Fast Convergence of Competitive Spiking Neural Networks with Sample-Based Weight Initialization

Paolo Gabriel Cachi[1(✉)], Sebastián Ventura[2], and Krzysztof Jozef Cios[1,3]

[1] Virginia Commonwealth University, Richmond, VA 23220, USA
{cachidelgadpg,kcios}@vcu.edu
[2] Universidad de Córdoba, Córdoba, Spain
sventura@uco.es
[3] Polish Academy of Sciences, Gliwice, Poland

Abstract. Recent work on spiking neural networks showed good progress towards unsupervised feature learning. In particular, networks called Competitive Spiking Neural Networks (CSNN) achieve reasonable accuracy in classification tasks. However, two major disadvantages limit their practical applications: high computational complexity and slow convergence. While the first problem has partially been addressed with the development of neuromorphic hardware, no work has addressed the latter problem. In this paper we show that the number of samples the CSNN needs to converge can be reduced significantly by a proposed new weight initialization. The proposed method uses input samples as initial values for the connection weights. Surprisingly, this simple initialization reduces the number of training samples needed for convergence by an order of magnitude without loss of accuracy. We use the MNIST dataset to show that the method is robust even when not all classes are seen during initialization.

Keywords: Spiking Neural Networks · Competitive learning · Unsupervised feature learning

1 Introduction

The competitive learning paradigm has been successful in dealing with unsupervised data [7,24,33]. In competitive learning, units/neurons compete with each other for the right to respond to the given input. The winner units are then updated and become more specialized. At the end of training, all units are tuned to respond to specific input patterns and their activation is used to classify new unseen samples [29,33].

Competitive learning inspired design of several clustering and unsupervised feature learning algorithms, such as Vector Quantization [22], Self Organizing Maps (SOM) [17], and Deep Self Organizing Maps (DSOM) [39]. While these algorithms are good for extracting spatial information from unlabeled data, their use for classification tasks is limited by their performance. For example, classification accuracy achieved by DSOM on the MNIST dataset was 87.12% [39], compared with 99.79% achieved by current state of the art fully supervised algorithms [5,34,38].

M.-J. Lesot et al. (Eds.): IPMU 2020, CCIS 1239, pp. 773–786, 2020.
https://doi.org/10.1007/978-3-030-50153-2_57

A considerable increase in classification performance has been achieved by competitive learning networks using spiking neurons. Spiking neurons are dynamic units that respond not only to the current state of their inputs, as traditional neural networks do [11,19,33], but also take into account their previous states [2,23,32]. These networks, named Competitive Spiking Neural Networks (CSNN), achieved 95% accuracy on the MNIST dataset, almost 8% increase over DSOM [6,39].

The CSNNs, however, are limited by two factors: high computational complexity and slow convergence. The first problem is due to the fact that the spiking neurons are implemented as independent units, which requires using highly parallel processors. The neuromorphic processors provide parallel architecture needed for these networks and that, at the same time, considerably reduce energy consumption when compared with traditional deep neural network implementations [8,27]. On the other hand, there is little, if any, work on reducing the network convergence time. The state of the art CSNN [6] needs around 20,000 samples to converge which in computational time represents more than 2 h of training using a single thread implementation on an Intel Core i9-9900K processor. Thus, developing a method that would require less number of samples for training is urgently needed to expand their real world applications [14,25,26].

Here, we show that using the input samples as initial weights in the CSNN reduces the number of training samples needed for convergence by an order of magnitude, and with no loss in accuracy. We use different combinations of the initial weights to check the method's robustness to cases where the samples used for initialization do not represent all classes in the data. The method is evaluated on the MNIST dataset.

The rest of the paper is organized as follows. Section 2 presents a review of work on the CSNNs. A general overview of the network topology and its main characteristics are presented in Sect. 3. Section 4 introduces the proposed initialization method. Section 5 describes the dataset used, experimental settings, and evaluation metrics. Section 6 discusses the results. Section 7 ends with conclusions.

2 Related Work

The use of CSNN for unsupervised feature learning was originally proposed in [31]. The authors used an array of memrirstors to implement a CSNN for unsupervised feature learning. Their network used 300 spiking-like units to achieve 93% accuracy on the MNIST dataset and showed robustness to parameter variations. An extension of this work achieved accuracy of 95% but at the cost of using 6,400 complex spiking neurons [6]. In terms of the convergence time, both implementations converged only after 20,000 sample presentations, which in combination with the high computational cost undermines their practical applications.

