Unsupervised Learning with Self-Organizing Spiking Neural Networks

Hananel Hazan*, Daniel Saunders†, Darpan T. Sanghavi‡,
Hava Siegelmann§, Robert Kozma¶
College of Information and Computer Sciences
University of Massachusetts Amherst
140 Governors Drive
Amherst, MA 01003, USA
Email: {hhazan*, djsaunde†, dsanghavi‡, hava§, rkozma¶}@cs.umass.edu

***Abstract*—We present a system comprising a hybridization of self-organized map (SOM) properties with spiking neural networks (SNNs) that retain many of the features of SOMs. Networks are trained in an unsupervised manner to learn a self-organized lattice of filters via excitatory-inhibitory interactions among populations of neurons. We develop and test various inhibition strategies, such as growing with inter-neuron distance and two distinct levels of inhibition. The quality of the unsupervised learning algorithm is evaluated using examples with known labels. Several biologically-inspired classification tools are proposed and compared, including population-level confidence rating, and n-grams using spike motif algorithm. Using the optimal choice of parameters, our approach produces improvements over state-of-art spiking neural networks.**

***Index Terms*—spiking neural networks, self-organizing map, clustering, classification**

I. Introduction

Powerful deep learning approaches that dominate AI today use “global” gradient-based learning [1], [2]. The success of deep learning [3] is based on using massive amounts of data to train the networks, which requires significant computational resources. In some practical problems, however, we may not have a sufficiently large data set to span the problem space, or we may need to make a decision fast, without an expensive training process. There are several approaches to circumvent the above mentioned constraints of deep learning. Some of these alternatives are based on local learning rules, such as the biologically motivated Spike-Timing Dependent Plasticity (STDP) [4]. These approaches address a trade-off between the inevitably decreased classification performance due to the unsupervised nature of the model, and the advantage provided by the distributed nature of the local training architecture. Substituting global learning with suitable local learning rules can provide a solution to the computational bottleneck of deep learning, by striking a balance between significantly increased learning speed and less requirements for computational memory resources at the cost of slightly reduced accuracy of learning and classification.

In order to run the training phase of the spiking network models, one records synaptic traces (for modifiable synapses) and conductances and neuron membrane potentials. Certain spiking neural networks have the potential advantage of being more memory efficient than their deep learning counterparts. Deep neural networks must cache the results of its layer-wise computation produced during its forward pass, and use these cached results to compute its backward pass. This effect is proportional to the number of layers utilized in the network. Studying the convergence of the training shows that the number of data samples needed to train the model to a reasonable level of accuracy was much fewer than in traditional networks.

There is extensive literature of spiking neural networks in various applications [5]. Depending on the requirements of the SNN simulations, various packages provide many different levels of biological plausibility. For example, the works by [6], [7] provide high biological realism, while [8] focus more on computational efficiency.

II. Related Work

Previous work of Diehl and Cook [9] used a spiking neural network (SNN) to classify the MNIST handwritten digits after learning network weights without supervision and with several spike-timing-dependent plasticity (STDP) rules. Classification performance of their networks increases with the number of spiking neurons used, ultimately achieving approximately 95% accuracy using network of 6,400 excitatory and inhibitory neurons. A number of other SNNs trained with STDP are used to classify image data [10], [11]. The former uses Gabor filter features as a pre-processing input to their network, uses the rank-order coding scheme for input spikes, and classifies data with the winner-take-all strategy on the output layer. The latter is comprised of a difference of Gaussians pre-processing step, followed by convolutional and pooling layers, and whose output is trained on a linear SVM to perform classification. Other systems use spiking neurons, but are trained with supervision; e.g. [9], which was first trained as a deep neural network using back-propagation and later transferred to a spiking neural network without much loss in performance.

In this paper We extend the work presented in [12] by combining STDP rules with a version of the Self-Organizing Map (SOM) algorithm [13]. One of the properties of SOMs is the ability to cluster an unlabeled dataset in an unsupervised manner. The topological arrangement created by the SOM algorithm forms clusters that specialize and are unique to categories that exist in the input data. This clustering property is reminiscent of specialized areas in the primate cortex, where groups of neurons organize themselves according to similar functionality of parts of the primate body [14]. This property has never been addressed in the SNN literature. This paper aims to amend this issue. We show examples of SNN learned representations, and present classification results based on network filters and spiking activity on the MNIST handwritten digit dataset [15].

III. Methods

A. LIF neuron with synaptic conductances

One step towards a more biologically realistic system is to include a more powerful neural unit. The basic computational operations of a deep neural network (DNN) do not incorporate time, and unlike the all-or-nothing action potential of the biological neuron, they communicate by sending precise floating-point numbers downstream to the next layer of processing units. Moreover, the standard neuron that is used in traditional neuronal networks is synchronous, without a memory of its previous action, compared to spiking neurons which integrate time in their internal operation.

