DARPA Quarterly Report - Biologically inspired efficient learning algorithms

Daniel Saunders Pegah Taheri Hananel Hazan

December 15, 2017

1 Introduction

It has been recently established that deep learning (DL) is the preferred approach to machine learning problems in domains in which large amounts of data, computational power, and time are available [1]. However, in the presence of little data or time, this approach may fail. Motivated by these shortcomings, and inspired by brain computation, we are developing unsupervised learning algorithms which learn *robust representations of data* with few training examples, and whose training has the potential to be *massively parallelized* to scale with the dimensionality of the input data and number of computational nodes. Building on the spiking neural network (SNN) model introduced in [2], we make several architectural modifications in hopes to improve the learning procedure and consequent test dataset error rate. Our model is tested on the MNIST dataset [4], and we show how our modifications change error rates and modify the behavior of the SNN network models.

Realizing that the **brian** spiking neural networks simulation is ill-equipped to handle large networks needed to learn mappings from high-dimensional data, we are converting our SNN models to other programming frameworks. We have successfully implemented a simple SNN in the **PyTorch** (*http://pytorch.org/*) neural networks package, and are working towards **TensorFlow** (*https://www.tensorflow.org/*) and **NEST** (*http://www.nest-simulator.org/*) implementations as well. We show some learned filters and demonstrate the scaling capability of our SNNs in **PyTorch**.

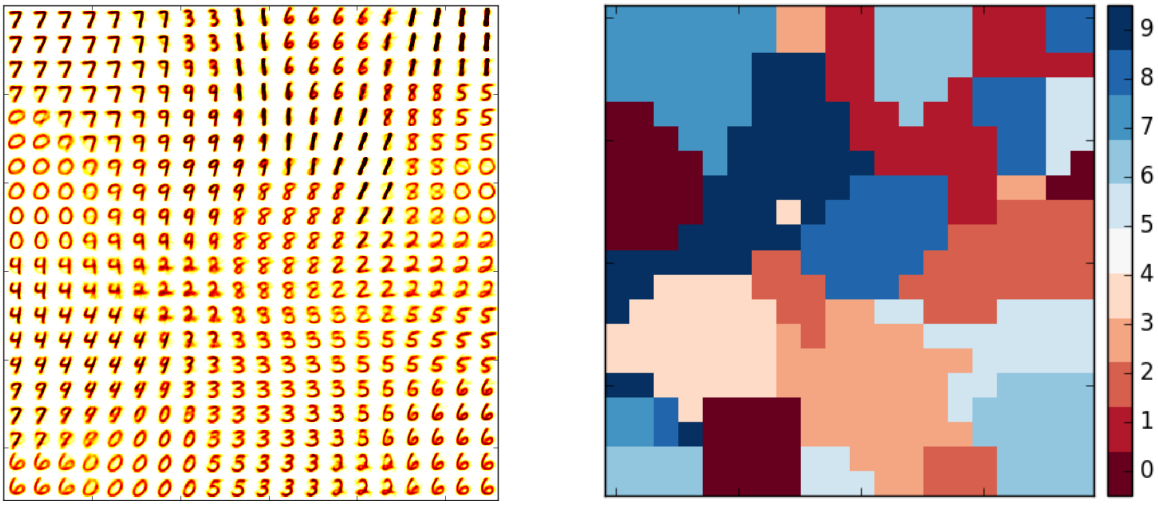
All networks are trained on one pass through the MNIST training data (60K) examples.

2 Methods

Previously, we modified the inhibition mechanism used in [2] such that neurons, arranged in a 2D lattice, are inhibited more if they are further away from a spiking neuron. As a result, with certain settings of the inhibition constant c_{inhib}, the learned filter map smoothly varied between different digit classes (c_{inhib} small) or arranged themselves into tight, similar clusters (c_{inhib} large). In both cases, filters were clustered spatially, giving a useful visualization of the training data. An example (with $c_{\text{inhib}} = 0.5$) is shown in Figures 1a and 1b. Although visually appealing, we found that this scheme only reduced the accuracy from our SNN baseline [2]. When $c_{\text{inhib}} = c_{\max} = 17.5$ (the inhibition level of [2]), the network achieved maximum classification accuracy, developing individualized, rather than clustered, filters.

