Neural Networks 119 (2019) 332–340

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Locally connected spiking neural networks for unsupervised feature learning

Daniel J. Saunders[a,*], Devdhar Patel[a], Hananel Hazan[a], Hava T. Siegelmann[a], Robert Kozma[a,b,**]

[a] *Biologically Inspired Neural and Dynamical Systems Laboratory (BINDS), College of Information and Computer Sciences, University of Massachusetts Amherst, 140 Governors Drive, Amherst, MA 01003, USA*
[b] *Center for Large-Scale Intelligent Optimization and Networks (CLION), Department of Mathematics, University of Memphis, Memphis, TN 38152, USA*

ARTICLE INFO

Article history:
Available online 26 August 2019

Keywords:
Spiking neural network
Machine learning
Spike-timing-dependent plasticity
BindsNET
Unsupervised learning
Image classification

ABSTRACT

In recent years, spiking neural networks (SNNs) have demonstrated great success in completing various machine learning tasks. We introduce a method for learning image features with *locally connected layers* in SNNs using a spike-timing-dependent plasticity (STDP) rule. In our approach, sub-networks compete via inhibitory interactions to learn features from different locations of the input space. These *locally-connected SNNs* (LC-SNNs) manifest key topological features of the spatial interaction of biological neurons. We explore a biologically inspired *n*-gram classification approach allowing parallel processing over various patches of the image space. We report the classification accuracy of simple two-layer LC-SNNs on two image datasets, which respectively match state-of-art performance and are the first results to date. LC-SNNs have the advantage of fast convergence to a dataset representation, and they require fewer learnable parameters than other SNN approaches with unsupervised learning. Robustness tests demonstrate that LC-SNNs exhibit graceful degradation of performance despite the random deletion of large numbers of synapses and neurons. Our results have been obtained using the `BindsNET` library, which allows efficient machine learning implementations of spiking neural networks.

1. Introduction

Processing of image data by machine learning (ML) systems often requires techniques for overcoming the problem known as the *curse of dimensionality*; that is, the inherent difficulty in processing high-dimensional data. The locally-connected layer of the neural networks literature (Chen et al., 2015) imposes an inductive bias on an artificial neural network (ANN) that stipulates that certain features only occur in particular regions of input space. This is in sharp contrast to the popular *convolutional layer*, in which a set of filters are learned that can be used to detect features across the entire input space, inducing a *translation invariant* representation (Kauderer-Abrams, 2016) in which the occurrence and relative locations of features are important. In the deep learning (DL) literature, both methods were key to reducing the number of parameters needed to train deep and efficient networks for processing complex data; e.g., images and videos (Goodfellow, Bengio, & Courville, 2016; LeCun, Bengio, & Hinton, 2015).

Spiking neural networks (Maass, 1997) are well-known for their energy efficiency and speed on neuromorphic hardware platforms (Zambrano, Nusselder, Scholte, & Bohte, 2017). For this reason, development of methods that enable SNNs to reach the performance of ANNs on machine learning tasks has gained interest in recent years (Tavanaei, Ghodrati, Kheradpisheh, Masquelier, & Maida, 2018). Previous work has shown that spiking neural networks can, in principle, be used for performing complex machine learning tasks; e.g., image classification (Diehl et al., 2015; Sengupta, Ye, Wang, Liu, & Roy, 2018) and reinforcement learning (Florian, 2007; Mozafari, Kheradpisheh, Masquelier, Nowzari-Dalini, & Ganjtabesh, 2018). On the other hand, few methods exist for the robust training of SNNs from scratch for general-purpose machine learning ability; i.e., their abilities are highly domain- or dataset-specific, and require much data preprocessing and tweaking of hyper-parameters in order to attain good performance.

* Corresponding author.
** Corresponding author at: Biologically Inspired Neural and Dynamical Systems Laboratory (BINDS), College of Information and Computer Sciences, University of Massachusetts Amherst, 140 Governors Drive, Amherst, MA 01003, USA.
E-mail addresses: djsaunde@cs.umass.edu (D.J. Saunders), devdharpatel@cs.umass.edu (D. Patel), hhazan@cs.umass.edu (H. Hazan), hava@cs.umass.edu (H.T. Siegelmann), rkozma@cs.umass.edu (R. Kozma).

https://doi.org/10.1016/j.neunet.2019.08.016
0893-6080/© 2019 Elsevier Ltd. All rights reserved.

This paper aims to extend the abilities of spiking neural networks by introducing and studying a spiking version of a computational layer borrowed from the artificial neural networks literature. In particular, the *locally connected layer* is introduced and implemented in spiking neural networks, to which any biological learning rule (e.g., Hebbian learning, spike-timing-dependent plasticity, etc.) may be applied (Zappacosta, Mannella, Mirolli, & Baldassarre, 2018). Simple two-layer networks are built and trained in an unsupervised fashion to produce representations of the MNIST (LeCun & Cortes, 2010) and EMNIST (Cohen, Afshar, Tapson, & van Schaik, 2017) datasets. Ultimately, the goodness of the representation (i.e., to what extent the learned features and network activity separate the categories of data) is assessed through a simple *voting mechanism* on the spiking outputs of the trained networks. In particular, we use the *n*-gram method of Hazan, Saunders, Sanghavi, Siegelmann, and Kozma (2018), and a number of simpler methods as points of comparison. The introduced spiking network layer and accompanying training procedure produce distributed representations which are robust to a large degree of randomly deleted synapses and neurons, a property commonly referred to as *graceful degradation*.

There are various libraries that allow efficient spiking neural network simulation. Some of them provide fine granularity of biological details (Carnevale & Hines, 2006; Cornelis, Rodriguez, Coop, & Bower, 2012; Dan Goodman & Brette, 2009; Gewaltig & Diesmann, 2007; Vitay, Dinkelbach, & Hamker, 2015), while others focus on the high-level function of SNNs, which are better suited to machine learning experimentation (Bekolay et al., 2014; Kasabov, 2014). The present work builds on the results of Saunders, Siegelmann, Kozma, and Ruszinko (2018) and (Diehl & Cook, 2015), wherein all networks were implemented in the BRIAN spiking neural networks simulator (Goodman & Brette, 2009). To support flexible spiking neural networks implementation, we developed BindsNET, a library written in Python specifically for machine learning experimentation with SNNs (Hazan et al., 2018). The SNNs introduced in this paper are implemented in BindsNET, which has enabled faster simulation run-time and networks with many more neurons, while maintaining and even improving upon quantitative results in image classification.

2. Related work

Due to their potential for efficient implementation on neuromorphic hardware devices, much recent work has focused on developing powerful and efficient learning algorithms for SNNs. We focus on SNNs trained for machine learning tasks in simulated environments, which may be either (1) converted to neuromorphic platforms post-training, or, in some cases (2) trained and evaluated entirely on neuromorphic chips. A review of machine learning with SNNs is treated by Tavanaei et al. (2018).

