Supplement to DARPA Quarterly Report Q2

Task 1.2 Biologically motivated spike-timing dependent plasticity (STDP) learning

1. Spiking Neural Network Model and STDP Rules

We first re-implemented the spiking neural network (SNN) model for MNIST handwritten digit classification from [1]. We tried models of various sizes (100 and 400 neurons), and found that classification results were similar to those reported in [1]. To learn the synaptic weights of the SNN, we implemented four different spike time-dependent (STDP) rules. All were implemented in an "online" fashion, where we only keep track of a "trace", or a memory, of the most recent spike (*temporal nearest neighbor*), which has arrived at a post- and presynaptic neurons. The first was online STDP without presynaptic weight modification, the second added an exponential dependence on the previous synaptic weight, the third included a rule for decreasing synaptic weight on presynaptic spikes, and the fourth was a combination of the second and third rules together.

The SNN model from [1] is illustrated below (Figure 1). The training of the model is as follows: At first, all neuron activations in the excitatory layer are set below their resting potentials. For 350ms (or 700 computation steps, with a simulation time step of 0.5ms), a single training image is presented to the network. For the MNIST digit dataset, each data sample is a 28 x 28 grayscale pixelated image of a hand-drawn digit from 0 to 9. The grayscale value of each pixel of the image determines the mean firing rate of a Poisson spike train it corresponds to. We include a synapse from all input Poisson spike trains to each neuron in the excitatory layer. There are then 784 x n synapses from input to the excitatory layer, if there are n neurons in the excitatory layer. The goal of the training procedure is to learn a setting of these weights that will cause distinct excitatory neurons to fire for distinct digit classes. Those neurons which fire most for a certain digit in the training phase we label with that digit's value. In the test phase, we classify new data examples by taking the majority vote of the labels of those neurons ,which fire for a test data sample.

2. Spiking Neural Network for MNIST Handwritten Digit Classification

2.1 - Description of ETH SNN Model

The excitatory neurons are connected in a one-to-one fashion with an equally-sized layer of inhibitory neurons, each of which *laterally inhibit* all excitatory neurons but the one which it received its synaptic connection from. The lateral inhibition mechanism typically causes a single neuron to fire strongly for a given data sample in a "winner take all" fashion, allowing the neuron which fires to adjust its weights to remember data samples of a similar shape and pixel intensity.

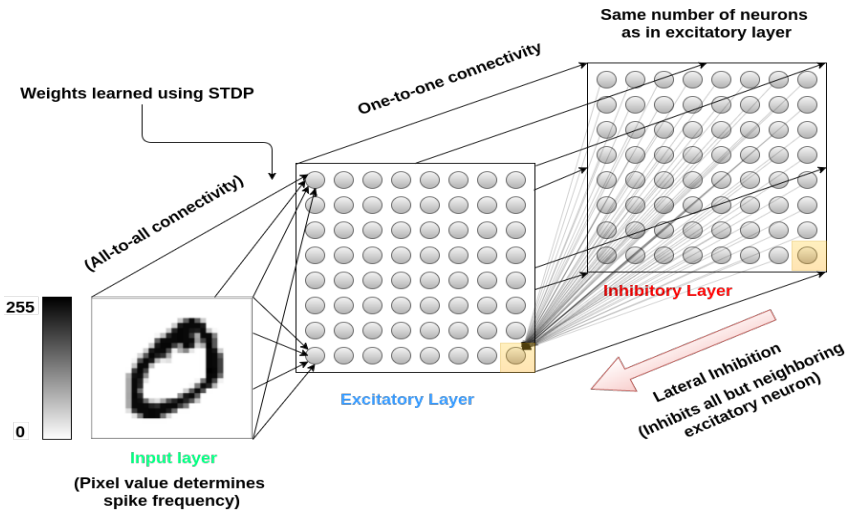

Figure 1: ETH Spiking Neural Network Model for MNIST Digit Recognition

2.2 - Online STDP Rules

To implement STDP in an online fashion, a pre- and postsynaptic trace is kept as a brief record of spike activity. On a spike from a presynaptic neuron, the presynaptic trace, a_{pre}, is set to 1, and on a spike from a postsynaptic neuron, the postsynaptic trace, a_{post}, is set to 1. Both exponentially decay towards zero otherwise, as given by:

$$\tau_{pre}\frac{d\,a_{pre}}{dt} = -a_{pre}, \quad \tau_{post}\frac{d\,a_{post}}{dt} = -a_{post},$$

where τ_{pre}, τ_{post} are time constants given as 1ms and 2ms, respectively.

The online STDP rule without presynaptic weight modification is given by the following equation, where Δw denotes the relative weight change on the affected synapse, η denotes a learning rate (typically set to 0.01), w_{max} indicates the maximum value a weight can attain, and μ denotes a weight dependence term, which is set to 1 for best results:

$$\Delta w = \eta a_{pre}(w_{max} - w)^{\mu}$$

This learning rule is illustrated below (Figure 2).