Other authors used self-organizing and convolutional spiking network implementations. In [12] the authors reported accuracy of 94.07% using 1,600 neurons, however, we believe that this increase of performance was due more to the use of a specific classification method. In fact, when using the same classification method as in [6], this CSNN achieves only 92.96% accuracy. The convolutional spiking network had only 84.3% accuracy [35].

3 Competitive Spiking Neural Networks

The CSNN uses a spiking neuron layer with Spike Time Dependence Plasticity (STDP), lateral inhibition, and homeostasis to learn input data patterns in an unsupervised way. At any given time, the output neuron that is most active (spikes the most) represents the current data input.

3.1 Network's Topology

The detailed network topology is shown in Fig. 1. The Sensory layer first transforms an N-dimensional input vector, via N Poisson units (using Poisson distribution), into a series of spikes, which are fed into a layer of M spiking neurons with lateral inhibition.

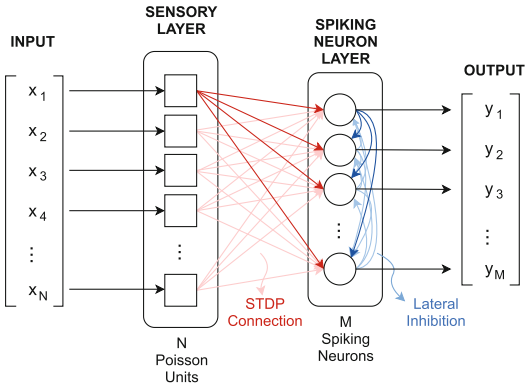


Fig. 1. Competitive spiking neural network topology

The N sensory units are fully connected to the M spiking neurons. The learning process uses the STDP implementation of the Konorski/Hebb rule [13,18]. All spiking neurons are connected with all others by fixed inhibitory weights; this is known as lateral inhibition used to ensure that only one neuron fires for a given input. The specific mechanisms used for both connections are described in detail in Sects. 3.3 and 3.4.

3.2 Spiking Neuron Model

In this paper we use a spiking neuron model known as Integrate and Fire [10,15,16]. This model uses a differential equation and a threshold mechanism to define the neuron behavior:

$$\tau \frac{du}{dt} = f(u) + i(u,t) \tag{1}$$

$$t^f : \quad u(t^f) = \theta_{reset} \quad and \quad \frac{du(t)}{dt}\Big|_{t=t^f} > 0 \tag{2}$$

Equation 1 describes the evolution of the membrane potential u in terms of a linear/non-linear function $f(u)$ and a synaptic input term $i(u,t)$. Equation 2 defines the fire time t^f as the moment the membrane potential u crosses, from below, its threshold value θ_{reset}. When this happens, a spike is generated and the neuron enters a refractory period, during which it cannot respond to any input, after that the neuron's membrane potential is reset.

The choice of $f(u)$ gives rise to different variations of the Integrate and Fire neuron model [10]. A linear choice (Eq. 3) defines the Leaky Integrate and Fire Model. Non linear choice gives rise to the Exponential Integrate and Fire Model (Eq. 4) and the Quadratic Integrate and Fire Model (Eq. 5):

$$f_1(u) = -(u - u_{rest}) \tag{3}$$

$$f_2(u) = -(u - u_{rest}) + \Delta_T \exp \frac{(u - \vartheta_{rh})}{\Delta_T} \tag{4}$$

$$f_3(u) = a_0(u - u_{rest})(u - u_c) \tag{5}$$

where $f_1(u)$, $f_2(u)$ and $f_3(u)$ are the linear, exponential and quadratic function terms, u_{rest} represents the resting potential, Δ_T the sharpness factor, ϑ_{rh} the threshold, and a_0 and u_c are constant factors with $a_0 > 0$ and $u_c > u_{rest}$.

There are two typical choices for the synaptic contribution term $i(u,t)$. The simplest one considers the synaptic inputs as direct membrane potential modifiers, Eq. 6:

$$i(u,t) = i(t) = R \sum_j \sum_{t^f} w_j \rho(t - t_j^f) \tag{6}$$

A more complex one uses additional differential equations (Eqs. 7 and 8) for conductance level contributions [3,4]:

$$i(u,t) = g(t)(u_{input} - u) \tag{7}$$

$$\frac{dg}{dt} = -\frac{g}{\tau_g} + \sum_j \sum_{t^f} w_j \delta(t - t_j^f) \tag{8}$$

where $g(t)$ represents the conductance contribution, τ_g the conductance time constant, w_j the synaptic weight of connection j, and t_j^f is the firing time of the input neuron j.