This work is related most closely to that of Diehl & Cook (Diehl & Cook, 2015), in which a simple three-layer network is trained unsupervised with spike-timing-dependent plasticity along with excitatory–inhibitory interactions between neurons to learn to classify the MNIST handwritten digits (LeCun & Cortes, 2010). The input layer is comprised of Poisson spiking neurons fully-connected with STDP-modifiable synapses to an arbitrarily-sized layer of excitatory neurons. Neurons in this layer "compete" with one another to learn filters that capture prototypical examples from the data via connections from an inhibitory layer. Once trained, this network operates similarly to a *winner-take-all* (WTA) circuit, with sparse activity in the excitatory population coding for a single category of data. Several extensions of this work have been explored quite recently, including the simultaneous clustering and classification of image data (Hazan et al., 2018) and the incremental learning of distinct categories (Allred & Roy, 2016).

Several other unsupervised learning methods have been developed and demonstrated with SNNs. A network inspired by the autoencoder of the neural networks literature is trained layer-wise, without labels, to reconstruct the MNIST and CIFAR-10 datasets, and whose output is trained in a supervised fashion to perform classification (Panda & Roy, 2016). A learning algorithm for SNNs was developed on the hypothesis that biological neurons tend to maximize their activity in competition with other neurons (Demin & Nekhaev, 2018). Three-layer networks of Izhikevich regular spiking neurons were trained with STDP on the binary task of distinguishing the digits '0' and '1', as well as the full set of MNIST digits (Tavanaei & Maida, 2015).

In this paper, we are primarily interested in the implementation of the locally-connected layer (see, e.g., Chen et al., 2015) for spiking neural networks. A closely related concept is the convolutional layer (Goodfellow et al., 2016), which is named for how it convolves a number of weight kernels across the input space while its weights remain fixed, or *shared* between locations in the input space. A number of studies have implemented convolutional layers for SNNs, with a mixture of supervised and unsupervised learning methods used to learn their weights (Kheradpisheh, Ganjtabesh, Thorpe, & Masquelier, 2018; Lee, Panda, Srinivasan, & Roy, 2018; Tavanaei, Kirby, & Maida, 2018; Tavanaei & Maida, 2016, 2017). The locally connected layer is similar to this concept, except that a different set of weight kernels is learned per input location; i.e., the weights are not shared as it strides across the input space.

3. Methods

3.1. Spiking neuron model

Our networks use a variant of the popular *leaky integrate-and-fire* (LIF) spiking neuron with adaptive firing thresholds. The dynamics of the LIF voltage $v(t)$ is given by

$$\tau_v \frac{\mathrm{d}v(t)}{\mathrm{d}t} = -(v(t) - v_{\text{rest}}) + I(t), \tag{1}$$

where τ_v is the membrane time constant, v_{rest} is the neuron's *resting voltage* (to which it exponentially decays), and $I(t)$ denotes the total input to the neuron at time t. The voltage (membrane potential) $v(t)$ of a *post-synaptic* neuron is increased at the occurrence of a spike from a *pre-synaptic* neuron connected to it by the weight w of the synapse that connects them; all such incoming currents are gathered in $I(t)$. When a neuron reaches or exceeds its threshold membrane potential $v_{\text{thresh}}(t)$, it emits a spike to downstream neurons and resets to a voltage v_{reset}. At this point, the neuron is clamped to the reset voltage for a short *refractory period* Δ_{ref} and does not integrate its inputs.

The adaptive threshold $\theta(t)$ is used to ensure that a single neuron does not end up dominating the firing activity of the output layer, and that different neurons tune selectively to the intensity of different prototypical inputs. It has dynamics

$$\tau_\theta \frac{\mathrm{d}\theta(t)}{\mathrm{d}t} = -\theta(t), \tag{2}$$

where $v_{\text{thresh}}(t) = \theta_0 + \theta(t)$, with $\theta_0 > v_{\text{reset}}, v_{\text{rest}}$ a constant. Each time a neuron spikes, the adaptive threshold θ is increased by a constant θ_{plus}.

In our experiments, we set $\theta_0 = -52$mV, $v_{\text{rest}} = v_{\text{reset}} = -65$mV, $\Delta_{\text{ref}} = 5$ ms, $\tau_v = 20$ ms, $\tau_\theta = 1000$s, and $\theta_{\text{plus}} = 0.05$. In the context of machine learning, it may be of interest to include these in a grid- or random-search hyper-parameter optimization algorithm, but for the experiments described in this paper, they remain fixed.

Input neurons (used to convert non-negative image data to spikes in our experiments) are modeled by *Poisson spikes trains* with average firing rate equal to the input intensity in Hertz. For the experiments in Section 4, we multiply the MNIST and EMNIST images (with grayscale values in [0, 255]) by a factor of $\frac{1}{2}$ to constrain input firing rates in [0 Hz, 127.5 Hz].

3.2. Learning rule

Our locally-connected SNN layers are trained with a simple spike-timing-dependent plasticity (STDP) rule combined with a weight normalization scheme. The STDP rule combines both pre- and post-synaptic weight updates in order to detect temporally coincidental spikes from connected neurons. Our STDP rule has the form:

$$\Delta w = \begin{cases} \eta_{\text{post}} x_{\text{pre}} & \text{on post-synaptic spike;} \\ -\eta_{\text{pre}} x_{\text{post}} & \text{on pre-synaptic spike.} \end{cases} \tag{3}$$

The parameters $\eta_{\text{pre}}, \eta_{\text{post}}$ are pre-, post-synaptic learning rates, respectively, and the quantities $x_{\text{pre}}, x_{\text{post}}$ are pre-, post-synaptic *spike traces*, which are set to 1 on (pre-, post-synaptic) spike events, and are exponentially decaying to 0 otherwise. The spike traces are a simple memory trace of a neuron's previous spike, used for implementing STDP in an *online* fashion. In our experiments, we set $\eta_{\text{post}} \gg \eta_{\text{pre}}$ in accordance with the effect that pre-synaptic spikes have on synapse potentiation. In particular, we used values around $\eta_{\text{post}} = 1 \times 10^{-2}$ and $\eta_{\text{pre}} = 1 \times 10^{-4}$, but these (and their exponential decay rate) were subject to grid search in all experiments.

Due to the unbalanced nature of the pre- and post-synaptic STDP updates, another mechanism is required to ensure that synaptic weights do not grow too large. First and foremost, weights are clipped after each iteration to the range $[0, w_{\text{max}}]$, and secondly, the sum of weights incident to a post-synaptic neuron is normalized after each iteration to have sum c_{norm} (*normalization constant*). That is, if there are n_{pre} pre-synaptic neurons, the synapses incident to a single post-synaptic neuron have average value $c_{\text{norm}}/n_{\text{pre}}$. After some manual tuning, we set $c_{\text{norm}} = 0.2$ for all experiments.

3.3. Network architecture: locally-connected SNN

The locally-connected layer is borrowed from the deep learning literature (Chen et al., 2015) and implemented in a spiking neural network. We restrict our discussion to data with two spatial dimensions; e.g., image data, with or without multiple color channels. At its most basic, the layer requires parameters k_1, k_2 (horizontal and vertical kernel size, respectively), s_1, s_2 (horizontal and vertical stride length, respectively), and n_{filters}, the number of filters to learn per input location (*receptive field*). Locations in the input space are determined by the kernel size and stride parameters; the first receptive field resides at the upper left corner of the input space, with spatial extent $k_1 \times k_2$, and the filter is slid over by s_1 pixels until the right edge of the filter is incident to the right edge of the image, at which point the filter is moved down s_2 pixels and back to the left edge of the image, and process starts anew. The process is repeated until the entire input space is tiled with filters.