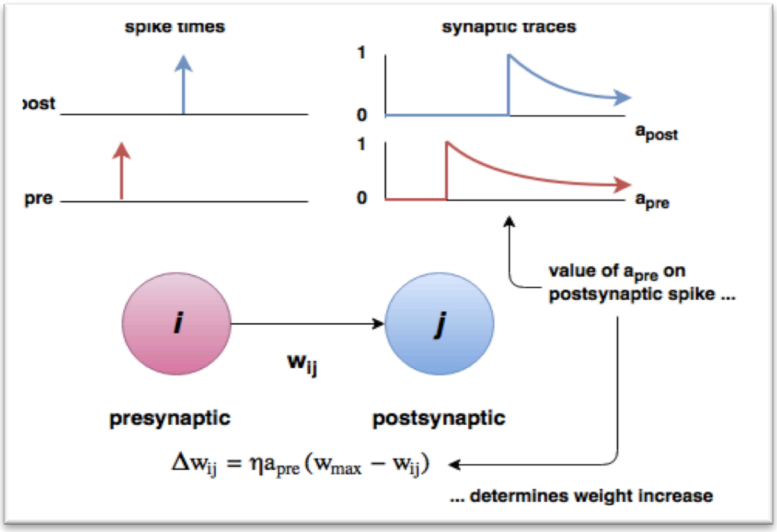

Figure 2: STDP without presynaptic modification

The online STDP rule without presynaptic weight modification and exponential weight dependence is given by

$$\Delta w = \eta a_{pre} e^{-\beta(w_{max}-w)},$$

where β determines the strength of the weight dependence, and is typically chosen to be 1. The online STDP rule with presynaptic weight modification is given by the following two rules:

$$\Delta w = \eta_{post} a_{pre} (w_{max} - w)^{\mu}, \ \Delta w = \eta_{pre} a_{post} w^{\mu},$$

where η_{post} and η_{pre} are post- and presynaptic learning rates typically set to 0.01 and 0.0001, respectively. Finally, the online STDP rule which is a combination of the second and third STDP rules is given by the following two rules:

$$\Delta w = \eta_{post} a_{pre} e^{-\beta(w_{max}-w)}, \ \Delta w = \eta_{pre} a_{post} e^{-\beta w},$$

After each training step, the synaptic weights of each excitatory neuron are *normalized* so that they do not grow so large as to fire indiscriminately for every possible input sample. Furthermore, the network is run for 150ms (300 computation steps with 0.5ms simulation time step) after each input sample to allow the activations of the excitatory neurons to decay back to their resting potential. We also implement an adaptive membrane threshold potential for each excitatory neuron, which stipulates that the neuron's threshold is not only determined by a fixed threshold voltage v_{thr}, but by the sum $v_{thr}+\theta_i$, where θ_i increases by a small fixed constant each time the i-th neuron fires, and is decaying exponentially otherwise.

On the next page, we show a plot of the activations of a spiking neural network as above, with only 4 excitatory and inhibitory neurons, and a plot of the change in the weights of the synapses from the input to the "0"-indexed excitatory neuron, both over a single input presentation and rest period (500ms or 1000 time steps are 0.5ms simulation time step). As you may observe, during this iteration, only the "0"-

th neuron ever reaches its threshold and spikes, laterally inhibiting all other excitatory neurons in the process.

Notice that the increase in synaptic weight corresponds in time to the firing of the excitatory neuron (in this case, the postsynpatic neuron), and are larger when the difference between presynaptic and postsynaptic spikes is smaller. Those synaptic weights, which do not change correspond to presynaptic neurons that haven't fired; i.e., their mean firing rate is either zero or close enough to zero so that they hadn't emitted a spike during the training iteration. Thanks to the adaptive membrane potential mechanism, one can also observe the increase in the threshold parameter each time the "0"-th neuron fires a spike. After the 350ms training portion (or 700 timesteps), the 150ms resting portion (or 300 timesteps) allows the activations of each of the excitatory neurons to settle to their resting potential.

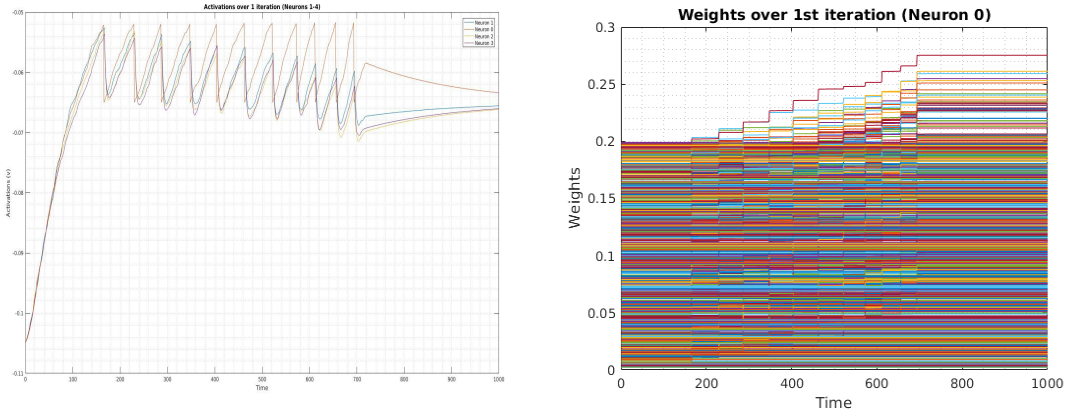

Figure 3: Neuron activations (left) and corresponding weight changes (right)