3.3 Learning Rule

The CSNN uses STDP to modify the connection weights between the Poisson units and the spiking neurons [31]. In STDP, the adjustment of the strength of each weight depends on the relative activity between the pre- and post-synaptic neurons, Eq. 9:

$$STDP(\Delta t) = \begin{cases} \alpha_+ \exp(-\Delta t/\tau_+) & \text{if } \Delta t > 0 \\ -\alpha_- \exp(\Delta t/\tau_-) & \text{if } \Delta t \leq 0 \end{cases} \tag{9}$$

where δt is the time between pre- and post-synaptic neuron firings, α_+ and α_- are learning rates, and τ_+ and τ_- are time constants [1,37]. Over time, and because STPD

takes effect only after a spiking event, the synaptic weights become selective to specific input patterns. In contrast to the gradient descent learning, STDP is a local rule that uses information only from the pre- and post-synaptic neuron firings, while gradient descent updates all weights based on the minimization of a global loss function.

STDP can be implemented using exponential decay variables to keep track of the weight update values a_{pre} and a_{post}, Eqs. 10 and 11 [28]. When a pre-synaptic spike is registered, the synaptic weight is decreased by a_{post} and a_{pre} is updated to a constant value A_{pre}. In contrast, when a post-synaptic spike is registered, the weight value is increased by a_{pre} and at the same time a_{post} is updated to A_{post}.

$$\tau_{pre}\frac{da_{pre}}{dt} = -a_{pre} \tag{10}$$

$$\tau_{post}\frac{da_{post}}{dt} = -a_{post} \tag{11}$$

where a_{pre} and a_{post} are the pre- and post-synaptic trace values used to update connection weights in the event of a pre- or post-synaptic neuron spikes. τ_{pre} and τ_{post} are the constant time factors for each exponential decay variable.

3.4 Lateral Inhibition

All spiking neurons at the spiking layer are connected to each other via direct inhibitory synapses, with the purpose of feedback regulation [6,31]. When a neuron produces a spike, all neurons connected to it receive a negative potentiation (their membrane potentials are decreased) which reduces the neuron's probability of reaching its firing threshold to generate a spike, see Fig. 1.

3.5 Homeostasis

It is important that the membrane firing threshold $\theta_{threshold}$ is adaptive to make sure all neurons have a chance to fire during training. The threshold value is defined by an exponential decay function with a constant increase every time the neuron fires. Thus, a neuron that fired recently is less able to fire again because of its higher threshold value. Equation 12 describes the membrane reset value θ_{reset} as a function of a dynamic variable θ, Eq. 13:

$$\theta_{reset} = \theta_{offset} + \theta \tag{12}$$

$$\frac{d\theta}{dt} = \frac{-\theta}{\tau_\theta} + \sum_{t^f} \alpha\delta(t - t_j^f) \tag{13}$$

where θ_{offset} is the offset value when $\theta = 0$, τ_θ is the time constant and α is the increase constant value [30,40].

3.6 Weight Normalization

The purpose of weight normalization is to limit the total input a neuron receives. To do so, each input connection is normalized according to Eq. 14:

$$w_{ij}^{norm} = w_{ij}\frac{\lambda}{\sum_i w_{ij}} \tag{14}$$

where w_{ij} is the weight value for connection i of neuron j, and λ is the total sum of weights [9,21].

While a straightforward effect of normalization is to balance all input connections, a not so obvious effect can be stated as helping to spread "information" to all active inputs. This means that if one synapse is increased/decreased by STDP, the normalization will average the change in all the incoming inputs. In that way, not only one input is modified, but all of them.

4 Sample-Based Weight Initialization

What are the effects of using STDP, normalization, and lateral inhibition on the network operation. If a neuron, using STDP learning, is excited with a single input image for a long period of time, its synapse weights will increase/decrease proportionally to each pixel input activation rate. The weights corresponding to high pixel values will increase the most. Performing weight normalization bounds the weight changes so the system will not become unstable. These two operations result in the weights trying to copy its input. If more images are used, then the changes are averaged and the final weights are "finding" single prototypes among all the input images. Finally, using lateral inhibition makes sure that the neuron only updates in response to the inputs that are close to its current prototype. For example, a neuron that is following the prototype for number "2" will be only updated with inputs of this class (other inputs of "2") thus increasing its selectiveness.

We use the above analysis to reduce the network's training convergence time as follows. If each neuron strives to find prototypes among the input images, we can reduce the training time by initializing its weights with the input pixel values, which are closer to some of the final prototypes than a random initialization. We thus use the first M (out of P) training samples to serve as initial connection weights between the sensory layer and the M neurons at the spiking neuron layer. Since we use weight normalization, there is no need to re-scale the pixel values. We also tested the effects of using different degrees of blurring filters to soften the contrast in the input images; for that purpose, the OpenCV's blurring filter was used.

