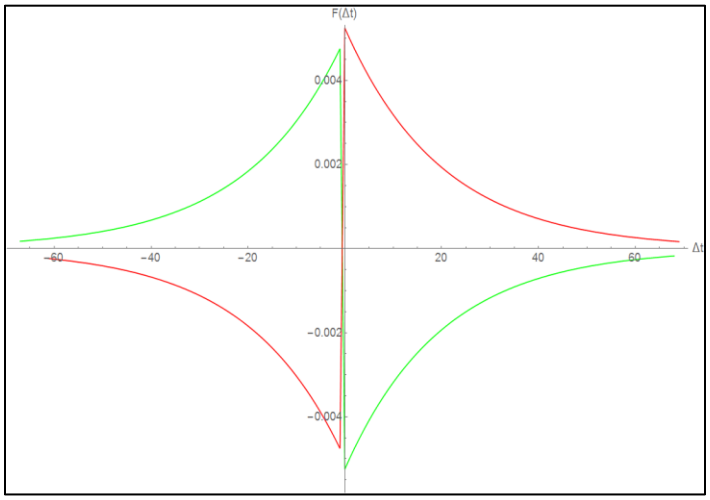

Figure 3. Recreated STDP function from synaptic weight updates of a 500 neuron network. Green line for excitatory synaptic weight updates and red for inhibitory synaptic weight updates. F represents the change in percent of the synaptic weight.

3.2 Network performance on a categorization task

Taking inspiration from a recent memristor simulation [4], we created a simple network which consisted of a layer of excitatory synapses and a layer of inhibitory synapses which both converged on an output layer (Figure 4A).

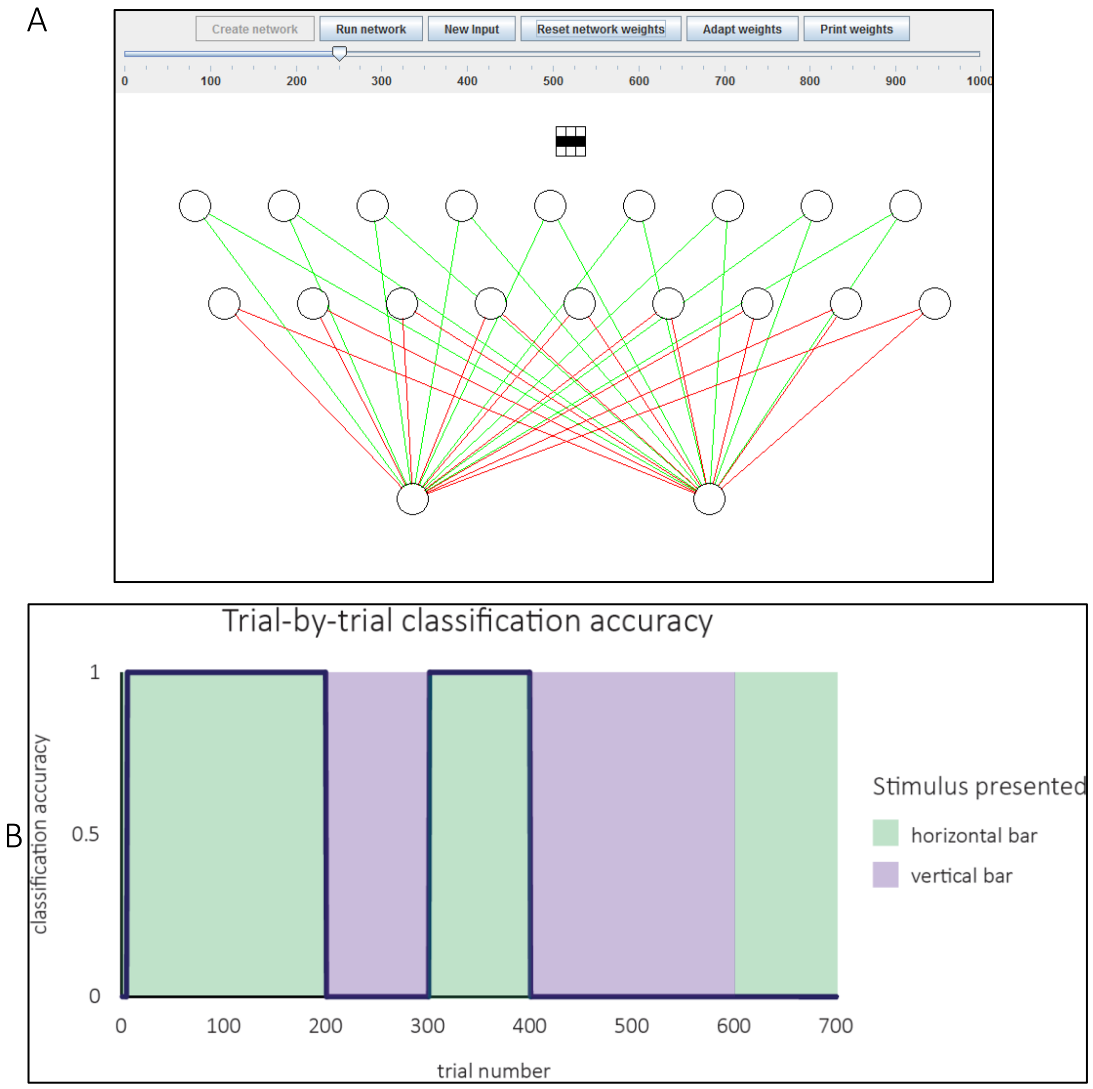

Figure 4. Visual categorization task with a two-dimensional binary input. A] Dark input squares provide excitatory input into the top layer (excitatory connections in green). Blank input squares provide input into the inhibitory layer (inhibitory connections in red). B] Classification performance on 100 trial blocks of serial presentation of either a horizontal bar input or a vertical bar input. The network is unable to categorize the inputs.