We proposed the training techniques of *input smoothing*, in which the mean firing rate of each of the input Poisson spike trains was averaged with each of its neighbors' to the immediate left, right, top, and bottom, and *weight smoothing*, in which, during the training phase, after each input example was presented to the network and network weights had been modified, we averaged the weights from the input to a single neuron in the excitatory layer using the neighborhood smoothing described above.

To compare the four STDP rules and the smoothing operations, we trained a 400 excitatory-inhibitory neuron SNN on the MNIST handwritten digit training dataset (60K examples), and tested it on the corresponding test dataset (10K examples), for all possible combinations of STDP rule and smoothing operation. The results are below (Table 1).

Table 1: Classification results: ETH model and modifications

	Standard Online STDP	**Exponential Weight Dependence**	**With Presynaptic Weight Modification**	**Both**
Diehl & Cook Algorithm	89.35	89.28	90.54	88.51
With Input Smoothing	88.42	88.75	90.23	89.23
With Weight Smoothing	88.90	89.85	90.02	89.54

3. Novel Approaches: Convolutional Spiking Neural Networks

3.1 - Proposed Network Architecture

To extend the spiking neural network (SNN) model from [1], we first implemented a single layer of convolution "features" or "patches". This layer is adopted from the widely-adopted convolutional neural network originating from deep learning research [2]. To accomplish this, we use convolution windows of size $k \times k$, with a stride of size s. In our case, a convolution window corresponds to a function which maps a section of input to a single excitatory neuron. A single convolution feature maps regularly spaced subsections of the input to a "patch" of excitatory neurons, all via shared synaptic weights (a single set of weights is learned for a single convolution feature). We use multiple of these convolution features, mapping to distinct excitatory neuron patches, in order to learn multiple different features of the input data. Our goal is to learn a setting of these weights such that we may separate the input data (in the current study, the MNIST handwritten digit dataset) into salient and frequent categories of features, as evidenced by the firing pattern of the excitatory neurons, so that we might later compose them in order to classify the digits with their respective categorical labels.

As with the SNN from [1], the weights from input (convolution windows) to excitatory layer (subdivided into convolution patches) are learned in an unsupervised manner using spike-timing dependent plasticity (STDP) in the training phase. In the test phase, these weights are held constant, and we use them to predict the labels of a held-out test dataset. The labels of new data are determined first by assigning labels to the excitatory neurons based on their spiking activity on training data, and then using these labels in a majority "vote" for the classification of new data samples.

The number of excitatory neurons per convolutional patch can be determined via the constants k and s, and is given by

$$(\frac{\sqrt{n}-k}{s}+1)^2,$$

where n is the dimensionality of the input, assuming n is a square number. The number of convolution features is the network designer's choice, which we denote by m, but with more features come greater representation capacity and computational cost. As a first pass, we chose to use exactly m inhibitory neurons, each of which correspond to a single convolution patch. Each neuron in a convolution patch connects to this inhibitory neuron, which is connected to all other neurons of all other patches but the one it receives its connections from. This has the effect of allowing a particular convolution patch to spike while damping out the activity of all others, creating competition to represent certain features of the input data. Below, we have provided a diagram of the network architecture (Figure 4).