The pseudo-code for Sample-Based Weight Initialization is shown in Algorithm 1. The competitive spiking network is instantiated in line 2. Line 3 initializes the connection weights with the resulted images after passing the first M training samples through a 5×5 blurring filter. Line 4 creates a Spike Monitor instance used to keep track of each neuron's firing events. The FOR loop in lines 5 through 9 presents all the $P - M$ remaining training samples. First, the connection weights are normalized. Then, the firing rates of the Poisson neurons are set based on the input image. Each sample is presented for 350 ms.

After training, a new run over all training samples, with STDP turned off, is done again to associate each spiking neuron with a unique class label. Algorithm 2 describes the pseudo-code for the labeling process.

Line 2 loads the resulted network from the training process and line 3 turns STDP off so the network connections are not any longer modified. Line 4 creates a spiking counter to save each neuron's firing pattern. As in the training process, a FOR loop is

Algorithm 1. Training - Sample-Based Weight Initialization

```
1: trainingSet = load(MNIST-training)
2: spikingNetwork = CompetitiveSpikingNetwork()
3: spikingNetwork.STDPconnection[:, :] = CV2.blur(trainingSet[: M], (5,5))
4: spikeMonitor = SpikeCounter(spikingNetwork['Spiking'])
5: for iterator = m, m + 1, ..., P do
6:     normalizeSTDPConnection(78.0)
7:     spikingNetwork['Poisson'].rate = trainingSet.data[iterator]
8:     run(spikingNetwork, 350ms)
9: end for
10: saveSpikingNetwork(spikingNetwork)
```

Algorithm 2. Labeling

```
1: trainingSet = load(MNIST-training)
2: spikingNetwork = loadSpikingNetwork()
3: spikingNetwork.disableSTDP()
4: spikeMonitor = SpikeCounter(spikingNetwork['Spiking'])
5: for iterator = 1, 2, ..., P do
6:     spikingNetwork['Poisson'].rate = trainingSet.data[iterator]
7:     run(spikingNetwork, 350ms)
8: end for
9: labels = getSpikingNeuronLabels(spikeMonitor, trainigSet.labels)
10: saveLabels(labels)
```

used to present all training samples (lines 5 to 8) but the difference is that normalization is no longer needed since all connections are fixed. The spiking counter and the training labels are used to decide each neuron's label in line 9.

The already assigned labels are used to classify new unseen samples via a voting process, such as maximum, confidence, or distant-based [6,12,35]. Additionally, the firing pattern can be used directly for predictions through some decision function, which can be predefined [12,36], or learned by using the firing pattern matrix as input to any add-on machine learning classifier [31], such as a conventional neural network.

5 Experiments

5.1 Dataset

All experiments are performed on the MNIST dataset, which consists of 70,000 samples of hand written 28×28 pixel images divided into 60,000 samples for training and 10,000 samples for testing [20]. The raw images are first flattened (turned into column vectors) and scaled to the range from 0 to 63.75, and are used as input to the sensory layer to determine the firing rates of the Poisson units.

5.2 Experimental Settings

To analyze the performance of the sample-based initialization, three different experiments are performed. First, we compare the training convergence and testing accu-

racy of random initialization with our initialization method. Second, we evaluate our method's robustness using samples from only one class (from 10 total) as the initial weights. Third, we compare the prediction results of the CSNN with a fully supervised traditional neural network, using the same topology and number of neurons.

Two CSNNs with 400 spiking neurons are used: the state of the art CSNN [6], and another one simplified by us. The state of the art CSNN uses 784 sensory layer Poisson units, 400 Leaky Integrate and Fire neurons with conductance-based stimulation input to the spiking neuron layer, trace-based STDP, indirect inhibition, weight normalization, and resting period of 150 ms between each sample presentation. The simplified CSNN uses the same spiking neuron model, learning rule and weight normalization but differs in the use of direct inhibition, with no resting period, and a different value of the membrane constant time ($3 \cdot 10^6$ ms instead of $1 \cdot 10^7$ ms). Importantly, the simplified CSNN trains in half the time than the state of the art CSNN.

All simulations were carried out using the Python's Brian Simulator package on an Intel Core i9-9900K with 64GB RAM computer (the code is publicly available on GitHub).[1]

5.3 Evaluation Metrics

The training convergence and testing accuracy are used to evaluate all experiments. The training convergence is based on the number of samples needed to reach a stable state, which is defined as the number of samples needed to reach 80% accuracy. The training accuracy is calculated after every 1,000 sample presentations in a two step process. First, the neuron labels are assigned based on the maximum firing rate of the previous 1,000 samples. Second, the assigned labels are used to predict the classes for the next 1,000 samples using maximum voting.