The SNN implementation of a locally-connected layer simply stipulates that there are synapses from the pre-synaptic to post-synaptic populations which coincide with the location of filters in the pre-synaptic population, and where the size of the post-synaptic population is determined by the parameters $(k_1, k_2, s_1, s_2, n_{\text{filters}})$ of the layer. In this paper, we restrict attention to the case $k_1 = k_2$ and $s_1 = s_2$ to simplify parameter search, and refer to the parameters respectively as k and s.

We refer to a network which contains a locally-connected layer as a *locally-connected spiking neural network* (LC-SNN),

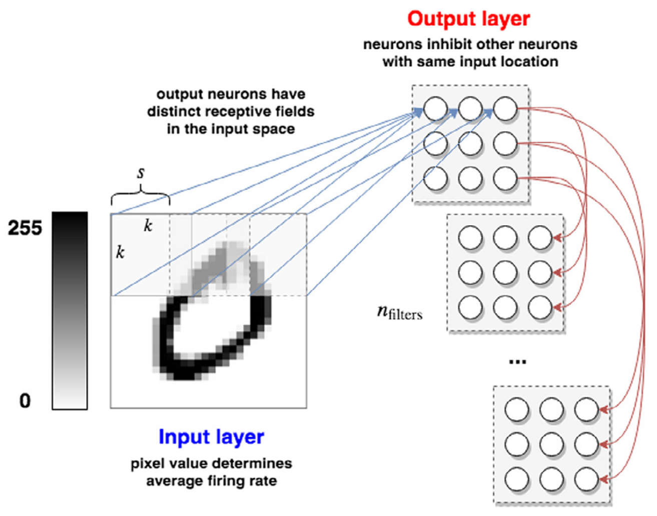

Output layer
neurons inhibit other neurons with same input location
output neurons have distinct receptive fields in the input space
s
k
k
255
0
n_{filters}
...
Input layer
pixel value determines average firing rate

Fig. 1. Two-layer locally-connected spiking neural network architecture.

although the network may include other types of layers. Locally-connected layers are not restricted to receiving connections directly from the input; they may appear in any sequence of processing in a SNN. For example, multiple locally-connected layers can be connected in sequence to detect progressively more complex features in a hierarchical fashion.

In this work, we report classification results from a two-layer (*input*, *output*) LC-SNN. A schematic of this architecture is shown in Fig. 1. Inputs are rate-coded as the average firing rates of Poisson spiking neurons, which are connected with locally specific, STDP-modifiable synapses to the output layer. The output neurons are connected to other neurons which share the same receptive field with fixed inhibitory synapses. This inhibitory connectivity forces neurons in the output to operate in a WTA-like fashion, where, ultimately, one neuron per receptive field remains firing after all others are inhibited by its activity. This is similar to the fully-connected SNN architecture of Diehl and Cook (2015), except that the input population is divided into a number of (possibly overlapping) sub-regions, instead of connecting the full input population to every output neuron.

3.4. Voting mechanism

In a commonly used approach, each individual output spike counts as a single "vote" during the observation period. In this work, we use the n-gram classification method originally explored in Hazan et al. (2018), which makes classification decisions based on length n sequences of spikes that occur in simulation. Namely, we record all length n sequences of spikes from the output layer (indexed by neuron ID) during the training phase. These sequences are aggregated over the training phase, and are paired with the data label for which they occur the most. When a test example is presented, all length n sequences are recorded, and each gets a vote (based on the label they are paired with) towards the classification of the new input. We set $n = 2$ for the sake of computational efficiency.

In this way, we are capturing simple information about the ordering of spikes and harnessing it to distinguish the label of the input data. This method is a step beyond a simple voting approach, in which individual output spikes count as single votes, and which may be considered as the n-gram method with $n = 1$. In the general n-gram approach, we are taking advantage of the time domain aspect of SNNs by supposing that *the ordering of firing matters*; i.e., permuting the order of output spikes

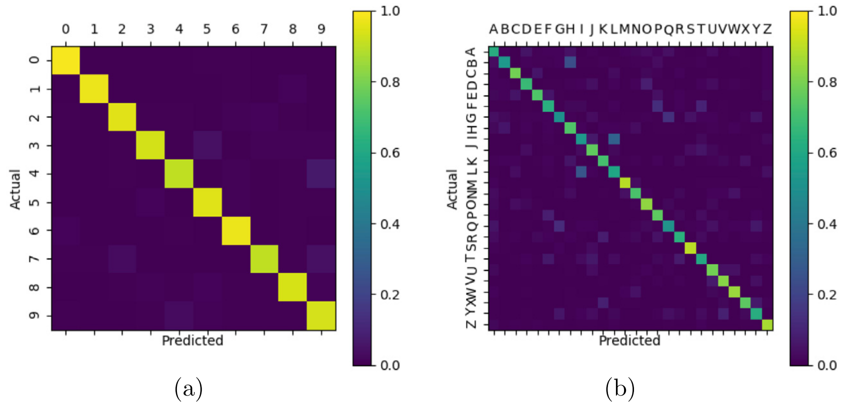


Fig. 2. Average confusion matrices on the MNIST and EMNIST test partitions for 10 independently trained LC-SNNs with $n_{\text{filters}} = 1000$, $k = 16$, $s = 2$. (a) LC-SNN MNIST confusion matrix; the most common mistake is interchanging the digits '4' and '9'. (b) LC-SNN EMNIST confusion matrix; the most common mistake is interchanging the letters 'I' and 'L'.

results in a loss of information. In this work, we report results using the n-gram method, given that this method outperforms our experiments with single spike, vote-based (and re-weighted vote-based) classification schemes (Hazan et al., 2018).

3.5. Model selection

A problem common to many machine learning tasks is picking the best version of a trained model to use for the test phase. During the training phase, the test accuracy of the network is periodically estimated by evaluating it with the n-gram method on a random sample from the training dataset. The version of the network which obtains the highest sample accuracy is saved, to be evaluated on the full test dataset. In our experiments, we estimate the test accuracy after every 250 iterations on a randomly sampled batch of 250 images.

4. Results

In the following sections, we present unsupervised learning results from LC-SNN networks with a two-layer architecture, and show how training convergence of our methods improves over that of previous work on unsupervised learning with SNNs. These results build on those obtained by Saunders et al. (2018).

4.1. Two-layer LC-SNN results

We report test accuracy results from two-layer LC-SNNs with varying numbers of locally-connected filters n_{filters}. The network consists of an input layer of size equal to the input space (images center-cropped to 20 × 20), an output layer with size determined by parameters (k, s, n_{filters}), STDP-modifiable synapses from input to output layer, and a recurrent inhibitory connection in the output layer. As mentioned above, output neurons are inhibited by other output neurons that share the same receptive field in the input space.