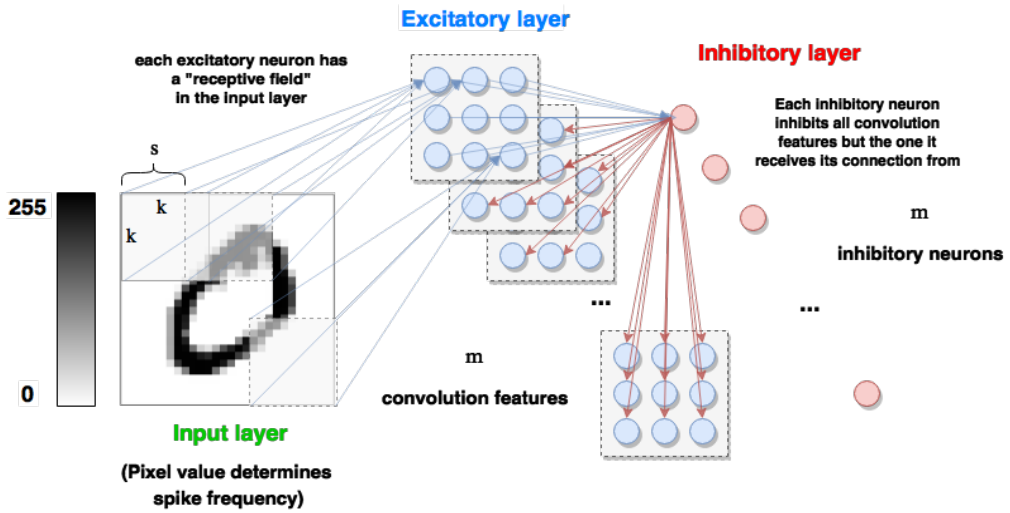

Figure 4: Convolutional Spiking Neural Network (Single Convolutional Layer)

3.2 - Experiments

We experiment with different choices of k and s and observe how the classification performance of the network degrades from that of the SNN model from [1]. We use the same training and test procedure from [1], but use only a single iteration through the entire training dataset (60,000 examples), because training the network is rather slow. We chose to set k = 27 and s = 1 for our first experiments, causing the convolution windows to cover almost the entire input space, and we therefore expect that the classification performance of the network will not degrade too much from the model from [1] given sufficient m. We then gradually decreased k, keeping s and m constant, expecting to see a gradual degradation in classification performance. The results are below in Figure 5; for reference, the test classification performance of the SNN model from [1] with 100 excitatory neurons is also plotted.

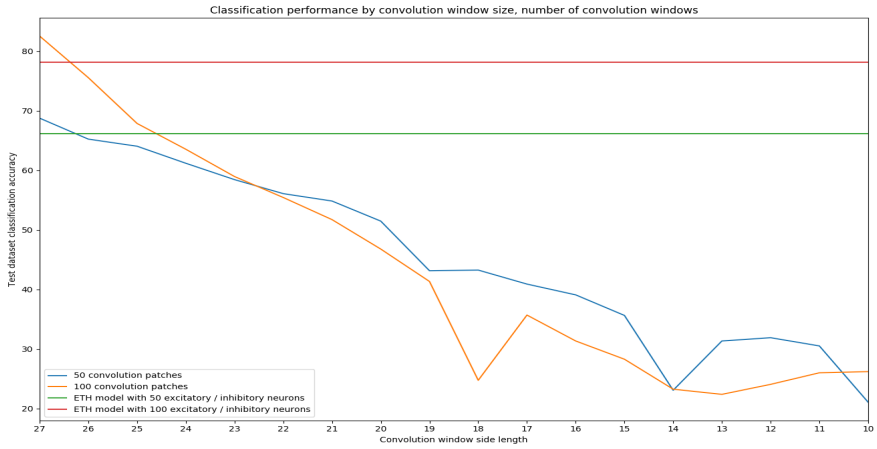

Figure 5: Classification accuracy by convolution size, number of convolution patches

3.3 - Weight Smoothing

We experimented with a weight smoothing technique, which can be described as follows: After each training sample has been run through the network (for the 500ms or 1,000 time steps at 0.5ms each), for each convolution patch, we take a weighted sum of the weights of the neuron which spiked the most during the training iteration and its neighbors in a horizontal / vertical lattice in the convolution patch, and copy this across all neurons in the same patch. For example, if the i-th neuron is the "winner" (having the most spikes within its patch), we compute

$$\frac{1}{2} W_{i,j} + \frac{1}{2} \sum_{i',j' \in \mathcal{N}_{i,j}} W_{i',j'},$$

where $W_{i,j}$ denotes the set of weights connected a convolution window (a portion of the input) to the (i, j)-th neuron in the convolution patch, and $\mathcal{N}_{i,j}$ is the set of indices of neighboring neurons of (i, j)-th neuron in the lattice, in the same convolution patch. See Figure 6 for an illustration of this procedure. This approach does not appear to improve the algorithm's performance, indeed, it often harms the classification performance of the original algorithm, so we do not report performance results from the networks we tested.