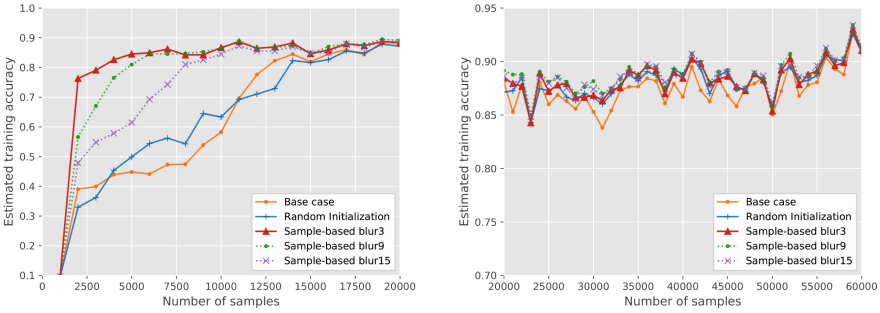
The testing accuracy for all 10,000 testing samples is calculated using three different methods: the maximum- and confidence-based voting, and using an add-on two layer neural network classifier. The latter uses 200 neurons with Relu activation in its first layer, 10 neurons with soft max activation in the output layer, dropout of 0.2 between the layers, and cross entropy loss function. All results are reported as average of 10 runs.

6 Results and Discussion

6.1 Convergence Time

The accuracies for 60K training samples using random initialization and the sample-based initialization are shown in Fig. 2. Figure 2a shows the accuracy on the first 20K sample presentations, and Fig. 2b shows the result on the next 40K samples. Five lines are plotted: one for the current state of the art CSNN with random initialization (Base case) [6], one for the simplified by us CSNN with random initialization (Random), and three for sample-based initialization with different degrees of image blurring. The plot starts after 1K iterations since we estimate the training accuracy using a 1K window.

[1] https://github.com/PaoloGCD/fastCSNN.



(a) Training 0-20K samples (b) Training 20K-60K samples

Fig. 2. Training accuracy vs. number of samples, using samples from all classes for initialization

The convergence time for our initialization method is faster than for random initialization (both base case and random). Specifically, using sample-based initialization with blurring of 3 achieves 80% accuracy using less than 3K samples. Bigger blurring factors reduce the convergence time (blurring of 9 and 15), but are still faster than the base case and the random initialization that need around 12.5K samples each to reach 80% accuracy. Blurring of 9 reaches 80% accuracy after around 5K samples and blurring of 15 after 8K samples.

Table 1 shows results using maximum and confidence voting, and using an add-on neural network classifier for classification prediction of the test set (trained on 60K samples). We see that sample-based initialization with blurring of 9 achieves the best accuracy in all three methods. Namely, it achieves 90.87%, 91.27% and 92.54% accuracies, which are higher than for the base case (88.89%, 90.37% and 91.73%), and higher than for random initialization (90.74%, 91.17% and 92.43%).

Table 1. Testing accuracy using different decision methods.

	Max voting	Confidence voting	Neural network
Base case	88.89 $\pm$ 0.44	90.37 $\pm$ 0.31	91.73 $\pm$ 0.16
Random	90.67 $\pm$ 0.19	91.14 $\pm$ 0.12	92.43 $\pm$ 0.06
Blur 3	90.53 $\pm$ 0.21	91.08 $\pm$ 0.16	92.37 $\pm$ 0.12
Blur 9	**90.87** $\pm$ 0.10	**91.27** $\pm$ 0.11	**92.54** $\pm$ 0.11
Blur 15	90.80 $\pm$ 0.17	91.18 $\pm$ 0.12	92.54 $\pm$ 0.12

Table 2 shows accuracy results on the testing set after training with 5K, 10K, 20K, 40K and 60K sample presentations, using maximum voting.

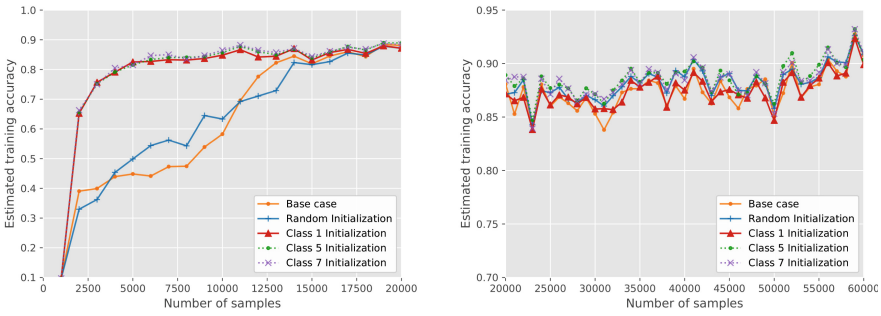
Before convergence (5K and 10K) sample-based initialization produces better results than random initialization. While at convergence (20K, 40K, and 60K) the results are about the same. The results with blurring of 9 are consistently the best in all cases.