4.1.1. MNIST digits dataset

Grid search is conducted to find the best setting of hyper-parameters in terms of classification accuracy, subject only to experimental time constraints. All experiments are limited to running for 24 h with 12Gb of RAM. With large n_{filters} and certain settings of k and s, there is not enough memory or time to complete the simulation, further constraining the hyper-parameter search space. All networks are trained for 60K iterations; i.e., one pass through the MNIST training data, and tested on all 10K test examples. Both training and test examples (cropped to size 20 × 20) are input to the network for 250 ms with a 1 ms simulation time step. The results are reported in Table 1 along with the setting of kernel size (k) and stride (s) that resulted in the best accuracy per number of filters n_{filters}, as well as the number of STDP-modifiable parameters n_{synapses} and output layer neurons n_{neurons} those settings required, computed as a function of k, s, and n_{filters}.

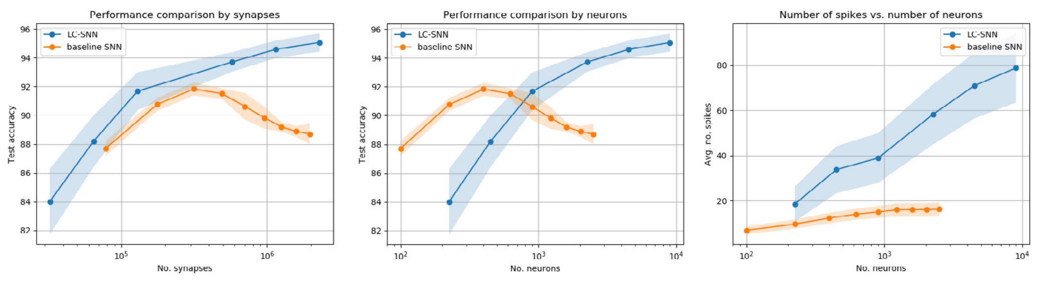
Table 1
Test accuracy results for LC-SNNs with varying numbers of n_{filters} using the 2-gram classification scheme. Each result is estimated from 10 independent train and test phases.

n_{filters}	(k, s)	n_{synapses}	n_{neurons}	mean ± std.
25	(12, 4)	32.4K	225	84.01 ± 2.28
50	(14, 4)	64.8K	450	88.21 ± 1.79
100	(12, 4)	129.6K	900	91.68 ± 1.31
250	(16, 2)	576K	2250	93.71 ± 0.68
500	(16, 2)	1.15M	4500	94.59 ± 0.61
1000	(16, 2)	2.3M	9000	**95.07 ± 0.63**

The average test confusion matrix for the best set of hyper-parameters for LC-SNNs with $n_{\text{filters}} = 1000$ is shown in Fig. 2(a).

These accuracy results are comparable to similar work by Diehl and Cook (2015), in which simple SNNs with a fully-connected layer are trained with STDP and competitive inhibitory interactions to classify the MNIST dataset. We refer to these networks from this point onwards as a baseline. However, our method is able to achieve higher test accuracy with fewer parameters due in part to the parameter-efficient local connectivity and improved training methods. On the other hand, the accuracy of methods is similar when compared instead by the number of neurons utilized. We plot a comparison of n_{synapses} and n_{neurons} versus test accuracy for the baseline SNN and the LC-SNN in Fig. 3. By adding more filters, we expect that LC-SNN test accuracy will exceed that of the baseline SNN's best result.

Moreover, while we limit ourselves to a single pass through the training data, the results of Diehl and Cook (2015) are reported with 1, 3, 7, and 15 passes through the training dataset for networks with 100 (82.9%), 400 (87.0%), 1,600 (91.9%), and 6,400 (95.0%) neurons, respectively. For a better comparison, we re-implemented the baseline SNN in `BindsNET` and ran a superset of the same experiments with a single pass though the training data. These results are also depicted in Fig. 3, from which one can see that the larger baseline SNNs are severely under-trained without multiple passes though the training data. We expect that LC-SNN performance will further improve with multiple presentations of the training data.



(a) (b) (c)

Fig. 3. Various tradeoffs between LC-SNNs and the baseline SNN in terms of the number of learnable weights, neurons, and spikes during the test phase.

Fig. 3(a) demonstrates that, with a single pass through the training data, LC-SNNs achieve better performance than the similarly-sized baseline network counterparts. On the other hand, Fig. 3(b) shows that smaller baseline SNNs attain better accuracy in terms of the number of spiking neurons required in the output layer. However, in our simulations, the main simulation bottlenecks were related to the dimensionality of the tensors implicated in linear algebra operations, namely by the size of the weight matrices of the SNNs. In that respect, the LC-SNN demonstrates a clear advantage over the baseline in terms of simulation training time.

Fig. 3(c) shows that the LC-SNN induces many more spikes than the baseline SNN, due to the distributed nature of the recurrent inhibitory connection; i.e., whereas the baseline network operates similarly to a winner-take-all circuit, in which a single neuron causes all others to fall quiescent, the LC-SNN allows one neuron *per receptive field* to "win" in a similar manner. From an energy efficiency point of view, the LC-SNN would be more costly than the baseline due to the increased activity, but with the added benefit of reduced training time and fewer synaptic connections. This can be seen as a trade-off between energy and training time requirements, where the extreme sparsity of the baseline network's activity results in a bottleneck in training convergence. Overall, the number of LC-SNN spikes is only ≈ 2 to 5-times greater than the baseline networks' for all numbers of neurons considered. Overall, increased spiking activity together with decreased training time results in a similar number of spikes over the course of training.

Next, we compare the performance of our LC-SNN approach with the accuracy of convolutional SNNs. It is important to point out that convolutional SNNs take local processing a step further that our LC-SNN. Namely, both in LC-SNN and in convolutional NNs different neurons receive synapses from different receptive fields; however, synapse parameters are *shared* between locations in the input in convolutional NNs, allowing parameter re-use and enabling feature detection at any location in the input. We do not have such parameter sharing in LC-SNN, which leads to different results.

In the present studies, we do not perform a systematic comparison of LC-SNN with convolutional SNNs, rather we make some comparisons using data from the literature. Table 2 displays the accuracy obtained by convolutional SNNs trained layer-wise with STDP, using MNIST data; the results are based on Lee, Srinivasan, Panda, and Roy (2018). These SNNs are trained in an unsupervised, layer-wise fashion with STDP, except for the final layer, which is trained with a supervised STDP scheme to perform the classification task. We observe, that the convolutional networks in Lee et al. (2018) under-perform the proposed LC-SNN architecture in terms of the number of neurons employed, as shown in Fig. 3(b). Moreover, convolutional networks require in some cases an order of magnitude more neurons for the same accuracy as LC-SNNs; at the same time, they use significantly less synapses. Further work is planned for a systematic comparison of LC-SNN and their convolutional counterparts.

Table 2
MNIST test set accuracy results of networks from Lee et al. (2018) for various settings of k (convolutional kernel width and height) and n_{layers} (number of convolutional layers).

k	n_{layers}	# Neurons $\times 10^4$	# Synapses $\times 10^4$	Accuracy %
3	1	1.43	2.71	85
	2	2.31	2.33	90
	3	3.05	1.98	86
5	1	1.23	2.34	86
	2	1.80	1.68	85
	3	2.15	1.14	80
7	1	1.05	2.01	86
	2	1.37	1.18	82
	3	1.46	0.63	61

Table 3 compares a collection of results from training spiking neural networks to classify the MNIST handwritten digits. We consider methods which explicitly use labels to learn all network parameters *supervised*, whereas methods which employ a partial unsupervised training are denoted *semi-supervised*. Of all considered unsupervised methods, our approach reaches state of the art results. On the other hand, semi-supervised and supervised methods have often higher accuracy, as they benefit from the additional information incorporated in label assignment directly available in the learning procedure.