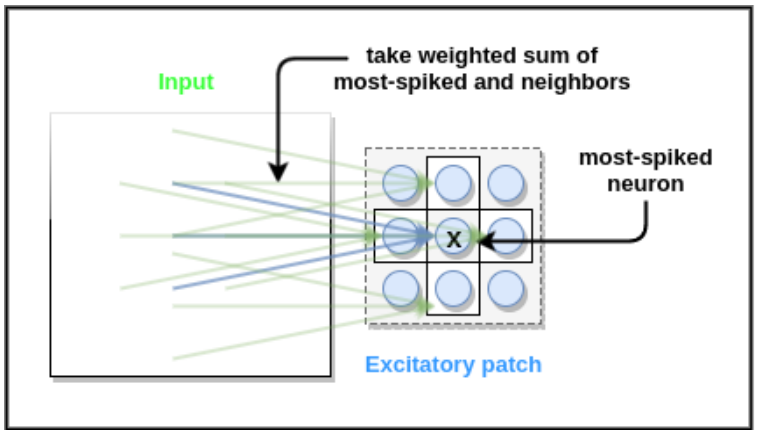

Figure 6: Weight smoothing after each training example

3.4 - Convolutional Spiking Neural Network With Between-Patch Connectivity

We describe a modification to our convolutional spiking neural network (SNN) model which includes a lattice connectivity between pairs of neighboring convolutional patches. The weights connectivity pairs of convolutional patches are learned via the same STDP rule which is used to learn the weights from input to the patches. We also describe a new procedure for inhibiting excitatory neurons and assigning labels to new data which allows for the network to quickly learn features of input data.

Added Connectivity

Suppose there are m convolutional patches (or "features") of convolution neurons, and let p_i be the patch indexed by the integer i. If i is even, then we connect p_i to p_{i+1}; otherwise, we connect it to p_{i-1}. The connectivity pattern is as follows: for every neuron (i,j) in a patch, we connect it to all neurons $(i+1,j),(i-1,j),(i,j+1),(i,j-1)$ in the neighboring patch, as long as the indices remain within the patch. This gives a horizontal and vertical lattice connectivity pattern between neighboring convolution patches. Note that there are only $\frac{m}{2}$ such lattice connections; p_0 and p_1 are connected, p_2 and p_3 are connected, up to p_{m-2} and p_{m-1}. The connectivity pattern is illustrated, for a single neuron in a single pair of patches, in Figure 7. The rest of the synapses are omitted for ease of visualization.

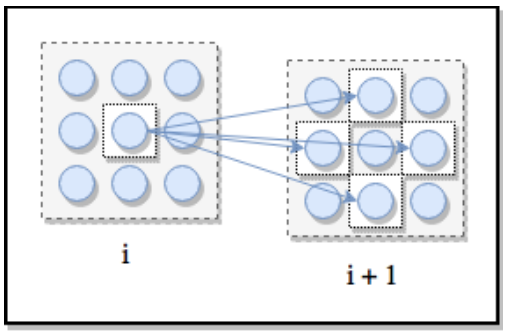

Figure 7: Connectivity between convolutional patches ("i" is even)

This is a first attempt at adding connectivity between patches. We may experiment with different lattice neighborhoods (to encompass more of the nearby input space), and with a different overarching connectivity pattern; e.g., instead of connectivity certain neighboring patches, we could connect all neighboring patches, or connect each patch to all others.

Inhibition

Since we wish to learn weights between neighboring neurons in neighboring patches, it no longer makes sense for a single inhibitory neuron to be fired by all neurons in a single patch, which would then inhibit all other patches totally. Instead, to allow for the learning of correlations between the patches, each excitatory neuron projects to its own inhibitory neuron. This inhibitory neuron, when it reaches its threshold and spikes, will inhibit the excitatory neuron in all other excitatory patches which reside in the same spatial location as the neuron from which it receives its synapse. This inhibition scheme is depicted in Figure 8.