2.1 Growing the inhibition level over the training phase

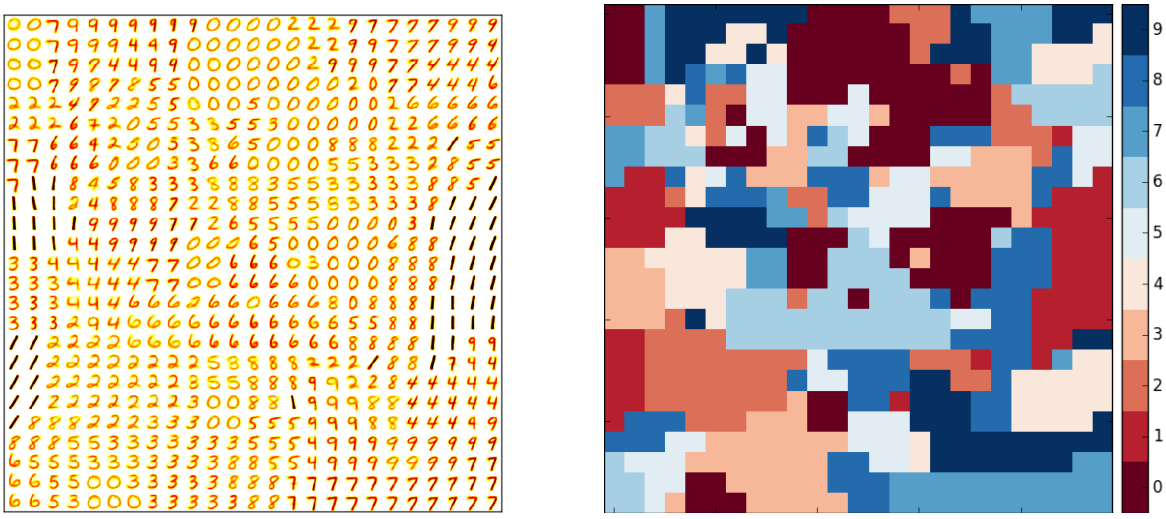
We wanted to produce individualized filters as learned by the SNN presented in [2], yet retain the clustering of filters achieved by our *increasing inhibition* modification. To that end, we implemented another modification to the inhibition scheme, in which the inhibition constant c_{inhib} grows on a linear schedule from a small $c_{\min} \approx 0.5$ to a large $c_{\max} \approx 17.5$. The increasing inhibition is used as before; however, by the end of the training, the inhibition is equivalent to that of [2]. In this setting, the filters self-organize into smoothly-varying clusters, and then individualize as the inhibition level becomes large. We also considered growing the inhibition level to $c_{\max}$ for some percentage of the training (p_{grow}) and holding it fixed for the rest ($1 - p_{\text{grow}}$). Shown in Figures 2a and 2b are example filters and corresponding class assignments



(a) 20x20 lattice of neuron filters

(b) Corresponding neuron class assignments

Figure 1: Inhibition increasing with distance - filter map and class assignments



(a) 25x25 lattice of neuron filters

(b) Corresponding neuron class assignments

Figure 2: Growing inhibition over training phase - filter map and class assignments

