Supplementary Material

Gaspard Goupy[1], Pierre Tirilly[1], and Ioan Marius Bilasco[1]

[1]Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

1 Overall Algorithm

We present, in Algorithm 1, the pseudo-code for training a spiking classification layer with S2-STDP+NCG. We omitted early stopping for clarity, as well as how we handle intra-class inhibition if several neurons fire at the same time.

Algorithm 1 Training of a spiking classification layer with S2-STDP+NCG.

1: **Input:**
2: Number of samples $D \in \mathbb{Z}^+$;
3: Number of input neurons $F \in \mathbb{Z}^+$;
4: Number of output neurons $N \in \mathbb{Z}^+$;
5: Number of neurons per class $M \in \mathbb{Z}^+$;
6: Number of classes $C \in \mathbb{Z}^+$;
7: Firing threshold $V_{\text{th}} \in \mathbb{R}$;
8: Maximum firing time $T_{\max} \in \mathbb{R}^+$;
9: Initial weight normal distribution of mean $\mu \in \mathbb{R}$ and std $\sigma \in \mathbb{R}_0^+$;
10: Minimum and maximum weight value $w_{\min} \in \mathbb{R}$ and $w_{\max} \in \mathbb{R}$;
11: S2-STDP time gap $g \in \mathbb{R}^+$;
12: STDP learning rates $A^+ \in \mathbb{R}_0^+$ and $A^- \in \mathbb{R}_0^-$;
13: STDP annealing $\beta \in \mathbb{R}_0^+$;
14: Threshold learning rate $\eta_{\text{th}} \in \mathbb{R}_0^+$;
15: Threshold annealing $\beta_{\text{th}} \in \mathbb{R}_0^+$;
16: Matrix of spike-encoded extracted features $X \in [0, T_{\max}]^{D \times F}$;
17: Vector of class labels $Y \in \mathbb{Z}^D$.
18: **Output:** Matrix of trained weights $W \in \mathbb{R}^{F \times N}$.
19: Define C NCGs of M neurons
20: For each NCG, label $M-1$ neurons as *target* and 1 as *non-target*
21: Map each neuron n_j to a class $c_j \in \mathbb{Z}$ based on its NCG (j from 1 to N)
22: Initialize matrix of weights $W \sim \mathcal{N}(\mu, \sigma^2)$
23: Initialize vector of test thresholds $\theta \in \mathbb{R}^N, \theta_j := V_{th}$ (j from 1 to N)
24: Initialize vector of training thresholds $\theta' \in \mathbb{R}^N, \theta'_j := V_{th}$ (j from 1 to N)
25: Compute vector of weight normalization factors $\overline{w} \in \mathbb{R}^N, \overline{w}_j := \sum_i W_{ij}$ (j from 1 to N)
26: **for each** epoch **do**
27:     Reset θ'_j to θ_j (j from 1 to N)   ▷ Reset training thresholds to test thresholds
28:     **for each** x, y **in** X, Y **do**   ▷ For each sample of data x and class y

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

29:       **// Forward propagation**

30:       Convert $x \in [0, T_{\max}]^F$ into an ordered vector of input spike times $t^I \in [0, T_{\max}]^F$

31:       Initialize vector of output spike times $t^O \in [0, T_{\max}]^N$, $t_j^O := T_{\max}$ (j from 1 to N)

32:       $// T_{\max} = no\ spike$

33:       Initialize vector of membrane potentials $V \in \mathbb{R}^N$, $V_j := 0$ (j from 1 to N)

34:       Initialize vector of active flags $A \in \{0, 1\}^N$, $A_j := 1$ (j from 1 to N)

35:       **for each** $t_k^I \in t^I$ **do**   ▷ For each ordered input spike time

36:          **for each** n_j **where** $A_j = 1$ **do**   ▷ For each active neuron

37:             $V_j := V_j + W_{kj}$ ▷ Integrate input spike, Eq. 1 in main

38:             **if** $V_j \geq \left(\theta_j \text{ if } c_j \neq y \text{ else } \theta'_j\right)$ **then**   ▷ Threshold depends on sample class

39:                $t_j^O := t_k^I$ ▷ n_j fires at t_k^I

40:                $A_p := 0$, **for each** n_p **in** the NCG of n_j ▷ Intra-class inhibition