A natural choice for a biologically-inspired unit of computation is the leaky integrate-and-fire (LIF) spiking neuron [16]. These have the property of incorporating time in their operation, yet are computationally simple enough to solve problems involving large data. LIF neurons naturally incorporate time in their operation, dynamically changing membrane potential and spiking threshold, yet are computationally simple enough to scale to large networks trained on many data examples. In our models, we use a network of these units, some we call *excitatory*, and some *inhibitory*, and we additionally model synaptic conductances. Their membrane voltage v is given by

$$\tau \frac{dv}{dt} = (v_{rest} - v) + g_e(E_{exc} - v) + g_i(E_{inh} - v).$$

Where, v_{rest} is the resting membrane potential, E_{exc} and E_{inh} are the equilibrium potentials of excitatory and inhibitory synapses, respectively, and g_e and g_i are the conductances of excitatory and inhibitory synapses. The time constant τ is chosen to be longer for excitatory neurons than for inhibitory neurons. When a neurons membrane potential exceeds its membrane threshold v_{thresh}, it fires an action potential and resets back to v_{reset}. The neuron then undergoes a 5ms refractory period, during which time it cannot spike.

Synapses are modeled by conductance changes: a synapse increases its conductance at the moment a presynaptic action potential arrives by its synaptic weight w [6]. Otherwise, the conductance is decaying exponentially. The dynamics of the synaptic conductance are given by

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

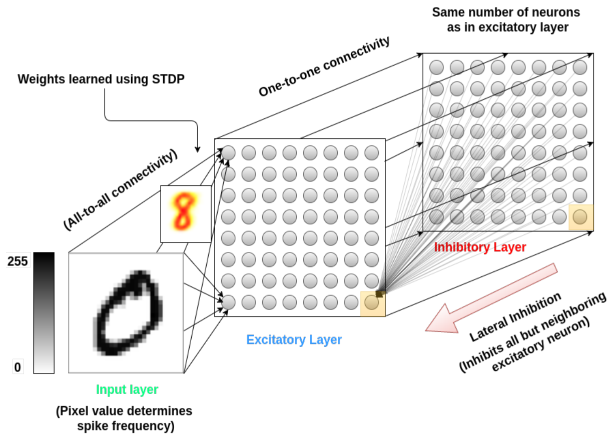
Same number of neurons as in excitatory layer
Weights learned using STDP
One-to-one connectivity
(All-to-all connectivity)
255
0
Inhibitory Layer
Lateral Inhibition (Inhibits all but neighboring excitatory neuron)
Excitatory Layer
Input layer
(Pixel value determines spike frequency)

Fig. 1: Spiking neural network architecture; from [12].

B. Spike-timing-dependent plasticity

Spike-timing-dependent plasticity (STDP) [4] as been used to modify the weights of the synapses connecting certain neurons. We call the neuron from which the synapse projects *presynaptic*, and the one to which it connects *postsynaptic*. For the sake of computational efficiency, we use online STDP, in which we record values x_{pre}. x_{post} for every synapse, a decaying memory of its recent presynaptic spiking history. Each time a presynaptic (postsynaptic) spike occurs at the synapse, x_{pre} (x_{post}) is set to 1; otherwise, it decays exponentially towards zero. When a spike arrives at the synapse, the weight change Δw is calculated using a STPD update rule. The rule we use in our simulations is given by

$$\Delta w = \begin{cases} \eta_{post} x_{pre}(w_{max} - w) & \text{on postsynaptic spike} \\ -\eta_{pre} x_{post} w & \text{on presynaptic spike} \end{cases} \quad (1)$$

The term η denotes a learning rate and w_{max} is the maximal allowed synaptic weight.

C. Network architecture

The spiking neural network has a multi-layer architecture, including input layer, excitatory layer, and inhibitory layer. The input layer is an array of $N \times N$ neurons, which correspond to the pixel of the input images in an image processing application. The excitatory and inhibitory layers have size of $K \times K$. Each input pixel projects to each of the excitatory nodes through modifiable synapses. Spikes are generated in the input neurons using Poisson spiking neurons, with average firing rate λ proportional to pixel intensity [5]. This process is simulated for a specific time duration, and the STDP rule is executed for the connection between the winning neuron in the excitatory layer and the corresponding input pixel. It is an important property of the model that once a neuron fires, it inhibits all other neurons according to the inhibition scheme described below. For further details of the basic model structure, including considerations on the system size, see [12].