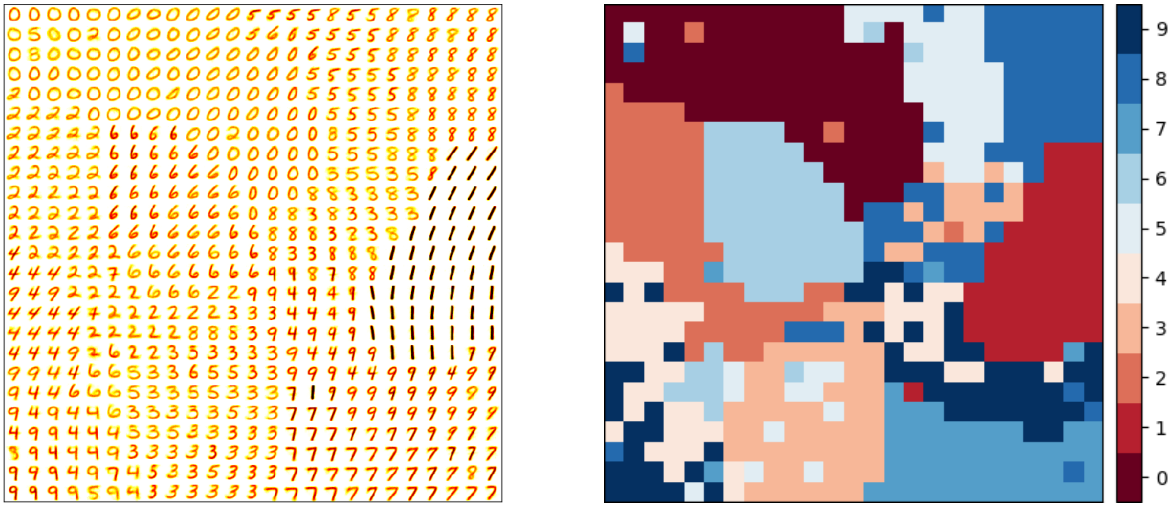
learned by this scheme, with $c_{\min} = 0.1, c_{\max} = 17.5, p_{\text{grow}} = 0.25$. The filters shown have clearly organized into clusters based on similarity (with a few exceptions), yet have largely individualized so as to avoid a redundant data representation. Test accuracy results for networks of size 400, 625, and 900 are shown in Table 1.

2.2 Two-level inhibition

Going one step further, in order to reduce the computational requirement of re-computing inhibitory synapse weights continually throughout network training, we implemented a simple two-*level inhibition* scheme: For the first p_{grow} proportion of the training, the network is trained inhibition level $c_{\min}$; for the last $1 - p_{\text{grow}}$ 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_{grow} mark. Example learned weights and corresponding class assignments are shown in Figures 3a and 3b. Clearly, the filter clustering is somewhat more fragmented, but subjectively, filter quality and diversity seem to have been maintained. Test accuracy results are given in

Table 1: Growing inhibition test accuracy

n_e, n_i	p_{grow}	Test accuracy
400	25%	**91.48**%
400	50%	91.32%
400	75%	89.63%
400	100%	89.83%
625	25%	**91.71**%
625	50%	91.41%
625	75%	91.51%
625	100%	90.63%
900	25%	(seizure)
900	50%	**93.06**%
900	75%	92.95%
900	100%	92.53%



(a) 25x25 lattice of neuron filters

(b) Corresponding neuron class assignments

Figure 3: Growing inhibition over training phase - filter map and class assignments

Table 2. All networks are comprised of 625 excitatory and inhibitory neurons.

2.3 Comparing baseline SNN and two-level inhibition

To get a feeling for how our best system to date compares with the baseline SNN of [2], we present in Figure 3 a comparison of their accuracies across several settings of excitatory, inhibitory neurons, including the *confidence*, *all*, and *distance* voting schemes. All networks are trained for 60K iterations (a single pass through the training data) and evaluated on all 10K examples from the test data. 10 independent experiments with different initial configurations and input Poisson spike trains were run, and their results are averaged and reported along with a single standard deviation. Note that some experiments failed to finish due to numerical instability or out-of-memory errors. In this case, the number reported in parentheses next to the accuracies in each cell corresponds to the number of successful experiments.

One may notice a downward trend in the results after $n_e, n_i = 1,225$; we believe that these networks are *underfitting* to the data, and their test performance should improve given more passes over the training data. Consult Figure 4 for example filter maps learned by the two-level inhibition scheme while training on only one pass through the MNIST training data. Compare with Figure 2a.