41:             **end if**

42:          **end for**

43:       **end for**

44:       **// Weights and thresholds update**

45:       $\overline{T} := \text{mean}\left\{t \in t^O | t < T_{\max}\right\}$, $\overline{T} \in \mathbb{R}$ ▷ Average firing time of non-inhibited neurons

46:       **for each** NCG **do**   ▷ One update per NCG

47:          Get the first n_{j*} to fire within the NCG   ▷ Non-inhibited neuron

48:          **if** n_{j*} **is** *target* **and** $c_{j*} = y$ **then**   ▷ Target update

49:             $e_{j*} := t_{j*}^O - \left(\overline{T} - \frac{C-1}{C}g\right)$, $e_{j*} \in \mathbb{R}$ ▷ Eq. 5 in main

50:             **for each** n_p **in** the NCG **where** n_p **is** *target* **do**   ▷ **Competition regulation**

51:                **if** $p = j*$ **then**   ▷ Non-inhibited target neuron

52:                    $\theta'_p := \max\left\{\theta_p, \theta'_p + \eta_{\text{th}} \cdot \frac{M-2}{M-1}\right\}$ ▷ Eq. 6 in main

53:                **else**   ▷ Inhibited target neurons

54:                    $\theta'_p := \max\left\{\theta_p, \theta'_p - \eta_{\text{th}} \cdot \frac{1}{M-1}\right\}$ ▷ Eq. 6 in main

55:                **end if**

56:             **end for**

57:          **else**   ▷ Non-target update

58:             $e_{j*} := t_{j*}^O - \left(\overline{T} + \frac{1}{C}g\right)$, $e_{j*} \in \mathbb{R}$ ▷ Eq. 5 in main

59:          **end if**

60:          **for** $k \in [1, F]$ **do**   ▷ Update weights of n_{j*} with error-modulated STDP

61:             **if** $t_k^I \leq t_{j*}^O$ **then**   ▷ Long-term potentiation

62:                $W_{kj*} := W_{kj*} + e_{j*} \cdot A^+$ ▷ Eq. 3 in main

63:             **else**   ▷ Long-term depression

64:                $W_{kj*} := W_{kj*} + e_{j*} \cdot A^-$ ▷ Eq. 3 in main

65:             **end if**

66:          **end for**

67:          $W_{kj*} := W_{kj*} \cdot \frac{\overline{w}_{j*}}{\sum_x |W_{xj*}|}$ (k from 1 to F)   ▷ Weight normalization

68:          Clip weights of n_{j*} in $[w_{\min}, w_{\max}]$

69:       **end for**

70:       **end for**

71:       $A^+ := A^+ \cdot \beta$, $A^- := A^- \cdot \beta$ ▷ STDP learning rate annealing

72:       $\eta_{\text{th}} := \eta_{\text{th}} \cdot \beta_{\text{th}}$ ▷ Threshold learning rate annealing

73: **end for**

2

2 Experimental Details

2.1 Classification Pipeline

We present, in Figure 1, the classification pipeline used in our experiments. Our classification system consists of an unsupervised feature extractor followed by a spiking classification layer.