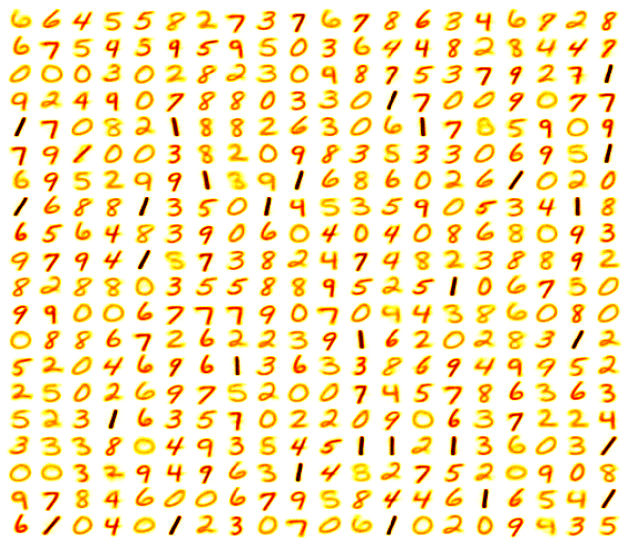

Fig. 2: Learned filters from baseline model

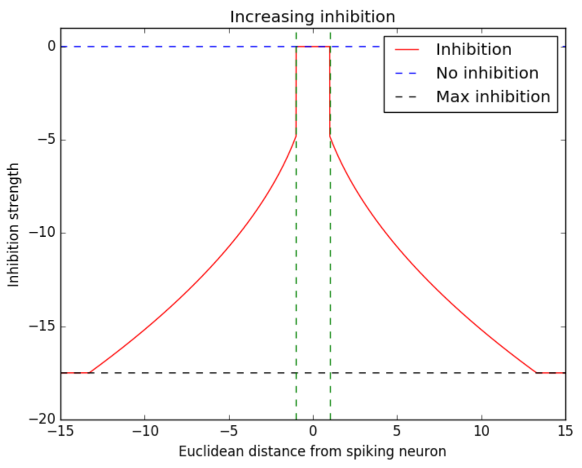

Fig. 3: Inhibition as a function of Euclidean distance

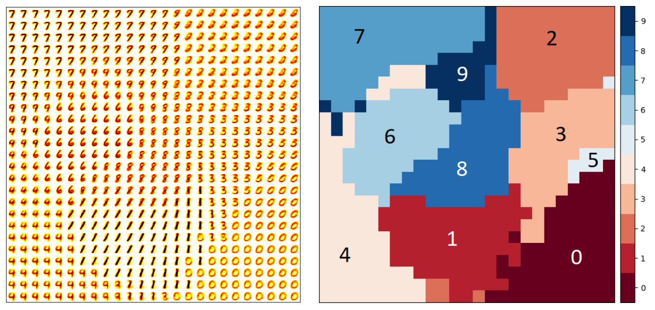

Fig. 4: *Increasing inhibition*: Filter map and class assignments

D. *Increasing inhibition with inter-neuron distance*

We make a change to the SNN architecture which is inspired by the self-organizing map (SOM) and corresponding algorithm [13]. This change is included in part to curb the degree of *competition* imposed by the connections from the inhibitory layer introduced in [17], in which network activation is too sparse throughout training to learn filters quickly. Additionally, this change causes network filters to self-organize into distinct clusters by classes of data.

Instead of inhibiting all other neurons at a large fixed rate as in [12], [17], we increase the level of inhibition with the distance between neurons. Inhibition level is increased in proportion to the square root of the Euclidean distance, similar to the SOM learning algorithm. This requires a parameter c_{inhib} which is multiplied by the distance to compute inhibition levels, as well as a parameter $c_{\max}$ that gives the maximum allowed inhibition (see Figure 3). With proper choice of c_{inhib}, when a neuron exceeds its firing threshold $v_{\text{threshold}}$, instead of inhibiting all other neurons from firing for the rest of the iteration, a neighborhood of nearby neurons will be weakly inhibited and may have the chance to fire. This encourages neighborhoods of neurons to fire for the same inputs and learn similar filters. See Figure 5 for plots of the effect of increasing inhibition to create clusters of filters. Compare this to Figure 2, in which filters are learned by using fixed inhibition with the learning algorithm described in [17].

To avoid confusion, we call SNNs with all variants of the *increasing inhibition* strategy *lattice map spiking neural networks*, abbreviated as LM-SNNs.

E. *Growing the inhibition level over the training phase*

We want to produce *individualized* filters as learned by the SNN presented in [17], yet retain the clustering of filters achieved by our *increasing inhibition* modification. Distinct filters are necessary to ensure that our learned representation contains as little redundancy as possible, making the best use of model capacity. To that end, we implemented another modification to the inhibition scheme, where the inhibition constant c_{inhib} grows on a linear schedule from a small value $c_{\min} \approx 0.5$ to a large value $c_{\max} \approx 17.5$. The *increasing inhibition* strategy is used as before; however, by the end of network training, the inhibition level is equivalent to that of [17]. In this setting, the filters self-organize into smoothly-varying clusters, and then individualize as the inhibition level becomes large. We also consider growing the inhibition level to $c_{\max}$ for some percentage of the training (p_{grow}) and holding it fixed for the rest ($1 - p_{\text{grow}}$). During the last $1 - p_{\text{grow}}$ proportion of the training phase, neuron filters are allowed to individualize more, allowing them to remember less frequent prototypical examples from the data.

F. *Two-level inhibition*

To remove the need to re-compute inhibitory synapse strengths throughout network training, we implemented a simple *two-level inhibition* scheme: For the first p_{low} proportion of the training, the network is trained inhibition level $c_{\min}$; for the last $1 - p_{\text{low}}$ proportion, the network is trained with $c_{\max}$. The inhibition level is not smoothly varied between the two levels, but jumps suddenly at the p_{low} mark. At the low

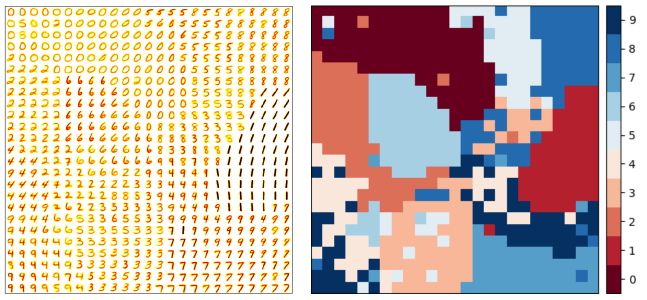
(a) 25x25 lattice of neuron filters (b) Neuron class assignments

Fig. 5: *Two-level inhibition*: Filter map and class assignments

inhibition level, large neighborhoods of excitatory neurons compete *together* to fire for certain types of inputs, eventually organizing their filters into a SOM-like representation of the dataset. At the high inhibition level, however, neurons typically maintain yet sharpen the filter acquired during the low inhibition portion of training. In this way, we obtain filter maps similar to those learned using the *growing inhibition* mechanism, but with a simpler implementation. This inhibition strategy represents a middle ground between that of [17] and our *increasing inhibition* scheme.