Notice also the clear superiority of the *confidence* classification scheme to the *all* classification scheme,

Table 2: Two-level inhibition test accuracy ($n_e, n_i = 625$)

p_{grow}	c_{low}	c_{high}	Test%	p_{grow}	c_{low}	c_{high}	Test%	p_{grow}	c_{low}	c_{high}	Test%
25%	0.1	15.0	92.21%	50%	0.1	15.0	90.91%	75%	0.1	15.0	89.88%
25%	0.1	17.5	91.18%	50%	0.1	17.5	91.37%	75%	0.1	17.5	91.05%
25%	0.1	20.0	91.87%	50%	0.1	20.0	90.9%	75%	0.1	20.0	90.41%
25%	1.0	15.0	90.84%	50%	1.0	15.0	91.74%	75%	1.0	15.0	89.31%
25%	1.0	17.5	91.44%	50%	1.0	17.5	90.64%	75%	1.0	17.5	89.8%
25%	1.0	20.0	92.71%	50%	1.0	20.0	91.32%	75%	1.0	20.0	90.12%
25%	2.5	15.0	91.26%	50%	2.5	15.0	90.96%	75%	2.5	15.0	90.25%
25%	2.5	17.5	91.04%	50%	2.5	17.5	91.24%	75%	2.5	17.5	90.18%
25%	2.5	20.0	**92.78**%	50%	2.5	20.0	91.55%	75%	2.5	20.0	90.44%

and the *distance* scheme to the *confidence* scheme. Whereas both the *confidence* and *all* 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.

Table 3: baseline SNN vs. Two-Level Inhibition SNN (60K train / 10K test)

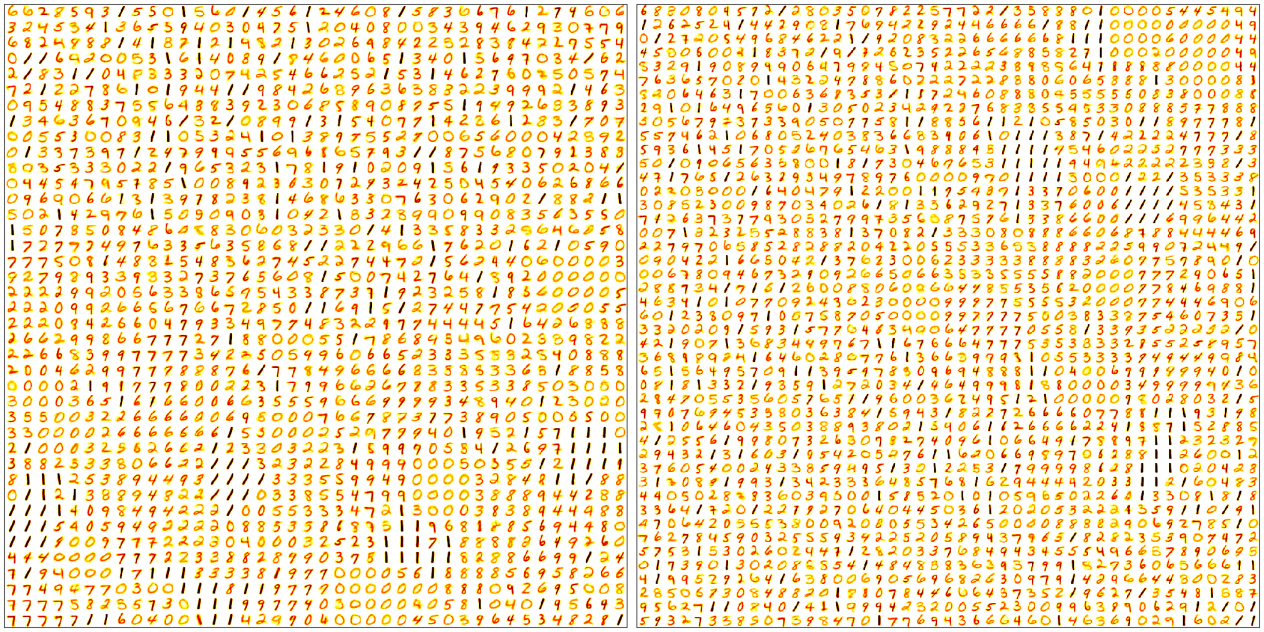
n_e, n_i	ETH	Two-level (*confidence*)	Two-level (*all*)	Two-level (*distance*)
100	80.71% ± 1.66% (10)	82.94% ± 1.47% (10)	81.12% ± 1.96% (10)	85.11% ± 0.74% (10)
225	85.25% ± 1.48% (10)	88.49% ± 0.48% (10)	87.33% ± 0.59% (10)	89.11% ± 0.37% (10)
400	88.74% ± 0.38% (10)	91% ± 0.56% (10)	90.56% ± 0.67% (10)	91.4% ± 0.38% (10)
625	91.27% ± 0.29% (10)	92.14% ± 0.50% (10)	91.69% ± 0.59% (10)	92.37% ± 0.29% (10)
900	92.63% ± 0.28% (10)	92.36% ± 0.63% (10)	91.73% ± 0.7% (10)	92.77% ± 0.26% (10)
1,225	93.20% ± 0.65% (7)	91.38% ± 0.89% (10)	90.93% ± 0.88% (10)	92.73% ± 0.36% (10)
1,600	91.99% ± 0.32% (4)	89.59% ± 0.98% (10)	89.26% ± 0.94% (10)	92.45% ± 0.33% (10)
2,025	(crashed)	88.33% ± 0.63% (6)	88.05% ± 0.54% (6)	91.96% ± 0.33% (6)