A supervised learning rule updated the weights with every iteration after the STDP learning rule. Our rule linearly scaled the weight changes with the correct or incorrect trial classification history of the system, such that the left output neuron should fire if the input was a horizontal bar, and the right output neuron should fire if the input was a vertical bar.

The classification performance of the system was poor. We suspect that the network architecture or the learning rule was ineffective for this task and possibly incompatible with the lower-level STDP weight updating rule that was already present in the network.

4 Conclusions

The simulation of a neural network with STDP does work. As demonstrated in Figure 3 and Figure 4, a neural network can be made with synapses that change over time due to spike timing. This can lead to the ability for the network to "learn" something as in the case of the classification mentioned in Section 3.2. This learning is not as adaptive as we had hoped, but it does demonstrate that the synaptic weights changed and that they led to some change in the network behavior.

The performance of the program is fast with no encumbrance due the neural network's multiple events. However, this would not be true if the program simulated a fully connected neural network in which each of the N neurons connected with N other neurons. In such a case, the number of events to handle would drop to $O(N^2)$. Given that many of the neurons do not have that many connections, and some only a few, it's better to say that on average amount of operations is $O(N\sqrt{N})$. In terms of speed, the program is not hindered by multiple events. Each event is added on to the queue, but the processing of any one event is fast enough that the rate of processing does not deter.

Overall, this method of network performance could be a viable simulation of networks. Given its poor performance in classification, the current state of the program would be better suited for simulating biological systems rather than applying it to neuromorphic machine learning.

4.1 Future Work

In order to create a more biologically relevant version of this event oriented STDP learning, the program needs to handle subthreshold values of current input and decay rate. This could be implemented within the event handling architecture by calculating the decay of the voltage increase only during updates to the neuron rather than any other time. In other words, the subthreshold value is only relevant when another neuron spikes and tries to activate the current neuron.

For a neuromorphic approach, it's possible that the STDP model requires a different form of neural networks than previously used. In such a case, a variety of different architectures could be tried rather than two layer or linear progressions usually seen.

Finally, as shown in Figure 2, we can easily create a randomly generated neural network. By employing graph theory, and specifically random geometric graphs, we can create a new form of neural network. By using smallest last ordering, we can color the graph in such a way that the first color set has the most connections, while the last set has the least. Each subsequent set will be directly connected to the previous (the previous being the pre neuron connections in a synapse, and the subsequent as the post neuron connections). Input could be randomly distributed across the first color set, and classification neurons could be the last color set of neurons.

The random geometric graph of a neural network could then have a resistance parameter that determines how much a synaptic weight can change regardless of the timing. This resistance would increase with each correct classification and decrease with each incorrect one, thus allowing some form of learning.

Hopefully, this novel way of handling neural events and STDP can lead to new networks that do not rely on preconceived architecture.

Acknowledgments

The authors thank the BENG 260 class of Fall 2014 for enduring our repetitive presentations of this idea and this project, especially the teaching assistants Chul Kim, Bruno Pedroni, and Jonathan Garcia.

References

[1] T. Serrano-Gotarredona, T. Masquelier, T. Prodromakis, G. Indiveri, and B. Linares-Barranco, "STDP and STDP variations with memristors for spiking neuromorphic learning systems.," *Front. Neurosci.*, vol. 7, no. February, p. 2, Jan. 2013.

[2] S. H. Jo, T. Chang, I. Ebong, B. B. Bhadviya, P. Mazumder, and W. Lu, “Nanoscale memristor device as synapse in neuromorphic systems.,” *Nano Lett.*, vol. 10, no. 4, pp. 1297–301, Apr. 2010.

[3] D. Kuzum, R. G. D. Jeyasingh, B. Lee, and H.-S. P. Wong, “Nanoelectronic programmable synapses based on phase change materials for brain-inspired computing,” *Nano Lett.*, vol. 12, pp. 2179–2186, 2012.

[4] C. Zamarreño-Ramos, L. a Camuñas-Mesa, J. a Pérez-Carrasco, T. Masquelier, T. Serrano-Gotarredona, and B. Linares-Barranco, “On spike-timing-dependent-plasticity, memristive devices, and building a self-learning visual cortex.,” *Front. Neurosci.*, vol. 5, no. March, p. 26, Jan. 2011.

[5] S. Song, K. D. Miller, and L. F. Abbott, “Competitive Hebbian learning through spike-timing-dependent synaptic plasticity,” *Nature*, vol. 3, no. 9, pp. 919–926, 2000.