See Figure 5 for an example learned filter map and neuron class assignments. There is some degree of clustering of the filters; however, as the inhibition level approaches that of [17], they may eventually move away from the digit originally representing on their weights, *fragmenting* the filter clustering. The degree of this fragmentation depends on the choice of p_{low}: with more time training with the $c_{\max}$ inhibition level, the more likely a neuron is to change its filter to represent data outside of its originally converged class.

G. Evaluating learned representations

Although LM-SNNs are trained in an unsupervised manner, we may want to evaluate the quality of the representations they learn. The dataset representation is encoded in the learned weights of synapses connecting the input and excitatory layers. We use the activity of the neurons in the excitatory layer with their filter weights held fixed to (1) say what it means to *represent* input data from a certain class, and (2) *classify* new inputs based on historical network activity.

We perform a two-step procedure before the test phase to *label* excitatory neurons with the input category we believe they represent, and then *classify* new data based on these labels and the spiking activity of the excitatory neurons on it, as in [17]. We call the first step *neuron labeling* and the second *voting*. For some *voting schemes*, neurons are assigned the label of the input class for which they have fired most on average, and these labels are used to classify new data based on network activation. In the *all* voting scheme, all excitatory neuron spikes are counted in the "vote" for the label of the input example. In the *confidence weighting* voting scheme, we record the proportions of spikes each neuron fires for each input class, and used a weighted sum of these proportions and network activity to classify new inputs. These evaluation strategies are reminiscent of related work [18] in which the timing of individual neurons' spikes are discarded in favor of a rate-based code.

In order to leverage the information contained in the timing of the spikes, we also design an n-gram based testing scheme that considers the order of spiking to make a prediction. N-grams have long been used for sequential modeling, particularly in computational linguistics. Our *n-gram* scheme, like the rate-based schemes, also follows a two-step procedure: a *learning* phase to estimate the n-gram conditioned class probabilities from a subset of the training data, and a *voting* phase where the n-grams in the output spiking sequence "vote" for the classes. While this scheme does not exploit the spike timing explicitly, our motivation for now is only to demonstrate the importance of the information contained in spike ordering. There is growing evidence [19] on how most of the information in cortical neuronal networks is contained in the timing of individual spikes, and in particular on the crucial importance of the exact time of the first spike. Moreover, n-grams are able to identify repeating motifs [20] in the activation of synchronized bursts in cultured neuronal networks and can also classify stimuli based on the time to first spike [21].

The *distance* voting scheme is used to benchmark the performance of the spike-based schemes: new data is labeled with the class label of the neuron whose filter most closely matches the input data as measured by Euclidean distance.

In Section IV, we evaluate networks with the *all*, *confidence weighting*, *distance* and *n-gram* voting schemes. Other voting schemes were designed and tested, but eventually discarded in favor of those which produced the most consistent accuracy results.

H. Computational complexity

A potential advantage of the spiking neural networks approach is the amount of computation required for training and testing relative to large deep learning models. SNNs trained with spike-timing-dependent plasticity are trained in *real time*; i.e., there is no need for both a forward and backward pass through the network, as with deep learning models trained with back-propagation [22]. Instead, all synapse weights evolve independently, according to the relative timing of spikes emitting by their pre- and post-synaptic neurons. This *local learning* rule is the key to training large networks while removing interdependence between learned parameters.

Let m denote the number of excitatory and inhibitory neurons, n the number of input neurons, and k the number of convolution patches in a C-SNN. Weights from input to excitatory layers are continuously recorded, incurring a memory cost of nm. Membrane voltages for all excitatory, inhibitory neurons are updated on each timestep, requiring $2m$ calculations, and also update synaptic conductances for all inter-population connections, totaling $nm + m + (m-1)^2$ updates per time step, which incur an equally-sized memory cost. Synaptic traces are recorded for input to excitatory connections; i.e., nm values recording a fading memory of pre- and post-synaptic spike activity. Synapse weight updates are performed as needed: in the event of a spike from a pre- or post-synaptic neuron, we update the strength of the connecting synapse according to the value of the synaptic traces and the form of the STDP learning rule. Depending on the input data, both input and excitatory neurons spike at average rates r_X, r_E, respectively, requiring approximately $r_X m + r_E n$ weight updates per timestep. We estimated that $r_X \approx 4 \times 10^{-3}$ and $r_E \approx 2.75 \times 10^{-4}$ while training C-SNN networks of several sizes on the MNIST digit dataset for the first 1000 examples with a 0.5ms timestep. As the training phase progresses, fewer weight updates are made since network weights have converged to useful filters.