2.4 Sparse input

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. We use sparsity levels of 10%, 25%, 50%, 75%, and 90%, and include the fully-connected case in order to compare. We using the *growing inhibition* scheme as discussed in section 2.1. The results are shown in Table 4, in which the results from 10 independent training and test phases are averaged and reported along with their standard deviations. Interestingly, small amounts of sparsity do not degrade network performance much at all, and even with nearly all connections removed (90%), the network maintains $\tilde{5}8\%$ accuracy. Unfortunately, this did not produce an improvement in network classification performance. See Figure 5 for a visualization of learned filters in a network with 90% sparsity.

3 PyTorch Implementation

The **brian** (*http://briansimulator.org/*) spiking neural networks package used for our experiments was not designed to simulate large-scale networks, nor was it meant to solve machine learning problems. Although



(a) $n_e, n_i = 1{,}600$ (b) $n_e, n_i = 2{,}025$

Figure 4: Large SNNs trained with *two-level inhibition* scheme

Table 4: Sparse input test accuracy

n_e, n_i	% sparsity	Test accuracy
625	0%	**91.71**% ± 0.23%
625	10%	91.48% ± 0.31%
625	25%	89.79% ± 0.66%
625	50%	85.83% ± 0.95%
625	75%	75.71% ± 1.20%
625	90%	58.60% ± 1.31%

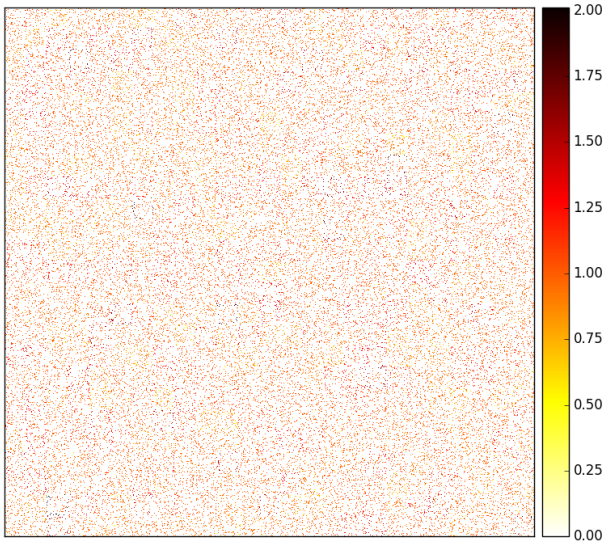
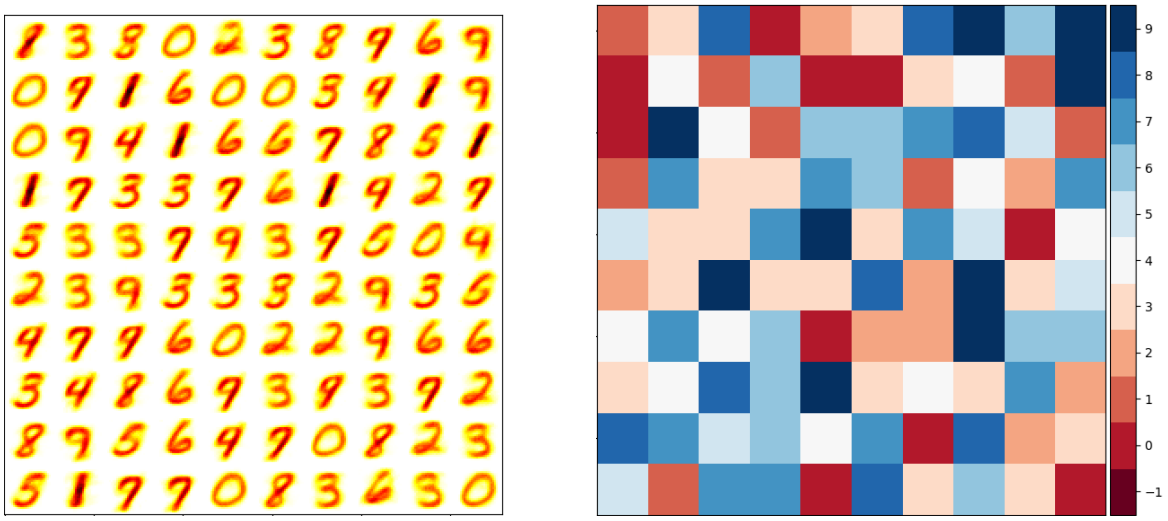