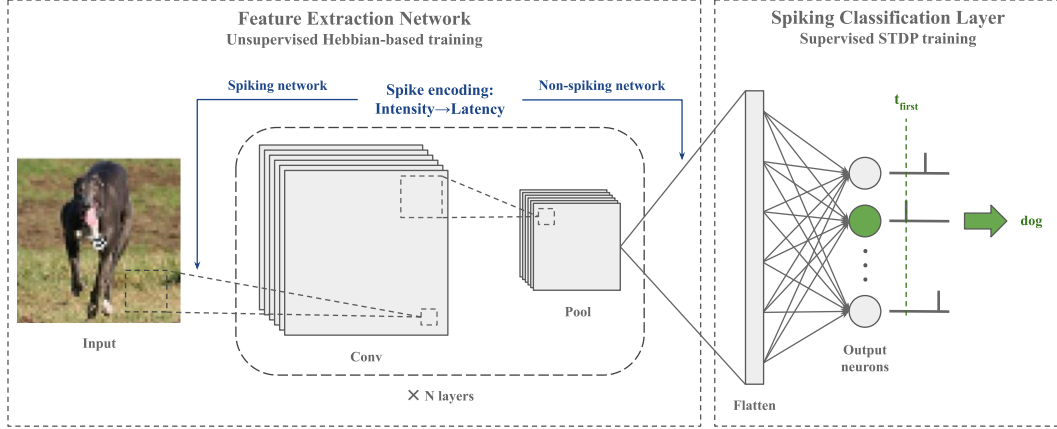



Figure 1: Pipeline of our classification system. Training is performed in a layer-wise fashion. First, the convolutional layers of the feature extractor are trained on the input images using an unsupervised Hebbian-based learning rule. Then, a fully-connected spiking classification layer is trained on the extracted features using a supervised STDP rule. Spike encoding occurs before feature extraction for spike-based feature extractors or afterward for non-spike-based extractors.

2.2 Unsupervised Feature Extractors

STDP-CSNN comprises a spiking convolutional layer trained with STDP and a spiking max-pooling layer. First, images are preprocessed with on-center/off-center coding [1] for MNIST and Fashion-MNIST, and with hardware-friendly whitening [2] for CIFAR-10/100. Then, preprocessed images are encoded into spikes with latency coding [3] (a form of first-spike coding) and transmitted to STDP-CSNN for unsupervised training. Latency coding converts each normalized pixel x into a single spike timestamp as follows: $T(x) = 1 - x$. Each output feature consists of a floating-point spike timestamp in $[0, 1]$, which is inherently compatible with the classification layer. We use the hyperparameters reported in [4]. The base code is available at `https://gitlab.univ-lille.fr/bioinsp/falez-csnn-simulator`, under the CeCILL-B license.

SoftHebb-CNN comprises three convolutional layers trained with SoftHebb. Each convolutional layer includes a succession of batch normalization, convolution, pooling (max-pooling for the first two layers, average-pooling for the last one), and a Triangle [5] activation function. Images undergo no preprocessing. The extracted output features of each sample are rescaled in $[0, 1]$ and encoded into spike timestamps with latency coding (see supra). For all the datasets, we use the hyperparameters employed for the CIFAR-10 task in the original paper [6]. The base code is available at `https://github.com/NeuromorphicComputing/SoftHebb`. No license information is available.

2.3 Spiking Classification Layers

The maximum firing time in the layer is $T_{\max} = 1$, which corresponds to the maximum possible firing timestamp from input neurons. We initialize weights with a normal distribution of mean μ and standard deviation σ, and we clip them in $[w_{\min}, w_{\max}]$ after each update. For SSTDP and S2-STDP models, we use values from [4]: $\mu = 0.5$ for SSTDP and $\mu = 0.3$ for S2-STDP, $\sigma = 0.01$, $w_{\min} = 0$, and $w_{\max} = 1$. For R-STDP models, we use values from [7]: $\mu = 0.8$, $\sigma = 0.01$, $w_{\min} = 0.2$, and $w_{\max} = 0.8$. Weight normalization is employed with S2-STDP models [4] to keep a similar weight average (of $\mu = 0.3$) across neurons during learning. We decrease the learning rates after each epoch by a factor $\beta = 0.98$. All the other hyperparameters (including firing threshold,

3

learning rates, and method-specific hyperparameters) are optimized with gridsearch. To ensure a fair comparison between methods, we performed extensive gridsearch optimization (from 600 to 1,440 runs, depending on the number of hyperparameters) for each model (except on CIFAR-100 where we used hyperparameters optimized on CIFAR-10). Optimized values and gridsearch ranges are located in the `config/` folder of our source code.