4.1.2. EMNIST: letters partition

The partition of the EMNIST dataset containing the 26 capital letters of the English alphabet is used to once again test our LC-SNN approach against the baseline network architecture. Grid search is used to select the best setting of hyper-parameters (kernel size, stride, learning rate, and learning rate decay) per setting of n_{filters}. A single pass through the 124K training examples is used (4.8K per class) regardless of network size. The results are reported in Table 4.

For reference, grid search was performed over hyper-parameters of the baseline SNN, which obtained its best accuracy of 66.56% $\pm$ 1.96% with $n_{\text{neurons}} = 900$. As with the LC-SNN, a single training epoch is used before the network is evaluated on the test partition. The best setting of LC-SNN hyper-parameters for $n_{\text{filters}} = 500$ has 648K trainable synapse weights, comparable to the best baseline network with 705.6K parameters, while achieving better performance. Again, as the number of neurons in the output layer of the baseline network is increased, the number of passes through the training data needed to converge in performance increases dramatically. The average test confusion

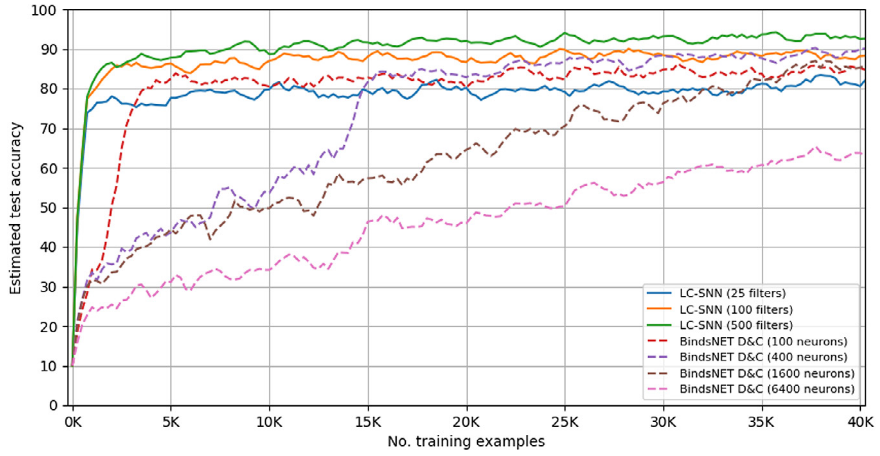


Fig. 4. Training performance of our LC-SNN architecture and the fully-connected baseline SNN replicated in BindsNET (denoted D&C in the legend). Shown are the estimated test accuracies on a single training phase (each using the same training data presentation order) evaluated at every 250 training examples.

Table 3
Comparison of SNN training results on the MNIST dataset. See respective papers for details on SNN architectures and learning procedures.

Paper	Architecture	Learning	Mean	Std. dev.
Ours	Local + recurrent connections	Unsupervised	95.07	0.63
Diehl and Cook (2015)	Dense + recurrent connections	Unsupervised	95.00	–
Tavanaei and Maida (2015)	Input segmentation	Unsupervised	75.93	–
Allred and Roy (2016)	Dense + recurrent connections	Unsupervised	86.59	–
Panda and Roy (2016)	Autoencoder	Semi-supervised	99.08	–
Demin and Nekhaev (2018)	Dense + lateral connections	Supervised/semi-supervised	95.4/72.1	–
Tavanaei and Maida (2017)	Convolutional	Semi-supervised	96.95	0.08
Lee et al. (2018)	Convolutional	Semi-supervised	99.28	$\approx$ 0.1
Tavanaei et al. (2018)	Convolutional	Semi-supervised	97.20	0.07
Kheradpisheh et al. (2018)	Convolutional	Semi-supervised	98.4	–

Table 4
Test accuracy results for LC-SNNs on the EMNIST "letters" partition with varying numbers of n_{filters} using the 2-gram classification scheme. Each result is estimated from 10 independent train and test phases.

n_{filters}	(k, s)	mean $\pm$ std.
25	(12, 4)	49.61 $\pm$ 1.98
50	(12, 4)	54.44 $\pm$ 2.04
100	(12, 4)	62.34 $\pm$ 2.11
250	(12, 4)	65.47 $\pm$ 1.73
500	(12, 4)	67.58 $\pm$ 1.78
1000	(16, 2)	**69.73** $\pm$ **1.57**

matrix for the best set of hyper-parameters for LC-SNNs with $n_{\text{filters}} = 1000$ is shown in Fig. 2(b).

To our knowledge, this work is the first to report results on the training of spiking neural networks to classify the EMNIST "letters" partition. Although we focus on only the 26 capital letters of the English alphabet, it remains to be seen if this approach will scale to all 62 categories of the original dataset.

4.2. Training convergence

A desirable feature of the LC-SNN approach is the rapid convergence speed which the distributed sub-network learning enables. That is, since only neurons within the same sub-population of the output layer (as determined by unique receptive fields) compete to learn filters, as many neurons as there are receptive fields can spike and update their synapse parameters at once. This leads to many more parameter updates than was previously possible, and a faster convergence to the network's peak classification accuracy.

Moreover, this convergence does not appear to depend on the size of the network; i.e., the number of locally-connected filters. On the other hand, the baseline SNN networks require more training iterations as the size of the network increases in order to reach their best performance. This phenomenon is depicted in Fig. 4. Note that LC-SNNs often come within a few percentage points of their peak accuracy within $\approx$ 5K training examples, whereas the baseline SNNs often require more than 60K examples to converge; e.g., for networks with 1,600 neurons, as demonstrated empirically in Section 4.1.1.

5. Robustness tests

In the following, we perform the following robustness tests on a fixed, trained LC-SNN:

1. Randomly deleting plastic synapses (from the input to the output layer) post-training.
2. Randomly deleting neurons from the output layer (equivalent to deleting all incoming synapses to said neurons) post-training.

We fix a single LC-SNN (with $k = 14$, $s = 2$, and $n_{\text{filters}} = 250$) and baseline SNN (with $n_{\text{neurons}} = 900$) which both achieved $\approx$ 93.5% test accuracy on the MNIST dataset, which are used in all robustness tests. The filters of the LC-SNN are visualized in Fig. 6(a). Note that the baseline network required 3 passes through the training data in order to attain the same level of accuracy as the LC-SNN with only a single training epoch.