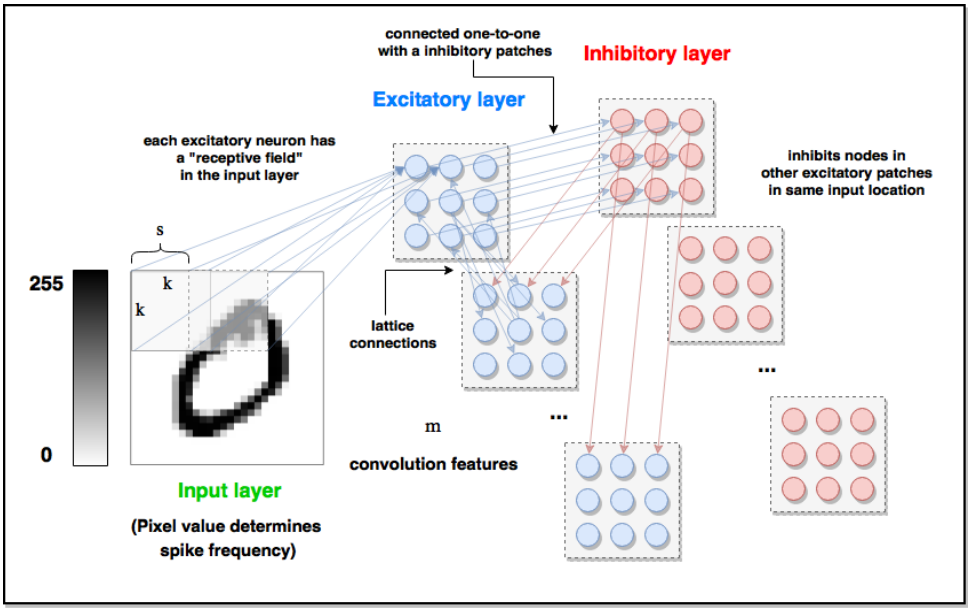

Figure 8: Convolutional SNN with between-patch connectivity

Labeling New Data

For this architecture modification, we implemented a different labeling scheme for the classification of new input data. Recall that, for the original SNN model, we label a new input datum with the label which corresponds to the high number of spikes from the excitatory layer. The neurons, in turn, are assigned digit labels based on the digit class for which they spike the most. Since each convolution patch learns a single "feature" of the input (the weights on the connections from input to all neurons in the same patch are identical), we choose to include only the counts of spikes on a new input datum from only the neuron in each patch which spiked the most while it was presented to the network. The justification for this was that many of the neurons in the network would spike infrequently on any given input, but only a few neurons would spike consistently. Removing the counts from the infrequently spiking neurons allows us to remove the "junk" counts from the labeling scheme.

3.5 - Preliminary Results

We are currently running a number of simulations of the above described network, for a variety of settings of convolution size, stride size, and number of convolution patches. We include in Figure 9, 10, and 11 plots of the accuracy of the network with the added pairwise lattice connectivity, evaluated every 100 training examples. We set the labels of the neurons using the spikes resulting from the last 100 examples, and use these labels to classify the next 100 examples. Since we are evaluating on so few data at each iteration, the accuracy curve exhibits high variance. Each of these networks have 50 convolution features.

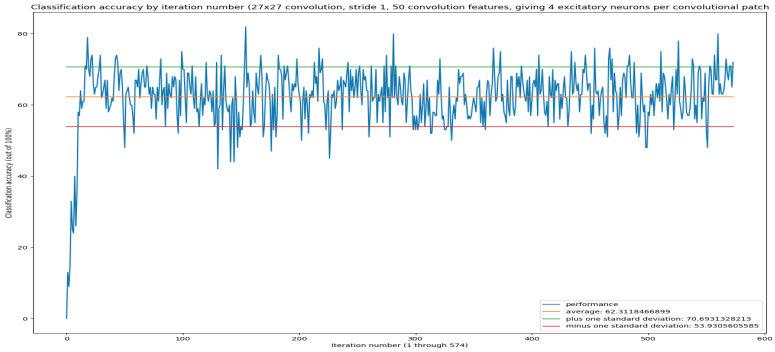

Figure 9: 27x27 convolutions, stride 1, 50 features

(62.3% average accuracy vs. 61.4% without lattice connections)

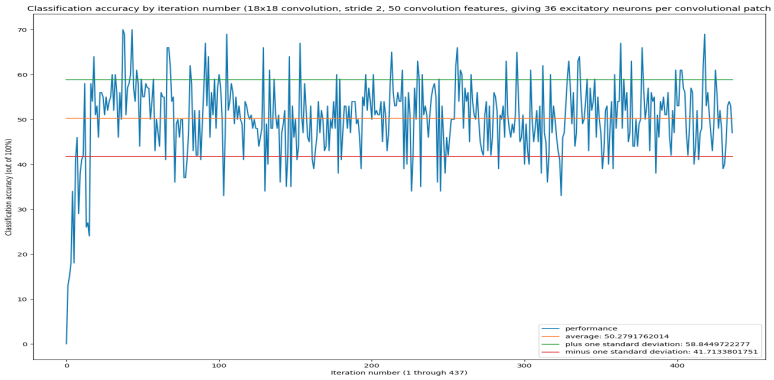

Figure 10: 18x18 convolutions, stride 2, 50 features

(50.3% average accuracy vs. 43.25% without lattice connections)

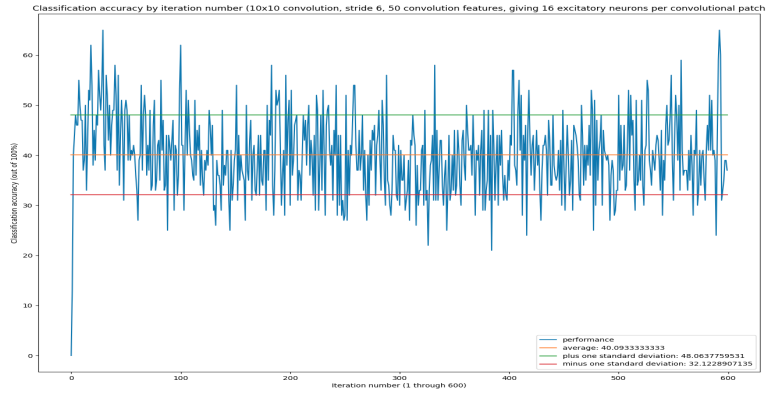

Figure 11: 10x10 convolutions, stride 6, 50 features

(40.1% average accuracy vs. 21.08% without lattice connections)

4. Discussion on SNN Computational Complexity

We describe the computational requirements of our spiking neural network models and compare it to that required by a convolutional neural network in the deep learning literature. Our spiking neural network models are programmed in Python, using the BRIAN neural network simulator [3].