Table 2. Testing accuracy for different number of training samples.

	5K	10K	20K	40K	60K
Base case	70.91 ± 0.71	83.37 ± 0.26	86.22 ± 0.62	87.54 ± 0.18	88.89 ± 0.44
Random	60.93 ± 0.54	82.11 ± 0.51	89.29 ± 0.25	90.66 ± 0.22	90.67 ± 0.19
Blur 3	86.65 ± 0.41	88.35 ± 0.53	89.17 ± 0.32	90.56 ± 0.23	90.53 ± 0.21
Blur 9	**87.61** ± 0.27	**88.81** ± 0.10	**89.73** ± 0.39	**90.90** ± 0.29	**90.87** ± 0.10
Blur 15	81.96 ± 0.40	88.31 ± 0.28	89.53 ± 0.39	90.84 ± 0.27	90.80 ± 0.17

6.2 Sample-Based Initialization Robustness

When training, often not all classes are seen in the first M samples, which are used to set the sample-based initial weights. Thus, we initialize the connection weights using samples of just one class (out of 10). Although, all classes were tested, we discuss here results only for training with classes 1, 5, and 7. Figure 3 shows training accuracies using initialization with these classes. The base and random cases are shown for reference using results from Fig. 2.



(a) Training 0-20K samples

(b) Training 20K-60K samples

Fig. 3. Training accuracy vs. number of samples, using samples from only one class for initialization

We see that the convergence times for all sample-based initialization cases are still faster than for random initializations even when only one class is used to initialize the connection values. All these cases reach 80% accuracy after 4.5K sample presentations, while random initialization reaches 80% accuracy after 12K samples.

Table 3 shows testing accuracy results for maximum and confidence voting and for the add-on neural network classifier.

Overall, sample-based initialization of class 5 achieved the best result for all three methods (91.06%, 91.41% and 92.66%), while class 1 initialization was the worst (89.85%, 90.52% and 91.86%), but is still higher than the base case. The variance in all cases is less than 0.5% which indicates consistency across all cases.

Table 3. Testing accuracy using samples from only one class for initialization.

	Max voting	Confidence voting	Neural network
Base case	88.89 ± 0.441	90.37 ± 0.308	91.73 ± 0.157
Random	90.67 ± 0.190	91.14 ± 0.115	92.43 ± 0.085
Class 1	89.85 ± 0.470	90.52 ± 0.335	91.86 ± 0.147
Class 5	**91.06** ± 0.095	**91.41** ± 0.152	**92.66** ± 0.122
Class 7	90.74 ± 0.258	91.23 ± 0.190	92.52 ± 0.087

6.3 CSNN and Fully Supervised Neural Network Comparison

We compared the CSNN's performance with a fully supervised classical neural network. Table 4 shows testing accuracy for two best CSNNs, namely, sample-based initialization using blurring of 5 (fastest convergence) and blurring of 9 (best accuracy). The used NN is a 3 layer feed-forward neural network with 400, 200, and 10 neurons.

Table 4. Testing accuracy, CSNN and Fully Supervised NN comparison.

	1 epoch	3 epochs	5 epochs	10 epochs
CSNN-blurr5-NN	92.26	92.52	92.57	92.41
CSNN-blurr9-NN	**92.55**	92.89	92.97	92.85
Fully supervised NN	91.81	**94.73**	**95.49**	**96.24**

Importantly, we observe that both CSNNs achieve better accuracy after just 1 epoch of training, which can be advantageous in many real world applications. The testing accuracy for the CSNN improves slightly after 3 and 5 epochs but starts decaying at 10 epochs.

7 Conclusions

In this paper we introduced a new initialization method that uses the training samples as initial values for the as connection weights, between the Poisson units and the spiking neurons in Competitive Spiking Neural Networks. This method reduces the amount of training samples needed to achieve convergence and increases accuracy. Specifically, it significantly reduced the convergence time to around 3K samples as compared with random initialization that needed around 12.5K samples on the MNIST dataset. It also achieved a slight increase of accuracy using maximum voting, confidence voting, as well as using an add-on neural network classifier. We also showed that the convergence time and accuracy gains are about the same regardless of the class distribution in the samples used to initialize the connection weights. Importantly, we compared the CSNN with a fully supervised feed forward neural network and have shown that it performed better for small number of sample presentations, which is a strongly desired characteristic for real world applications.