To simulate network dynamics and spiking activity for T timesteps per example, the SNN and LM-SNN architectures require $T(3m+2nm+(m-1)^2+r_X m+r_E n) = \mathcal{O}(T(nm+m^2))$ operations, and $3nm+(m-1)^2+3m = \mathcal{O}(nm+m^2)$ memory overhead per data item. We use a default of $T_1 = 700$ (350ms at 0.5ms timestep) for each example, and $T_2 = 300$ (150ms at 0.5ms timestep) for inter-example network relaxation periods. However, per-example simulation time T_1 can be reduced by appropriately adjusting simulation parameters to adapt filter weights at a quicker pace while maintaining stable network activity. Resetting of network state variables after each input example may allow the removal of the inter-example relaxation period.

Although the time and memory estimates are quadratic in the number of excitatory and inhibitory neurons, network simulation may tractably scale to larger or multi-layered SNNs by using parallel computation. Learning with STDP, there is no need to wait for a forward propagating signal to reach the network's output, or for an error signal back-propagation pass. Thus, network training may be massively parallelized to independently update synapse weights for all pairs of connected neurons. Improving network simulation to make better use of parallelization will drastically speed training and evaluation of spiking neural network architectures.

IV. Results

In the subsequent sections, we give quantitative results of variants of the LM-SNN models. We omit results on the *increasing* and *growing inhibition* strategies in favor of results using the simpler *two-level inhibition* strategy in Section IV-A. A large compilation of results demonstrates that network performance is robust to a wide range of hyper-parameter choices. This strategy is compared with a baseline SNN of [17] in Section IV-B, and is shown to outperform it, especially in the regime of small networks, and particularly so with the *n-gram* voting scheme. Section IV-C shows multiple passes through training data are required for successful training of progressively larger networks. An improved rate of convergence to near-optimal performance is quantitatively demonstrated in Section IV-D, using baseline SNNs of [17] to compare. Finally, in Section IV-E we discuss the robustness of LM-SNNs in the absence of inputs, showing a graceful degradation in performance.

A. Two-level inhibition

Networks are trained on a single pass over the training data and evaluated on all 10K test examples. Test accuracy results and single standard deviations are reported in Table I, each averaged over 10 independent trials. All networks are comprised of 625 excitatory and inhibitory neurons. Results are demonstrated for a number of choices of parameters p_{low}, c_{min}, and c_{max}.

B. Comparing baseline SNN and two-level inhibition

To compare how the two-level inhibition strategy performs with respect to the networks of [17], we present in Table II a comparison of their accuracies over several settings of the number of neurons, using the *confidence*, *all*, *distance* and *n-gram* voting schemes. All networks are trained for 60K iterations (a single pass through the training data) and evaluated on the 10K examples from the test data. The two-level inhibition hyper-parameters are fixed to $p_{\text{low}} = 0.1, c_{\text{min}} = 0.1, c_{\text{max}} = 20.0$. For the *n-gram* scheme, 12K examples are used for the learning phase. Preliminary tests showed that bi-grams gave the best performance, and hence we fixed $n = 2$. 5 independent experiments with different initial configurations and Poisson spike trains were run, and their results are averaged and reported along with a single standard deviation.

One can notice the superiority of the *confidence* scheme to the *all* scheme, the *distance* scheme to the *confidence* scheme, and the *n-gram* scheme to the *distance* scheme. While the *confidence*, *all* and *n-gram* schemes use the activity of the network in order to classify new data, the *distance* scheme simply labels new inputs with the label of the neuron whose filter most closely matches the input. This last evaluation scheme is reminiscent of the one-nearest neighbor algorithm. However, our spiking neural networks learn prototypical data vectors, whereas the one-nearest neighbor method stores the entire dataset to use during evaluation.

C. Training larger networks

One can notice in Table II that networks tend to achieve better accuracy by using more neurons. However, training networks with more than 900 neurons on a single pass through the training data shows a decrease in test performance. On the other hand, training networks with multiple passes through the data preserves the upward trend in performance (see Table III). The results for networks with 1,225 and 1,600 neurons suggest either or both of (1) the training algorithm for both of the baseline SNN and the LM-SNN does not make appropriate use of all data on a single training pass, or (2) network capacity is too large to adequately learn all filter weights with a single pass over the data. Inspection of the convergence of filter weights during training (data not shown) suggests that the training algorithm needs to be adjusted for greater data efficiency. The fact that training with more epochs improves

TABLE I: Two-level inhibition test accuracy (n_e, n_i = 625)