4.1 - Spiking Neural Network Computation - MNIST

We consider a single training "iteration" of the spiking neural network model from [2] (see Figure 1). Recall that, for each input example, we "present" the input to the network for 350ms and allow the network to relax back to equilibrium for 150ms, for a total of 500ms per iteration (or 1,000 integration steps with a 0.5ms time step). The input is encoded as Poisson spike trains [3], as described the document "Spiking Neural Network Model and STDP Rules", and each are connected to every neuron in an arbitrarily-sized layer of excitatory neurons. These Poisson spike trains emit a spike with a certain probability at each time step, which increases with the time elapsed since it last emitted a spike. So, for the MNIST digit dataset, we consider 28 * 28 = 784 independent Poisson processes, each of which must be solved for in each time step of each training iteration.

The neurons in the excitatory and inhibitory layers of the network are modeled with leaky integrate-and-fire neurons, whose membrane potentials are described by

$$\tau \frac{dV}{dt} = (E_{rest} - V) + g_e(E_{exc} - V) + g_i(E_{inh} - V),$$

Where E_{rest} is the resting potential of the neuron, E_{exc} and E_{inh} are the equilibrium potentials of excitatory and inhibitory neurons, respectively, τ is a biologically plausible time constant, and g_e and g_i are the conductances of excitatory and inhibitory synapses, respectively. The excitatory and inhibitory conductances update rule is given by

$$\tau_{g_e} \frac{dg_e}{dt} = -g_e, \quad \tau_{g_i} \frac{dg_i}{dt} = -g_i,$$

Where τ_{g_e}, τ_{g_i} are biologically plausible time constants. For each excitatory and inhibitory neuron, we solve the above equations at each time step. Assuming there are n_e excitatory neurons, and $n_i = n_e$ inhibitory neurons, we must solve $2*(n_e + n_i) + 784$ equations at each time step, accounting for all input Poisson spike trains spikes and excitatory, inhibitory neurons' membrane potentials and synapse conductance.

At the beginning of the training phase, the weights on the synapses from input to the layer of excitatory neurons are chosen uniformly at random in the range [0.01, 0.3], and throughout the training phase, these are updated using our chosen STDP rule. For each input Poisson spike train and excitatory neuron, a "trace" tells us how soon in the past this neuron has fired. It's general form is given by

$$\tau_a \frac{da_{\square}}{dt} = -a,$$

where a is the trace, which is set to 1 when the neuron fires, and updates according to the equation above otherwise, and τ_a is a time constant, which may be chosen to be different for presynaptic (input Poisson spike trains) or postsynaptic (excitatory layer) neurons. When an input spike train emits a spike, we update the synapse conductance of those neurons it is connected to (all excitatory neurons) by the weight along its connection to it. When an excitatory neuron emits a spike, we apply the STDP update rule to the connections from the input layer to itself, and update the synapse conductance of the inhibitory neuron to which it connects. Since there are approximately 15 spikes emitted from the excitatory layer per training iteration, there are approximately $15*784$ weight updates per iteration.

All together, over one training iteration (in which a single training example is presented to the network), there is on the order of $1,000 * (2 * (n_e + n_i) + 784) + 700 * (784 * 15)$ operations to compute (we only calculate weight updates during the active input period; the network is at "rest" for the last 300 time steps). For example, with a network of 400 excitatory and inhibitory neurons (which gives approximately 90% classification accuracy, depending on the STDP rule utilized), this amounts to approximately 10.6 million operations per training example.

Since the STDP learning rule is only applied locally (i.e., between any two given neurons), we don't have to wait to propagate signals through the network before computing a backward pass in order, as in gradient-based deep learning methods. Additionally, very little memory is required to simulate these networks. We keep track of matrices of weights, membrane potentials, synapse conductances, and synaptic traces only, and even with large networks, the memory overhead for the network is manageable on laptop machines.

4.2 - Convolutional Neural Network Computation – MNIST

We estimate the computational burden imposed by a simple convolutional network used to classify the MNIST handwritten digit dataset. The described network achieves approximately 99% classification accuracy on the test dataset after training on some 500,000 images (we subsample 50 images from the training dataset 10,000 times). All convolutional layers of the network utilize zero-padding of size 2 on all edges of the images. The network consists of a convolutional layer, with patch size 5x5 and stride 1, with 32 convolution features (giving 28x28x32 dimensionality), then a max-pooling layer with patch size 2x2 and stride 2 (giving 14x14x32 dimensionality), another convolutional layer, again with patch size 5x5 and stride 1, with 64 convolution features (giving 14x14x64 dimensionality) another max-pooling layer, again with patch size 2x2 and stride 2 (giving 7x7x64 dimensionality), a fully-connected layer of size 1024, and a logistic regression classifier of dimensionality 10, from which we extract the digit label using the softmax function.