5.1. Deleting synapses

Each STDP-modifiable synapse in the aforementioned fixed LC-SNN is deleted with probability p_{delete}, and the test phase over

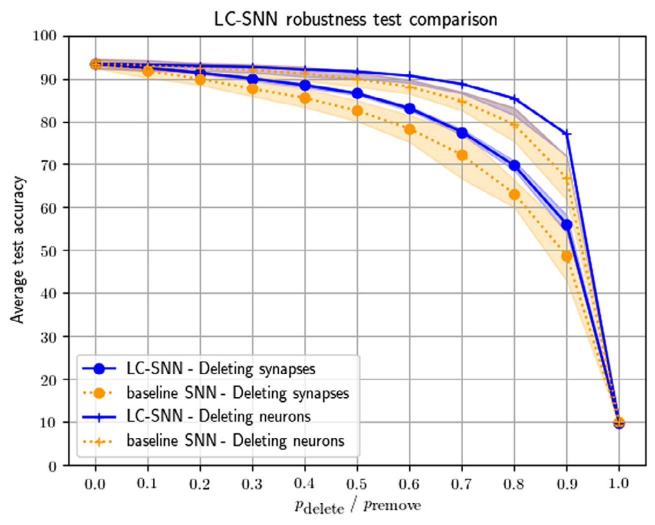


Fig. 5. Robustness of LC-SNN vs. baseline SNN to random synapse and neuron deletion.

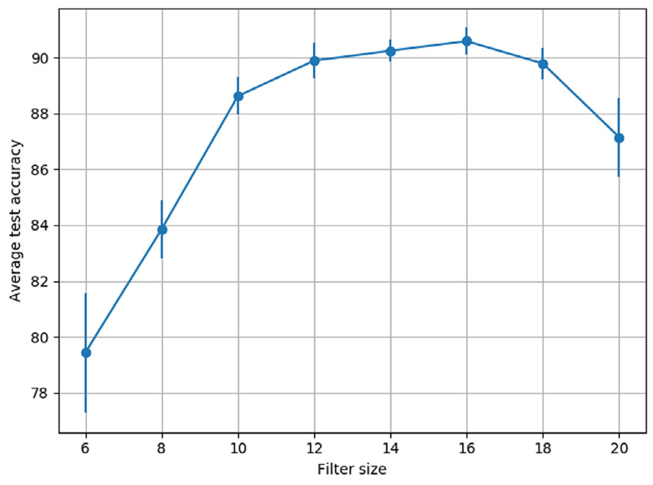


Fig. 7. Average and standard deviations of test accuracy of 2-layer LC-SNNs with 100 filters, stride $s = 2$, and variable filter size k.

the MNIST digits is run as normal. This gives us a sense of how performance degrades in the presence of missing inputs. We run the test phase 5 times and compute the average and standard deviation statistics of the performance result. The results are shown in Fig. 5.

As can be seen in the figure, the network without any synapses deleted maintains an accuracy of $\approx$ 93.5%, while increasing p_{delete} leads to a smooth degradation in performance. Interesting, with 50% of synapses removed, nearly 90% accuracy is maintained, and with 90% of synapses removed, nearly 60% accuracy is maintained. The baseline SNN drops off in performance at a faster rate than the locally connected network, possibly due to the fully-connected representation it learns, as opposed to the distributed nature of the LC-SNN representation.

An example deletion of synapses with $p_{\text{delete}} = 0.5$ is shown in Fig. 6(b).

5.2. Deleting neurons

Similar to the approach in Section 5.1, we consider neurons in the output layer of the LC-SNN, and remove each with probability p_{remove}. We can implement this simply by setting all incoming synapses to a "removed" neuron to 0, effectively cutting it off from any input activity. This experiment assesses the degree of redundancy encoded in individual neurons, as opposed to the previous experiment which assessed the importance of individual synapses. The MNIST test phase is run 5 times to obtain mean and standard deviation statistics of the test performance result, which are depicted Fig. 5.

From the figure, we conclude that the network filters encode a high degree of redundancy, as deleting large numbers of neurons results in relatively little reduction in accuracy. For example, deleting 90% of neurons reduces accuracy down to just under 80%, reducing the number of filters down from 150 to approximately 15 per receptive field. The baseline SNN's accuracy drops at a faster rate, again possibly due to the difference in nature of the two architecture's learned representations. On the other hand, since the compared baseline SNN has fewer neurons, deleting a large percentage of them may have a more dramatic effect than for the LC-SNN. An example removal of neurons with p_{remove} is shown in Fig. 6(c).

6. Discussion

6.1. Impact of kernel size on classification performance

We briefly discuss the effect of the choice of filter size k on the classification performance of the LC-SNN on the MNIST dataset. Fig. 7 compares the test accuracy, averaged over 5 trials, of LC-SNNs with 100 filters, stride 2, and variable filter size: $k \in \{6, 8, \ldots, 20\}$. All other hyper-parameters are fixed. Empirically, we find $k = 16$ optimal, and nearby values give similar performance, although our search is incomplete due to constraints

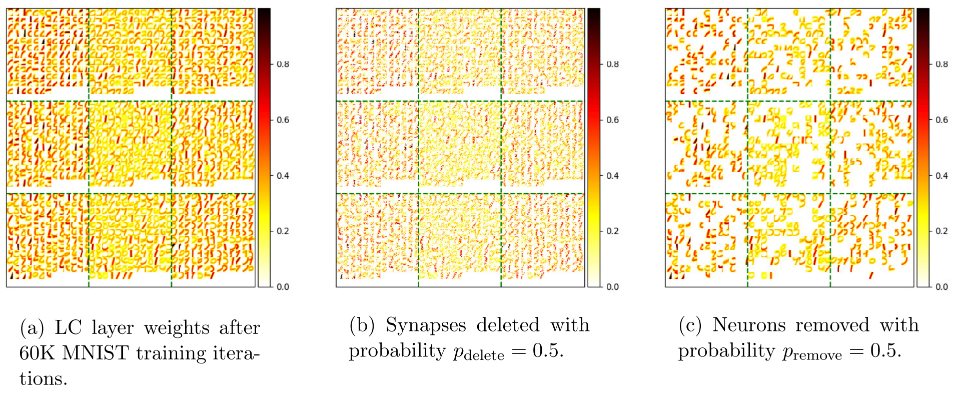

(a) LC layer weights after 60K MNIST training iterations.
(b) Synapses deleted with probability $p_{\text{delete}} = 0.5$.
(c) Neurons removed with probability $p_{\text{remove}} = 0.5$.

Fig. 6. LC-SNN filters and random deletion of synapses/removal of neurons.

Table 5
Test accuracy results for LC-SNNs with different classification schemes. Each result is with networks with $n_{\text{filters}} = 100$ and $(k, s) = (12, 4)$.

Method	Accuracy
Single thread: 2-gram	91.68
Multi-thread: 2-gram	88.91
Multi-thread (Sum of spikes): 3 votes	93.21

on computation. Namely, as filter size decreases, the number of output neurons increases dramatically, increasing simulation wall-clock time. We expect that, with $k < 6$, classification accuracy will quickly drop off to random chance (10%). Images are center-cropped to 20×20 pixels, rendering the network with $k = 20$ identical to the baseline, fully-connected SNN architecture. Indeed, LC-SNNs with this setting of parameters achieve similar accuracy to the baseline SNN with the same number of output neurons (cf. Fig. 3). Taking the analysis one step further, the inclusion of non-square filters and further hyper-parameter tuning may reveal a more optimal setting of filter size. This topic is the objective of future studies.