2.4 Computing Resources

We conducted our experiments on private servers as well as the Grid'5000 testbed [8], providing clusters of servers. Since our models rely on floating-point values to represent firing timestamps, we cannot benefit from GPU parallelization of operations. Therefore, we exclusively used CPUs, primarily Intel Xeon W-2245, Intel Xeon Gold 5218, Intel Xeon Gold 6126, Intel Xeon Gold 6130, Intel Xeon Platinum 8358, AMD EPYC 7343. The servers ranged from 16 to 128 CPU cores and 64 to 768 GiB of RAM. The entire experimentation process spanned over five months and resulted in over 70,000 runs (i.e. training of a single model), including gridsearch runs, K-fold runs, and unreported experiments. The average duration of a single run varies depending on several factors, such as the number of input features and the dataset. On CIFAR-10, training a classification layer with S2-STDP+NCG (total of 50 output neurons) using pre-extracted features requires approximately 1 hour with STDP-CSNN (for ~ 1.4 GiB of RAM) and 4 hours with SoftHebb-CNN (for ~ 8.9 GiB of RAM). On CIFAR-100, the same training (total of 500 neurons, as ten times more classes) requires approximately 8 hours with STDP-CSNN (for ~ 1.3 GiB of RAM) and 48 hours with SoftHebb-CNN (for ~ 8.7 GiB of RAM). Note that the amount of RAM needed is mainly influenced by the number of input features. Also, the computational overhead of NCGs scales linearly with both the number of neurons and the number of input features.

2.5 Datasets

MNIST is available at `http://yann.lecun.com/exdb/mnist/` under the CC BY-SA 3.0 license. Fashion-MNIST is available at `https://github.com/zalandoresearch/fashion-mnist` under the MIT license. CIFAR-10 and CIFAR-100 are available at `https://www.cs.toronto.edu/~kriz/cifar.html` under the MIT license.

3 Additional Experiments

3.1 Impact of the Non-Target Neuron

Table 1: Accuracy of S2-STDP+NCG with various numbers of target and non-target neurons.

Dataset	Non-target neurons	Target neurons	Accuracy (Mean±Std %)	
			STDP-CSNN	SoftHebb-CNN
MNIST	1	1	98.62 ± 0.07	$\mathbf{99.18 \pm 0.05}$
	0	4	98.79 ± 0.07	99.05 ± 0.07
	0	5	98.82 ± 0.05	99.08 ± 0.08
	1	4	$\mathbf{98.92 \pm 0.07}$	99.17 ± 0.07
Fashion-MNIST	1	1	87.45 ± 0.16	91.33 ± 0.21
	0	4	87.73 ± 0.24	91.34 ± 0.17
	0	5	87.76 ± 0.16	91.33 ± 0.22
	1	4	$\mathbf{88.72 \pm 0.23}$	$\mathbf{91.86 \pm 0.14}$
CIFAR-10	1	1	62.94 ± 0.17	78.33 ± 0.18
	0	4	64.85 ± 0.16	78.54 ± 0.26
	0	5	65.51 ± 0.26	78.78 ± 0.15
	1	4	$\mathbf{66.41 \pm 0.17}$	$\mathbf{79.55 \pm 0.23}$

In [4], the accuracy of a classification layer trained with S2-STDP is improved by using two neurons per class with intra-class WTA competition. The authors show that this accuracy gain is attributed

4

to implicit neuron specialization toward target or non-target samples. In S2-STDP+NCG models, we take advantage of this mechanism by explicitly labeling one neuron of each NCG as non-target, whereas the others are labeled as target. To evaluate the impact of such labeling, we compare, in Table 1, the accuracy of S2-STDP+NCG with various numbers of target and non-target neurons. When using one target and one non-target neuron (similarly to [4] but with explicit labeling), the classification layer cannot learn various patterns per class but the accuracy is still improved through neuron specialization. We tend to obtain similar or better accuracies with multiple target neurons and no non-target neurons by learning various patterns per class. When combining multiple target neurons and a non-target neuron, we reach optimal performance. Preliminary experiments with multiple non-target neurons and competition regulation did not yield improvements in accuracy. We believe that a single non-target neuron is enough to benefit from the specialization effect.