p_{low}	c_{min}	c_{max}	*distance*	*all*	*confidence*	p_{low}	c_{min}	c_{max}	*distance*	*all*	*confidence*
0.1	0.1	15.0	91.8 ± 0.63	91.4 ± 0.13	91.69 ± 0.63	0.25	0.1	15.0	91.51 ± 0.25	90.62 ± 0.26	90.97 ± 0.25
0.1	0.1	17.5	92.02 ± 0.38	91.26 ± 0.11	91.68 ± 0.38	0.25	0.1	17.5	91.83 ± 0.18	91.06 ± 0.18	91.54 ± 0.18
0.1	0.1	20.0	92.38 ± 0.49	91.54 ± 0.14	92.1 ± 0.49	0.25	0.1	20.0	92.16 ± 0.34	91.07 ± 0.29	91.83 ± 0.34
0.1	1.0	15.0	91.67 ± 0.63	91.12 ± 0.39	91.59 ± 0.63	0.25	1.0	15.0	91.36 ± 0.24	90.26 ± 0.21	90.77 ± 0.24
0.1	1.0	17.5	92.25 ± 0.42	91.32 ± 0.29	91.67 ± 0.42	0.25	1.0	17.5	91.78 ± 0.61	91.09 ± 0.29	91.46 ± 0.61
0.1	1.0	20.0	92.36 ± 0.66	91.44 ± 0.38	91.9 ± 0.66	0.25	1.0	20.0	**92.3 ± 0.19**	91.51 ± 0.17	91.98 ± 0.19
0.1	2.5	15.0	91.99 ± 0.53	91.01 ± 0.29	91.3 ± 0.53	0.25	2.5	15.0	91.65 ± 0.64	90.89 ± 0.31	91.19 ± 0.64
0.1	2.5	17.5	92.17 ± 0.39	91.49 ± 0.17	91.86 ± 0.39	0.25	2.5	17.5	92.04 ± 0.62	91.52 ± 0.27	91.95 ± 0.62
0.1	2.5	20.0	**92.55 ± 0.54**	92.07 ± 0.3	92.49 ± 0.54	0.25	2.5	20.0	92.26 ± 0.33	91.43 ± 0.48	91.97 ± 0.33
0.5	0.1	15.0	90.82 ± 0.28	90.05 ± 0.12	90.45 ± 0.28	0.75	0.1	15.0	90.35 ± 0.44	89.91 ± 0.45	90.42 ± 0.44
0.5	0.1	17.5	90.99 ± 0.44	90.22 ± 0.17	90.87 ± 0.44	0.75	0.1	17.5	90.42 ± 0.38	89.99 ± 0.27	90.48 ± 0.38
0.5	0.1	20.0	**91.78 ± 0.16**	90.85 ± 0.32	91.42 ± 0.16	0.75	0.1	20.0	90.86 ± 0.31	90.31 ± 0.18	90.91 ± 0.31
0.5	1.0	15.0	91.09 ± 0.22	89.92 ± 0.19	90.49 ± 0.22	0.75	1.0	15.0	90.41 ± 0.11	89.22 ± 0.24	89.98 ± 0.11
0.5	1.0	17.5	91.18 ± 0.16	90.39 ± 0.46	90.97 ± 0.16	0.75	1.0	17.5	90.57 ± 0.3	89.49 ± 0.42	90.07 ± 0.3
0.5	1.0	20.0	91.57 ± 0.32	90.88 ± 0.79	91.35 ± 0.32	0.75	1.0	20.0	91.01 ± 0.16	89.9 ± 0.31	90.44 ± 0.16
0.5	2.5	15.0	91.3 ± 0.38	90.45 ± 0.35	90.95 ± 0.38	0.75	2.5	15.0	90.68 ± 0.27	89.5 ± 0.23	90.1 ± 0.27
0.5	2.5	17.5	91.45 ± 0.23	90.71 ± 0.37	91.24 ± 0.23	0.75	2.5	17.5	90.82 ± 0.38	89.76 ± 0.34	90.4 ± 0.38
0.5	2.5	20.0	91.72 ± 0.27	91.04 ± 0.38	91.63 ± 0.27	0.75	2.5	20.0	**91.21 ± 0.2**	90.47 ± 0.51	91.01 ± 0.2

TABLE II: baseline SNN vs. Two-Level Inhibition SNN (60K train / 10K test)

n_e, n_i	Baseline SNN	Two-level (*confidence*)	Two-level (*all*)	Two-level (*distance*)	Two-level (*n-gram*)
100	80.71% ± 1.66%	82.94% ± 1.47%	81.12% ± 1.96%	85.11% ± 0.74%	85.71% ± 0.85%
225	85.25% ± 1.48%	88.49% ± 0.48%	87.33% ± 0.59%	89.11% ± 0.37%	90.50% ± 0.43%
400	88.74% ± 0.38%	91% ± 0.56%	90.56% ± 0.67%	91.4% ± 0.38%	92.56% ± 0.48%
625	91.27% ± 0.29%	92.14% ± 0.50%	91.69% ± 0.59%	92.37% ± 0.29%	**93.39% ± 0.25%**
900	92.63% ± 0.28%	92.36% ± 0.63%	91.73% ± 0.7%	92.77% ± 0.26%	93.25% ± 0.57%

accuracy also points to the fact that network capacity may be too high to fit with a single pass through the data.

D. Network convergence

A major advantage of using a relaxed inhibition scheme is the ability to learn a reasonably good data representation while seeing only a small number of examples. In training LM-SNNs using the *growing* and *two-level* inhibition strategies, we observe a convergence to optimal network performance well before SNNs trained with large, constant inhibition as in [17]. With small inhibition, increasing with the distance between pairs of neurons, the development of filters occurs quickly, allowing the obtainment of respectable accuracy. Over the course of the training, those filters are gradually refined as the inter-neuron inhibition grows and allows the independent firing of neurons.