6.2. Multi-thread decision algorithm

Here we introduce an additional biologically-motivated method for deriving decisions from the LC-SNN spike trains. Previous results were obtained using the n-gram approach with very good accuracy. In the proposed multi-thread decision rule, we do not simply select the winning n-grams from all spike trains, but consider the location (patch) in the output layer where the winning n-grams come from. In this way, we contrast the temporal dimension of spike trains with their spatial distribution in multiple locations, representing multiple "threads". Each of the patches forms their own 2-grams and vote on the top N classes (the 1st class gets the largest weight, while the Nth class gets smallest), where the "top" classes are determined by the greatest number of spikes. We also tested a multi-thread classification method based on the sum of spikes. Unlike the n-gram method, this method does not utilize temporal information. Each neuron in a patch is mapped to the class label for which it spiked the most during the training phase. During the test phase, the classification decision is made by choosing the class that gets the most votes. This method of classification is much faster than the n-gram method as it does not require aggregating n-gram counts, and does not rely on the temporal aspect of the spikes.

The results with this multi-thread algorithm applied to MNIST classification are summarized in Table 5. Note that the multi-thread sum of spikes method where each patch vote for top 3 classes achieves similar performance to the logistic regression accuracy of 94.39%.

It is important to point out that the multi-thread approach has strong biological motivation and it may provide insights on the operation of biological mechanisms of spike processing. This is illustrated by Fig. 8, where the trained filters of the winning neurons are depicted for the single-thread 2-gram and multi-thread approaches. One can see that the multi-thread filters are in line with the expectation that a well-functioning processing system can efficiently extract information from various locations of the input data. The method has the promise of being scalable to large input data sizes.

6.3. Removing contiguous input patches

An additional test can be carried to test the robustness of trained networks to the removal of contiguous patches in the input data. Given a side length l, we may set random $l \times l$ patches of the test data to zero, causing the corresponding input neurons not to spike, and test networks on this modified dataset. We expect that LC-SNNs will perform better than baseline SNNs in this setting, since the former are trained such that separate sub-populations of neurons spike for separate locations in the input, whereas the latter are trained such that individual neurons recognize only prototypical whole digits. The more $l \times l$ patches are set to zero, and the larger the value of l, the lower the expected performance. We leave this additional robustness experiment to future work.

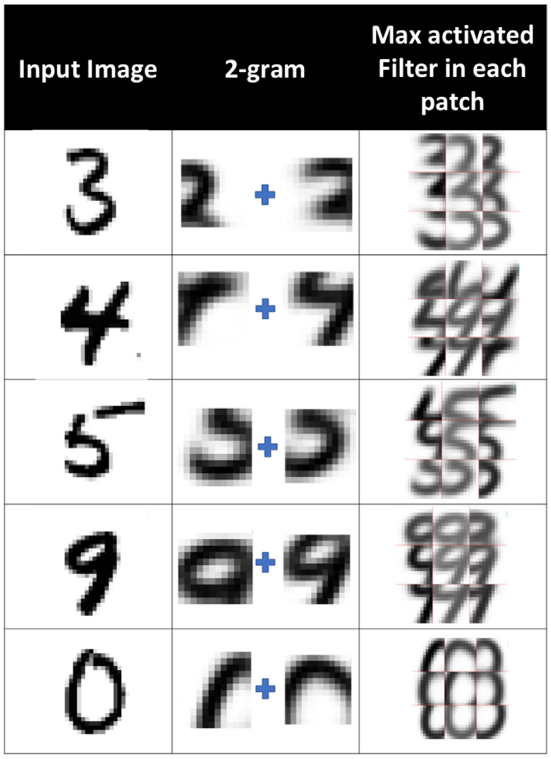


Fig. 8. Illustration of the trained filters obtained with n-gram and multi-thread approaches for a few input representative digit classes.

7. Concluding remarks

We have demonstrated a new architectural building block, the locally-connected layer, for spiking neural networks and applied it to solve simple image recognition tasks. The layer allows two-layer networks to divide and conquer the input space by learning a distributed representation. Not only does this division of labor allow a reduction in the number of learnable parameters needed to achieve a certain quality of representation (as measured by classification accuracy), but it also reduces the number of training examples required to achieve it. These LC-SNN properties have been empirically demonstrated on the MNIST and EMNIST datasets, where we have shown improved performance with shorter training times compared to a baseline, fully-connected SNN.

The locally-connected layer may easily be included as a component in a multi-layer spiking neural network alongside with other feed-forward, lateral, or recurrent connections. It encodes the inductive bias that features in the pre-synaptic space may be divided into possibly overlapping regions without loss of information, and for the added benefit of reduced parameter requirements and reduced training time. Although the LC-SNN induces approximately 2–5 times more spikes during its operation than the baseline SNN, we argue that this is an acceptable trade-off in light of the aforementioned convergence speed and accuracy properties. Indeed, we argue that the extreme sparsity of activity in the baseline networks led to a poor, centralized representation of the dataset akin to the notion of the gnostic ("grandmother") cell of the neuroscience literature (Gross, 2002). The LC-SNN layer may be implemented in neuromorphic chips and used to operate on event-based data (Pfeiffer & Pfeil, 2018), a domain in which spiking neural networks are known to have advantages.

It remains to be seen whether methods of unsupervised learning with SNNs will be useful in learning representations of more complex datasets useful for classification purposes. Whereas the optical character datasets considered in this work are relatively simple, to our knowledge, it has not yet been shown that SNNs trained with unsupervised, local learning rules can scale to the complexity of real-world image data. However, we have added a component to the SNN builder's toolbox that may invigorate further lines of work on adapting SNNs for more complex machine learning problems. Using it, we have shown that local learning rules and adaptive neurons combined with competitive inhibitory interactions may be applied to different SNN architectural primitives with benefits for feature learning.

Acknowledgments

The work of D.J.S., D.P., H.H., and R.K. has been supported in part by Defense Advanced Research Projects Agency, United States Grant, DARPA/MTO HR0011-16-l-0006. Partial support has been provided to R.K. by National Science Foundation, United States Grant NSF-CRCNS-DMS-13-11165. H.T.S. contribution to this work took place prior to assuming her position at DARPA. The information contained in this work does not necessarily reflect the position or the policy of the Government.

References

Allred, J. M., & Roy, K. (2016). Unsupervised incremental STDP learning using forced firing of dormant or idle neurons. In *2016 international joint conference on neural networks* (pp. 2492–2499).

Bekolay, T., Bergstra, J., Hunsberger, E., DeWolf, T., Stewart, T. C., Rasmussen, D., et al. (2014). Nengo: a python tool for building large-scale functional brain models. *Frontiers in Neuroinformatics*, *7*(48), [Online]. Available http://www.frontiersin.org/neuroinformatics/10.3389/fninf.2013.00048/abstract.

Carnevale, N. T., & Hines, M. L. (2006). *The NEURON Book*. Cambridge: University Press.

Chen, Y. H., Lopez-Moreno, I., Sainath, T. N., Visontai, M., Alvarez, R., & Parada, C. (2015). Locally-connected and convolutional neural networks for small footprint speaker recognition. In *INTERSPEECH*.