References

1. Bi, G., Poo, M.: Synaptic modification by correlated activity: Hebb's postulate revisited. Ann. Rev. Neurosci. **24**(1), 139–166 (2001). https://doi.org/10.1146/annurev.neuro.24.1.139. pMID: 11283308
2. Brette, R., Gerstner, W.: Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. J. Neurophysiol. **94**(5), 3637–3642 (2005). https://doi.org/10.1152/jn.00686.2005. pMID: 16014787
3. Cavallari, S., Panzeri, S., Mazzoni, A.: Comparison of the dynamics of neural interactions between current-based and conductance-based integrate-and-fire recurrent networks. Front. Neural Circuits **8**, 12 (2014). https://doi.org/10.3389/fncir.2014.00012. https://www.frontiersin.org/article/10.3389/fncir.2014.00012
4. Cessac, B., Viéville, T.: On dynamics of integrate-and-fire neural networks with conductance based synapses. Front. Comput. Neurosci. **2**, 2 (2008). https://doi.org/10.3389/neuro.10.002.2008. https://www.frontiersin.org/article/10.3389/neuro.10.002.2008
5. Ciregan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3642–3649, June 2012. https://doi.org/10.1109/CVPR.2012.6248110
6. Diehl, P., Cook, M.: Unsupervised learning of digit recognition using spike-timing-dependent plasticity. Front. Comput. Neurosci. **9**, 99 (2015). https://doi.org/10.3389/fncom.2015.00099. https://www.frontiersin.org/article/10.3389/fncom.2015.00099
7. Fukushima, K.: Cognitron: a self-organizing multilayered neural network. Biol. Cybern. **20**(3), 121–136 (1975). https://doi.org/10.1007/BF00342633
8. Furber, S.B., Galluppi, F., Temple, S., Plana, L.A.: The spinnaker project. Proc. IEEE **102**(5), 652–665 (2014). https://doi.org/10.1109/JPROC.2014.2304638
9. Gerstner, W., Kistler, W.M.: Mathematical formulations of Hebbian learning. Biol. Cybern. **87**(5), 404–415 (2002). https://doi.org/10.1007/s00422-002-0353-y
10. Gerstner, W., Kistler, W.M., Naud, R., Paninski, L.: Neuronal Dynamics: From Single Neurons to Networks and Models of Cognition. Cambridge University Press, New York (2014)
11. Goodfellow, I., et al.: Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems, vol. 27, pp. 2672–2680. Curran Associates, Inc. (2014). http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf
12. Hazan, H., Saunders, D., Sanghavi, D.T., Siegelmann, H., Kozma, R.: Unsupervised learning with self-organizing spiking neural networks. In: 2018 International Joint Conference on Neural Networks (IJCNN), July 2018. https://doi.org/10.1109/ijcnn.2018.8489673
13. Hebb, D.O.: The Organization of Behavior: A Neuropsychological Theory. Wiley/Chapman & Hall, Hoboken (1949)
14. Jia, Y., Huang, C., Darrell, T.: Beyond spatial pyramids: receptive field learning for pooled image features. In: 2012 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3370–3377, June 2012. https://doi.org/10.1109/CVPR.2012.6248076
15. Kandel, E.R., et al.: Principles of Neural Science, vol. 5. McGraw-hill, New York (2013)
16. Koch, C., Segev, I.: Methods in Neuronal Modeling: from Ions to Networks. MIT Press, Cambridge (1998)
17. Kohonen, T.: Self-organized formation of topologically correct feature maps. Biol. Cybern. **43**(1), 59–69 (1982). https://doi.org/10.1007/BF00337288
18. Konorski, J.: Conditioned Reflexes and Neuron Organization. Cambridge University Press, Cambridge (1948)
19. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015). https://doi.org/10.1038/nature14539