3.2 Impact of the Number of Neurons

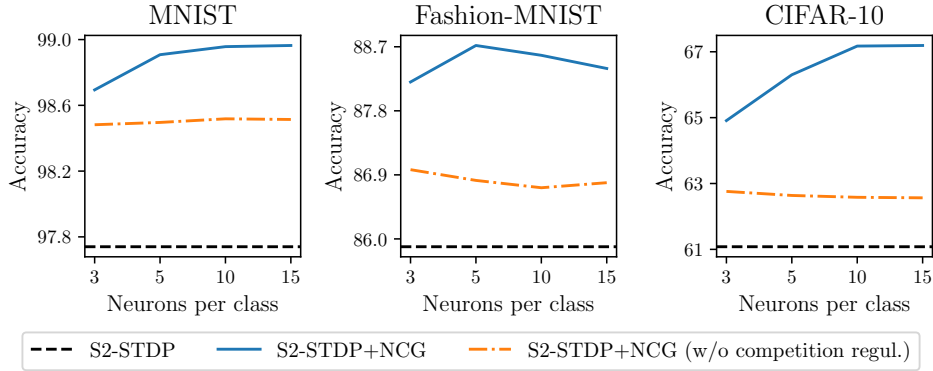



Figure 2: Accuracy against the number of neurons per class in the classification layer trained with S2-STDP+NCG. The features are extracted with STDP-CSNN.

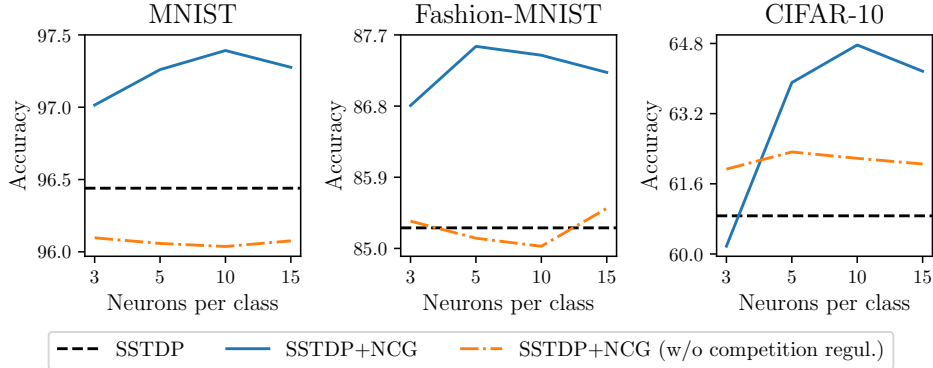



Figure 3: Accuracy against the number of neurons per class in the classification layer trained with SSTDP+NCG. The features are extracted with STDP-CSNN.

We analyze the impact of varying the number of neurons per class (M) on the performance of NCGs. Figures 2 and 3 present the accuracy achieved by S2-STDP+NCG and SSTDP+NCG using different numbers of neurons, with and without competition regulation. First, competition regulation is crucial to maximize performance with NCGs, especially for complex datasets. With competition regulation, increasing the number of neurons per class does not always result in better performance. For simpler datasets (MNIST, Fashion-MNIST), this increase often leads to either a plateau in accuracy or a decline when $M > 5$. Conversely, for harder datasets (CIFAR-10), using larger values ($M = 10$) improves the results from the main paper (which uses $M = 5$). Hence, the optimal number of neurons per class depends on the complexity of the task and intra-class pattern variations. It should be optimized as a hyperparameter. It is important to recall that the number of parameters in the

5

classification layer scales with the number of neurons. Hence, NCGs improve the performance of a classification layer at the cost of an increased number of parameters. Figures 2 and 3 show that significant accuracy gains can usually be achieved with only three neurons per class. This reveals that our method remains effective even with a minimal number of additional parameters. When selecting the value of M, the tradeoff between optimizing accuracy and minimizing parameter cost must be considered.

3.3 Impact of Hyperparameters

We study the impact of the threshold learning rate (η_{th}) and the threshold annealing (β_{th}), introduced by our competition regulation mechanism, on the accuracy. Figures 4 and 5 provide results for S2-STDP+NCG with STDP-CSNN and SoftHebb-CNN, respectively. Figures 6 and 7 provide results for SSTDP+NCG with STDP-CSNN and SoftHebb-CNN, respectively. Without annealing ($\beta_{th} = 1$), selecting an appropriate learning rate is crucial to obtain optimal performance. This is attributed to the increased difficulty in achieving convergence when employing higher learning rates that remain fixed during the entire training process. With annealing ($\beta_{th} < 1$), the performance of S2-STDP+NCG tends to be near-optimal regardless of the selected learning rate and annealing values. SSTDP+NCG is less robust to hyperparameters and requires more tuning to achieve optimal performance. This behavior may occur because SSTDP without NCG also exhibits lower hyperparameter robustness compared to S2-STDP [4]. However, the accuracy of SSTDP is still improved regardless of the hyperparameter values (when $\beta_{th} < 1$). Overall, annealing the threshold learning rate is not essential for benefiting from balanced competition, but it does offer better robustness against the selected value and can further improve performance.