Cohen, G., Afshar, S., Tapson, J., & van Schaik, A. (2017). EMNIST: an extension of MNIST to handwritten letters. ArXiv e-prints.

Cornelis, H., Rodriguez, A. L., Coop, A. D., & Bower, J. M. (2012). Python as a federation tool for genesis 3.0. *PLOS ONE*, *7*(1), 1–11. http://dx.doi.org/10.1371/journal.pone.0029018.

Dan Goodman, F. M., & Brette, R. (2009). The brian simulator. *Frontiers in Neuroscience*, *3*, [Online]. Available https://www.frontiersin.org/articles/10.3389/neuro.01.026.2009/full.

Demin, V., & Nekhaev, D. (2018). Recurrent spiking neural network learning based on a competitive maximization of neuronal activity. *Frontiers in Neuroinformatics*, *12*, 79, [Online]. Available https://www.frontiersin.org/article/10.3389/fninf.2018.00079.

Diehl, P. U., & Cook, M. (2015). Unsupervised learning of digit recognition using spike-timing-dependent plasticity. *Frontiers in Computational Neuroscience*.

Diehl, P. U., Neil, D., Binas, J., Cook, M., Liu, S. C., & Pfeiffer, M. (2015). Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 international joint conference on neural networks* (pp. 1–8).

Florian, R. V. (2007). Reinforcement learning through modulation of spike-timing-dependent synaptic plasticity. *Neural Computation*, *19*(6), 1468–1502.

Gewaltig, M.-O., & Diesmann, M. (2007). Nest (neural simulation tool). *Scholarpedia*, *2*(4), 1430.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press, http://www.deeplearningbook.org.

Goodman, D. F. M., & Brette, R. (2009). The brian simulator. *Frontiers in Computational Neuroscience*.

Gross, C. G. (2002). Genealogy of the "grandmother cell". *The Neuroscientist*, *8*(5), 512–518. http://dx.doi.org/10.1177/107385802237175.

Hazan, H., Saunders, D. J., Khan, H., Sanghavi, D. T., Siegelmann, H. T., & Kozma, R. (2018). BindsNET: A machine learning-oriented spiking neural networks library in Python. *Frontiers in Neuroinformatics*, *12*(89).

Hazan, H., Saunders, D. J., Sanghavi, D. T., Siegelmann, H. T., & Kozma, R. (2018). Unsupervised learning with self-organizing spiking neural networks. In *2018 international joint conference on neural networks* (pp. 1–6).

Kasabov, N. K. (2014). Neucube: a spiking neural network architecture for mapping, learning and understanding of spatio-temporal brain data. *Neural Networks*, *52*, 62–76, [Online]. Available http://www.sciencedirect.com/science/article/pii/S0893608014000070.

Kauderer-Abrams, E. (2016). Quantifying translation-invariance in convolutional neural networks. *CoRR*, *abs/1801.01450*.

Kheradpisheh, S. R., Ganjtabesh, M., Thorpe, S. J., & Masquelier, T. (2018). Stdp-based spiking deep convolutional neural networks for object recognition. *Neural Networks*, *99*, 56–67, [Online]. Available http://www.sciencedirect.com/science/article/pii/S0893608017302903.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, (521), 436–444.

LeCun, Y., & Cortes, C. (2010). MNIST Handwritten digit database. [Online] Available http://yann.lecun.com/exdb/mnist/.

Lee, C., Panda, P., Srinivasan, G., & Roy, K. (2018). Training deep spiking convolutional neural networks with STDP-based unsupervised pre-training followed by supervised fine-tuning. *Frontiers in Neuroscience*, *12*, 435. http://dx.doi.org/10.3389/fnins.2018.00435.

Lee, C., Srinivasan, G., Panda, P., & Roy, K. (2018). Deep spiking convolutional neural network trained with unsupervised spike timing dependent plasticity. *IEEE Transactions on Cognitive and Developmental Systems*, [ISSN: 2379-8920] http://dx.doi.org/10.1109/TCDS.2018.2833071, 1-1.

Maass, W. (1997). Networks of spiking neurons: the third generation of neural network models. *Neural Networks*, *10*(9), 1659–1671, [Online]. Available http://www.sciencedirect.com/science/article/pii/S0893608097000117.

Mozafari, M., Kheradpisheh, S. R., Masquelier, T., Nowzari-Dalini, A., & Ganjtabesh, M. (2018). First-spike based visual categorization using reward-modulated STDP. *IEEE Transactions on Neural Networks and Learning Systems*.

Panda, P., & Roy, K. (2016). Unsupervised regenerative learning of hierarchical features in spiking deep networks for object recognition. *CoRR*, *abs/1602.01510*, [Online]. Available arXiv:1602.01510.

Pfeiffer, M., & Pfeil, T. (2018). Deep learning with spiking neurons: opportunities and challenges. *Frontiers in Neuroscience*, *12*, 774, [Online]. Available https://www.frontiersin.org/article/10.3389/fnins.2018.00774.

Saunders, D. J., Siegelmann, H. T., Kozma, R., & Ruszinko, M. (2018). STDP learning of image patches with convolutional spiking neural networks. In *2018 international joint conference on neural networks* (pp. 1–7).

Sengupta, A., Ye, Y., Wang, R., Liu, C., & Roy, K. (2018). Going deeper in spiking neural networks: VGG and residual architectures. *CoRR*, *abs/1802.02627*, [Online]. Available http://arxiv.org/abs/1802.02627.

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. S. (2018). Deep learning in spiking neural networks. *CoRR*, *abs/1804.08150*, [Online]. Available http://arxiv.org/abs/1804.08150.

Tavanaei, A., Kirby, Z., & Maida, A. S. (2018). Training spiking convnets by STDP and gradient descent. In *2018 international joint conference on neural networks* (pp. 1–8).

Tavanaei, A., & Maida, A. S. (2015). A minimal spiking neural network to rapidly train and classify handwritten digits in binary and 10-digit tasks.

Tavanaei, A., & Maida, A. S. (2016). Bio-inspired spiking convolutional neural network using layer-wise sparse coding and STDP learning. *CoRR*, *abs/1611.03000*.

Tavanaei, A., & Maida, A. S. (2017). Multi-layer unsupervised learning in a spiking convolutional neural network. In *2017 international joint conference on neural networks* (pp. 2023–2030).

Vitay, J., Dinkelbach, H., & Hamker, F. (2015). Annarchy: a code generation approach to neural simulations on parallel hardware. *Frontiers in Neuroinformatics*, *9*, 19, [Online]. Available https://www.frontiersin.org/article/10.3389/fninf.2015.00019.

Zambrano, D., Nusselder, R., Scholte, H. S., & Bohte, S. M. (2017). Efficient computation in adaptive artificial spiking neural networks. *CoRR*, *abs/1710.04838*, [Online]. Available http://arxiv.org/abs/1710.04838.

Zappacosta, S., Mannella, F., Mirolli, M., & Baldassarre, G. (2018). General differential hebbian learning: capturing temporal relations between events in neural networks and the brain. *PLoS Computational Biology*, *14*(8), 1–30. http://dx.doi.org/10.1371/journal.pcbi.1006227.