We show in Figure 6 a comparison of the convergence in estimated test performance for networks of various sizes. SNNs are trained with large, constant inhibition, while LM-SNNs are trained with the *two-level inhibition* strategy with parameter values $p_{\text{low}} = 0.1$, $c_{\text{min}} = 1.0$, and $c_{\text{max}} = 20.0$. Estimates are calculated by assigning labels to neurons based on the firing activity on 250 training examples, and using these labels to classify the next 250 training examples. Training accuracy curves are *smoothed* by averaging each estimate with a window of the 10 neighboring estimates. The performance of LM-SNNs quickly outpace that of SNNs, and obtain near-optimal accuracy after seeing 10 to 20 thousand training examples. Due to the inhibition strategy, larger LM-SNNs

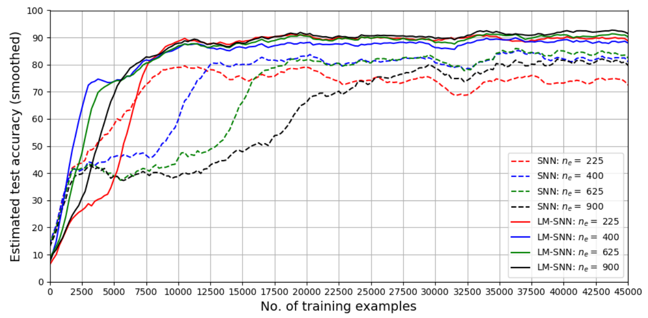

Fig. 6: LM-SNN vs. SNN smoothed performance estimate over the training phase.

achieve better accuracy more quickly, due to their ability to learn more filters at once. On the other hand, SNNs are limited to learning one filter at a time, and their accuracy during the training phase is hindered by the size of the network as a result.

E. Sparse input connectivity

Instead of connecting the input one-to-one with the layer of excitatory neurons, we experiment with varying degrees of random sparse connectivity. We are interested in whether small amounts of sparsity might make our network more robust to outliers in the MNIST data, therefore increasing the chance of good test performance. We also hope that our system will still perform well in the event of missing features, degrading in performance gracefully as the input data becomes less clear.

TABLE III: Large networks: baseline SNN vs. Two-Level Inhibition SNN

n_e, n_i	n_{train}	baseline SNN	Two-level (*confidence*)	Two-level (*all*)	Two-level (*distance*)	Two-level (*n-gram*)
1,225	1 × 60K	91.51% ± 0.44%	91.38% ± 0.89%	90.93% ± 0.88%	92.73% ± 0.36%	92.39% ± 0.47%
1,225	2 × 60K	92.37% ± 0.22%	92.25% ± 0.63%	91.77% ± 0.59%	92.39% ± 0.29%	93.69% ± 0.32%
1,225	3 × 60K	92.43% ± 0.23%	92.57% ± 0.57%	91.85% ± 0.52%	92.48% ± 0.29%	93.87% ± 0.25%
1,600	1 × 60K	90.18% ± 0.58%	89.59% ± 0.98%	89.26% ± 0.94%	92.45% ± 0.33%	92.42% ± 0.62%
1,600	2 × 60K	92.54% ± 0.51%	92.61% ± 0.51%	91.79% ± 0.56%	92.66% ± 0.26%	93.54% ± 0.5%
1,600	3 × 60K	92.80% ± 0.49%	92.96% ± 0.56%	92.87% ± 0.49%	93.03% ± 0.30%	**94.07% ± 0.46%**

Interestingly, small amounts of sparsity do not degrade network performance much, and with nearly all connections removed, the network maintains reasonable accuracy. In particular, with 90% of synapses from the input removed, networks of 625 excitatory and inhibitory neurons achieve nearly 60% test accuracy after being trained using the *two-level inhibition* mechanism. Although this technique did not result in improved test performance, it demonstrates the robustness of our system to missing information in the input data.

V. Conclusions and Future Work

We have demonstrated a learning scheme with spiking neural networks with self-organization, classification, and clustering properties in an unsupervised fashion. We have found that, in using our training algorithm, the LM-SNN tends to cluster the categories of the inputs into groups of similar filter representations. Moreover, the two-level inhibition scheme tends to create filter maps which vary smoothly between input classes, but our classification scheme does not yet exploit this clustering to improve test accuracy. Nevertheless, we discover that tuning the c_{inhib} parameter dynamically during training allows control over the smoothness of the learned filter map. We also demonstrated the importance of spike ordering for classification by using the *n-gram* scheme. The development of neuron labeling and subsequent classification strategies will continue to be an important part of evaluating SNN learned representations.

Similar to SOM behavior, when an input is presented to the excitatory layer, neurons *compete* to fire. When a neuron fires, it inhibits all other neurons beside its neighbors (the level and pattern of which depends on the choice of inhibition strategy), which encourages neighboring neurons near their threshold to spike. As a result of these spikes, filter weights get closer to the current input and groups of similar filters are formed spontaneously (see Figure 2, compared to Figure 5). By relaxing the scheme of large constant inhibition, we allow multiple neurons to fire at once, enabling faster learning and sparse network activation.

Due to the robustness of the LM-SNN in the event of missing inputs, networks can be trained and evaluated on unreliable hardware components. That is, we observe a *graceful degradation* in performance as we remove synapses learned using STDP. The relative infrequency of spikes in both the input and excitatory layer means that the number of STDP updates is low enough to train large LM-SNNs. This is contrast to deep learning neural networks, in which all network parameters are modified synchronously at each back-propagation pass. The asynchronous nature of our system enables this computational advantage.

In this paper we demonstrate a change in SNN training that is accomplished without back-propagation and without labeled data, comprising a system which is suited to rapid decision-making. Once the learning procedure has converged, one may choose to use an auxiliary labeled dataset and network activity together to determine *what* the excitatory neurons represent. Or, one can instead cluster or organize the learned representation as a means of visualizing or summarizing a dataset. Our system represents a step forward in unsupervised learning with spiking neural networks, and suggests additional, biologically inspired research directions. Additional inhibition schemes may lead to a more biologically plausible and useful distribution of network activity.