20. LeCun, Y., Cortes, C.: MNIST handwritten digit database (2010). http://yann.lecun.com/exdb/mnist/
21. Liang, Z., Schwartz, D., Ditzler, G., Koyluoglu, O.O.: The impact of encoding-decoding schemes and weight normalization in spiking neural networks. Neural Netw. **108**, 365–378 (2018). https://doi.org/10.1016/j.neunet.2018.08.024. http://www.sciencedirect.com/science/article/pii/S0893608018302508
22. Linde, Y., Buzo, A., Gray, R.: An algorithm for vector quantizer design. IEEE Trans. Commun. **28**(1), 84–95 (1980)
23. Maass, W.: Networks of spiking neurons: the third generation of neural network models. Neural Netw. **10**(9), 1659–1671 (1997). https://doi.org/10.1016/S0893-6080(97)00011-7. http://www.sciencedirect.com/science/article/pii/S0893608097000117
24. von der Malsburg, C.: Self-organization of orientation sensitive cells in the striate cortex. Kybernetik **14**(2), 85–100 (1973). https://doi.org/10.1007/BF00288907
25. McDonnell, M.D., Vladusich, T.: Enhanced image classification with a fast-learning shallow convolutional neural network. In: 2015 International Joint Conference on Neural Networks (IJCNN), pp. 1–7, July 2015. https://doi.org/10.1109/IJCNN.2015.7280796
26. Mishkin, D., Matas, J.: All you need is a good init. arXiv preprint arXiv:1511.06422 (2015)
27. Monroe, D.: Neuromorphic computing gets ready for the (really) big time. Commun. ACM **57**(6), 13–15 (2014). https://doi.org/10.1145/2601069
28. Morrison, A., Diesmann, M., Gerstner, W.: Phenomenological models of synaptic plasticity based on spike timing. Biol. Cybern. **98**(6), 459–478 (2008). https://doi.org/10.1007/s00422-008-0233-1. https://pubmed.ncbi.nlm.nih.gov/18491160. 18491160[pmid]
29. Nowlan, S.J.: Maximum likelihood competitive learning. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems, vol. 2, pp. 574–582. Morgan-Kaufmann (1990). http://papers.nips.cc/paper/225-maximum-likelihood-competitive-learning.pdf
30. Pfister, J.P., Gerstner, W.: Triplets of spikes in a model of spike timing-dependent plasticity. J. Neurosci. **26**(38), 9673–9682 (2006). https://doi.org/10.1523/JNEUROSCI.1425-06.2006
31. Querlioz, D., Bichler, O., Dollfus, P., Gamrat, C.: Immunity to device variations in a spiking neural network with memristive nanodevices. IEEE Trans. Nanotechnol. **12**(3), 288–295 (2013). https://doi.org/10.1109/TNANO.2013.2250995
32. Querlioz, D., Bichler, O., Gamrat, C.: Simulation of a memristor-based spiking neural network immune to device variations. In: The 2011 International Joint Conference on Neural Networks, pp. 1775–1781, July 2011. https://doi.org/10.1109/IJCNN.2011.6033439
33. Rumelhart, D.E., Zipser, D.: Feature discovery by competitive learning. Cognit. Sci. **9**(1), 75–112 (1985). https://doi.org/10.1016/S0364-0213(85)80010-0. http://www.sciencedirect.com/science/article/pii/S0364021385800100
34. Sato, I., Nishimura, H., Yokoi, K.: APAC: augmented pattern classification with neural networks. arXiv preprint arXiv:1505.03229 (2015)
35. Saunders, D.J., Siegelmann, H.T., Kozma, R., et al.: STDP learning of image patches with convolutional spiking neural networks. In: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–7. IEEE (2018). https://doi.org/10.1109/IJCNN.2018.8489684
36. Shin, J., et al.: Recognition of partially occluded and rotated images with a network of spiking neurons. IEEE Trans. Neural Netw. **21**(11), 1697–1709 (2010). https://doi.org/10.1109/TNN.2010.2050600
37. Sjöström, P.J., Rancz, E.A., Roth, A., Häusser, M.: Dendritic excitability and synaptic plasticity. Physiol. Rev. **88**(2), 769–840 (2008). https://doi.org/10.1152/physrev.00016.2007. pMID: 18391179
38. Wan, L., Zeiler, M., Zhang, S., Cun, Y.L., Fergus, R.: Regularization of neural networks using dropconnect. In: Dasgupta, S., McAllester, D. (eds.) Proceedings of the 30th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 28, pp. 1058–1066. PMLR, Atlanta, June 2013. http://proceedings.mlr.press/v28/wan13.html

39. Wicramasinghe, C.S., Amarasinghe, K., Manic, M.: Deep self-organizing maps for unsupervised image classification. IEEE Trans. Ind. Inf. **15**(11), 5837–5845 (2019). https://doi.org/10.1109/TII.2019.2906083
40. Zierenberg, J., Wilting, J., Priesemann, V.: Homeostatic plasticity and external input shape neural network dynamics. Phys. Rev. X **8**, 031018 (2018). https://doi.org/10.1103/PhysRevX.8.031018