Figure 5: Network weights - 90% sparsity



(a) 25x25 lattice of neuron filters

(b) Corresponding neuron class assignments

Figure 6: **PyTorch** 100-neuron SNN - learned filters and class assignments

it was successfully used to solve the MNIST digit classification task, we expect that the software would not be able to handle larger networks used to classify more complex image datasets; e.g., CIFAR-10 or ImageNet. We re-implemented the SNN from [2] in **PyTorch** (*http://pytorch.org/*), performing calculations with the strongly GPU-accelerated **torch.Tensor**. We use the simplified leaky integrate-and-fire (LIF) neuron, and tweak network hyper-parameters to produce learning behavior similar to that reported in [2].

3.1 Learned filters

We want our networks to learn filters similar to those learned the SNNs from [2]. Using the simpler neuron model requires that our simulation hyper-parameters (e.g., time constants, maximal weight values, etc.) be re-tuned to produce similar learning behavior. We show example filters and corresponding class assignments for networks of 100 and 400 excitatory and inhibitory neurons in Figures 6, 7, and 8. On inspection, the filters resemble the digit class with which they are labeled; however, filter quality and individuality appear to be somewhat reduced from those of the baseline SNN [2]. Some of the filters in the 900-neuron network have yet to settle on a representation of a digit: we believe this is again due to *underfitting*, and should be remedied with additional passes through the training data.