Acknowledgements

This work has been supported in part by Defense Advanced Research Project Agency Grant, DARPA/MTO HR0011-16-l-0006 and by National Science Foundation Grant NSF-CRCNS-DMS-13-11165.

References

[1] P. J. Werbos, "Beyond regression: New tools for prediction and analysis in the behavioral sciences." *Doctoral Dissertation, Applied Mathematics, Harvard University, MA.*, 1974.

[2] Rumelhart, D. E., Hinton, G. E., and Williams, R. J., "Learning representations by error propagation." *In D. E. Rumelhart, J. L. McClelland, and the PDP Research Group (Eds.), Parallel distributed processing*, vol. 1, pp. 318–362, 1986.

[3] Y. LeCun, Y. Bengio, G. Hinton, "Deep learning," *Nature*, no. 521, pp. 436–444, May 2015.

[4] Bi, G. and Poo, M., "Synaptic Modifications in Cultured Hippocampal Neurons: Dependence on Spike Timing, Synaptic Strength, and Postsynaptic Cell Type," *Journal of Neuroscience*, vol. 18, no. 24, pp. 10 464–10 472, 1998. [Online]. Available: http://www.jneurosci.org/content/18/24/10464

[5] W. Gerstner, W. M. Kistler, *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[6] M.-O. Gewaltig and M. Diesmann, "Nest (neural simulation tool)," *Scholarpedia*, vol. 2, no. 4, p. 1430, 2007.

[7] D. F. M. Goodman, R. Brette, "The Brian simulator," *Frontiers in Computational Neuroscience*, Sep. 2009.

[8] M. Beyeler, K. D. Carlson, Ting-Shuo Chou, N. Dutt and J. L. Krichmar, "Carlsim 3: A user-friendly and highly optimized library for the creation of neurobiologically detailed spiking neural networks," in *2015 International Joint Conference on Neural Networks (IJCNN)*, July 2015, pp. 1–8.

[9] Diehl, P. U. and Neil, D. and Binas, J. and Cook, M. and Liu, S. C. and Pfeiffer, M., "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2015, pp. 1–8.

[10] Liu, D. and Yue, S., “Fast unsupervised learning for visual pattern recognition using spike timing dependent plasticity,” *Neurocomputing*, vol. 249, pp. 212–224, Aug. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231217306276

[11] Kheradpisheh, S. R. and Ganjtabesh, M. and Thorpe, S. J. and Masquelier, T., “STDP-based spiking deep convolutional neural networks for object recognition,” *arXiv:1611.01421 [cs]*, Nov. 2016, arXiv: 1611.01421. [Online]. Available: http://arxiv.org/abs/1611.01421

[12] Saunders, D. J., Siegelmann, H. T., Kozma, R., Ruszinkó, M., “Stdp learning of image features with spiking neural networks,” *IEEE/INNS IJCNN2018, July 15-19, 2018, Rio de Janeiro, Brazil (submitted).*

[13] Kohonen, T., *Self-Organizing Maps*, 3rd ed., ser. Springer Series in Information Sciences. Springer, 2001, vol. 30.

[14] E. Erwin, K. Obermayer, and K. Schulten, “Models of orientation and ocular dominance columns in the visual cortex: A critical comparison,” *Neural Computation*, vol. 7, no. 3, pp. 425–468, 1995. [Online]. Available: https://doi.org/10.1162/neco.1995.7.3.425

[15] Y. LeCun, L. Bottou, Y. Bengio, and P., Haffner, “Gradient-based learning applied to document recognition.” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[16] W. M. K. W. Gerstner, *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[17] P. U. Diehl, M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Frontiers in Computational Neuroscience*, Aug. 2015.

[18] Gollisch, T. and Meister, M., “Rapid neural coding in the retina with relative spike latencies,” *Science*, vol. 319, no. 5866, pp. 1108–1111, 2008. [Online]. Available: http://science.sciencemag.org/content/319/5866/1108

[19] R. S. Petersen, S. Panzeri and M. E. Diamond, “Population coding of stimulus location in rat somatosensory cortex,” *Neuron*, vol. 32, no. 3, pp. 503 – 514, 2001. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0896627301004810

[20] Raichman, N. and Ben-Jacob, E., “Identifying repeating motifs in the activation of synchronized bursts in cultured neuronal networks,” *Journal of Neuroscience Methods*, vol. 170, no. 1, pp. 96 – 110, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0165027008000046

[21] Kermany, E., Gal, A., Lyakhov, V., Meir, R., Marom, S., Eytan, D., “Tradeoffs and constraints on neural representation in networks of cortical neurons,” *Journal of Neuroscience*, vol. 30, no. 28, pp. 9588–9596, 2010. [Online]. Available: http://www.jneurosci.org/content/30/28/9588

[22] D. E. Rumelhart, R. Durbin, R. Golden, and Y. Chauvin, “Backpropagation,” in *Backpropagation*, Chauvin, Y. and Rumelhart, D. E., Ed. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1995, ch. Backpropagation: The Basic Theory, pp. 1–34. [Online]. Available: http://dl.acm.org/citation.cfm?id=201784.201785