For a single training example, the first convolutional layer of this network requires the computation of 28*28*32 *ReLU*() (rectified linear unit) functions, each on the sum of 5*5 entries in the input layer multiplied by their connection weights, for a total of on the order of 627,200 operations. The first max-pooling layer simply requires computing the max operation over some 14*14*32 2x2 matrices, for a total of 6,272 operations. Similarly, the second convolutional layer and the second max-pooling layer require some 313,600 and 3,136 operations, respectively. The fully-connected layer requires computing 1,024 ReLU functions, each on the sum of products of the entries in the previous layer by their edge weights, resulting in on the order of $7*7*64*1024 = 3,211,264$ operations. Finally, the logistic regression requires another 1024*10 operations, and then simply takes the *argmax*() function of the output 10-

dimensional vector. Therefore, the forward pass of the network takes approximately operations. The backward pass, on the other hand, requiring computing partial derivatives for each parameter in each of these functions, but its complexity is the same as the complexity of the forward pass. To underestimate, we say that the backward pass requires, again, 4,171,712 operations, and so we estimate that this network requires approximately 8.3 million operations per training example.

4.3 - Classification Results

We include a plot of the performance of the spiking neural network model from [1] on the MNIST handwritten digit test dataset in Figure 2. This plot gives the classification accuracy of the model as a function of the number of excitatory and inhibitory neurons in the network. Each network was trained using a single pass through the entire 60,000 digit dataset. We have also include the classification accuracy of the above described convolutional neural network, also on the test dataset. We trained and tested the SNN model using excitatory, inhibitory layer sizes of 25, 50, 100, 200, 400, 800, 1600, 3200, 6400. The classification accuracies for these particular runs increase monotonically, as can be seen in the figure.

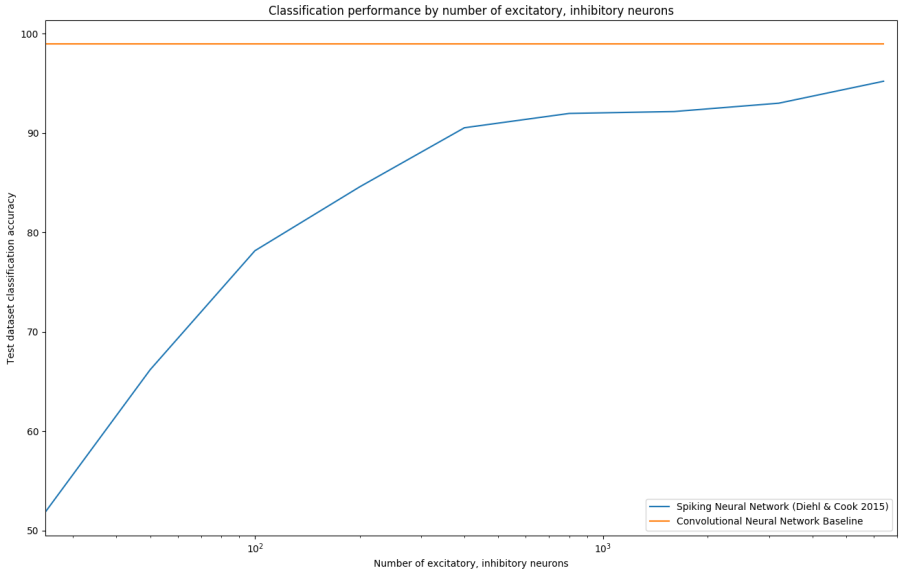

Figure 12: Classification accuracy of ETH SNN model by network size

4.4 - Possible Improvements - Spiking Neural Network

One simple way to reduce the time complexity of the spiking neural network model from [1] is to reduce the number of time steps used per iteration of the training and test phases. Currently, we “present” the network with an input for 350ms (700 time steps at a 0.5ms time increment), and then “turn off” the input for 150ms (300 time steps), allowing the membrane potentials and synapse conductances of the excitatory and inhibitory neurons to decay back towards their resting values. Reducing the number of time steps would require adjusting the parameter of the equations which govern the neurons in the network in order for the neurons to respond to the input data more quickly.

A current severe limitation of the BRIAN neural network simulation software (using version 1.4.3) is that it works most efficiently as a single-core process. Upgrading to version 2.2.x should allow us to take advantage of multi-core and GPU processing in order to speed up our experiments, but we must be careful to replicate the code correctly and consider all the parallel processing options available.

5. References

[1] Diehl, P. & Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers In Computational Neuroscience*, *9*. doi:10.3389/fncom.2015.00099

[2] Krizhevsky, A., Sutskever, I., & Hinton, G. (2012). *ImageNet Classification with Deep Convolutional Neural Networks*. *Papers.nips.cc*.

[3] Goodman DF and Brette R (2009). The Brian simulator. *Front Neurosci* doi:10.3389/neuro.01.026.2009

[4] http://www.cns.nyu.edu/~david/handouts/poisson.pdf