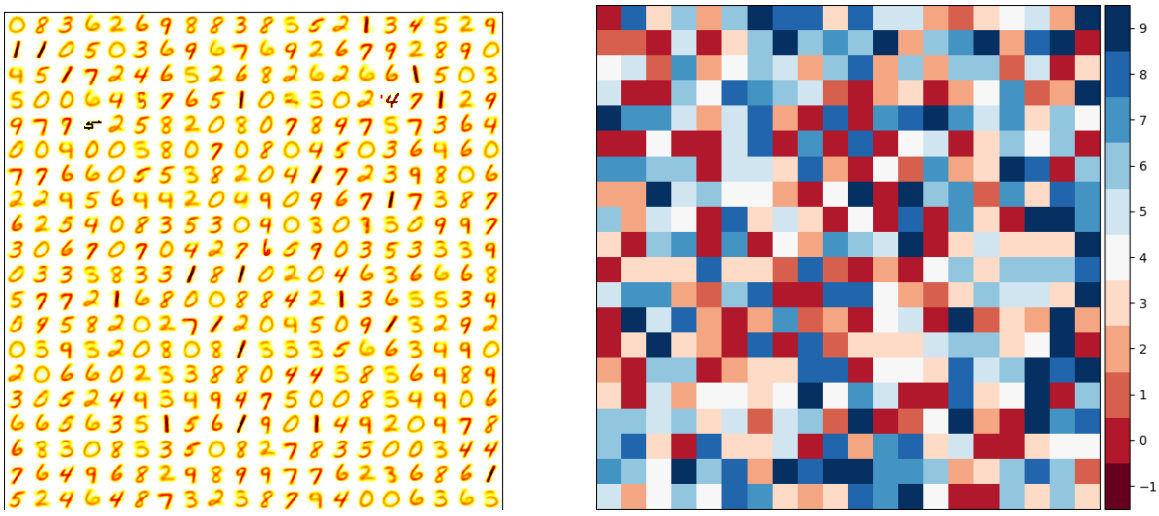
3.2 Performance comparison

While we want to use a more powerful and flexible programming framework, we must also keep in mind our goal of creating accurate unsupervised machine learning algorithms implemented by spiking neural networks. Due to lack of time, we were unable to perform large-scale experimentation, in order to find the best setting of network hyper-parameters and to provide a result average over 10 independent trials. For now, we report in Table 5 the result of training a 100-, 400-, and 900-neuron SNN training on one run over all 60K training iterations and evaluated on the entire 10K test dataset, alongside accuracy results from the baseline SNN [2].

References

[1] Lecun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning." *Nature* 521.7553 (2015): 436-44.

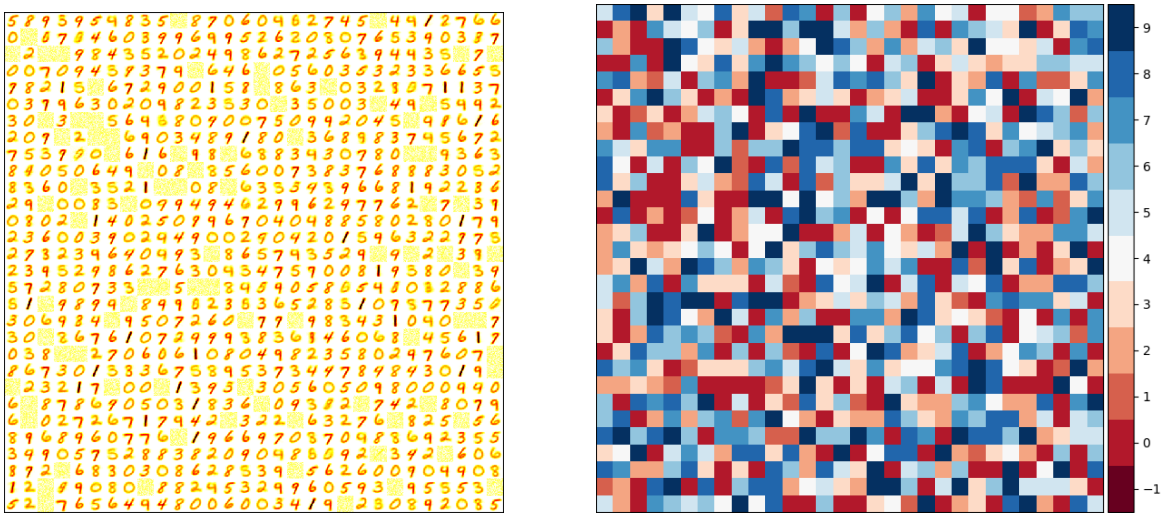
[2] Diehl, Peter U., and Matthew Cook. "Unsupervised Learning of Digit Recognition Using Spike-timing-dependent Plasticity." Frontiers in Computational Neuroscience (2015)



(a) 25x25 lattice of neuron filters

(b) Corresponding neuron class assignments

Figure 7: **PyTorch** 400-neuron SNN - learned filters and class assignments



(a) 25x25 lattice of neuron filters

(b) Corresponding neuron class assignments

Figure 8: **PyTorch** 900-neuron SNN - learned filters and class assignments

Table 5: **PyTorch** vs. baseline SNN

n_e, n_i	**PyTorch** SNN	baseline SNN
100	78.31%	80.71% ± 1.66%
400	82.80%	88.74% ± 0.38%
900	84.16%	92.63% ± 0.28%

[3] Markram, H., W. Gerstner. "Spike-Timing-Dependent Plasticity: A Comprehensive Overview." Frontiers in Synaptic Neuroscience (2012)

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998.