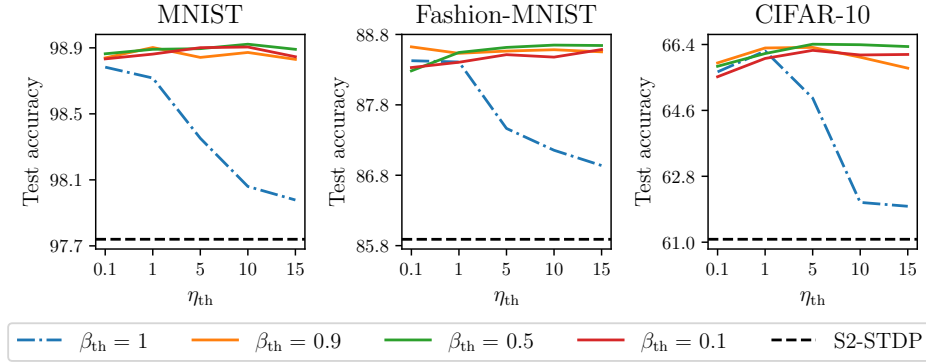



Figure 4: Accuracy of S2-STDP+NCG with different threshold annealing β_{th} against the threshold learning rate η_{th}. The features are extracted with STDP-CSNN.

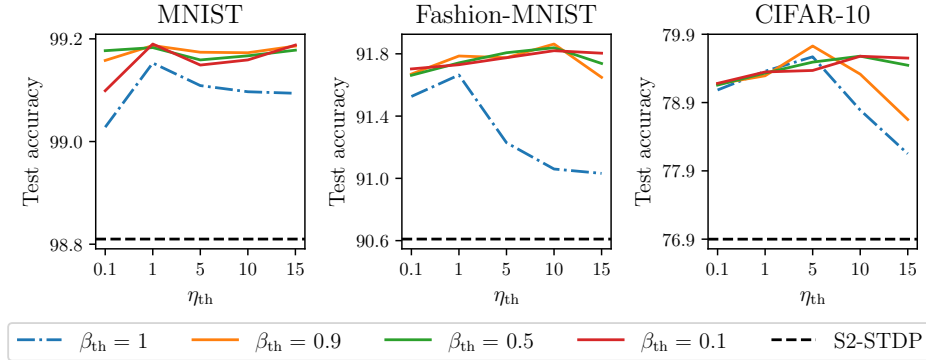



Figure 5: Accuracy of S2-STDP+NCG with different threshold annealing β_{th} against the threshold learning rate η_{th}. The features are extracted with SoftHebb-CNN.

6

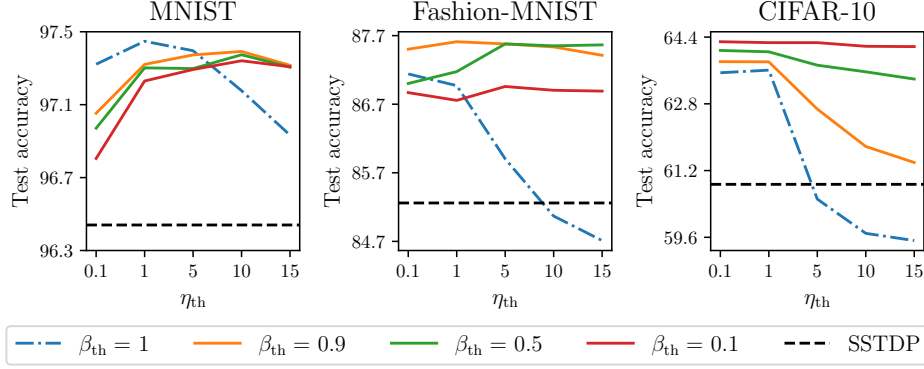



Figure 6: Accuracy of SSTDP+NCG with different threshold annealing β_{th} against the threshold learning rate η_{th}. The features are extracted with STDP-CSNN.

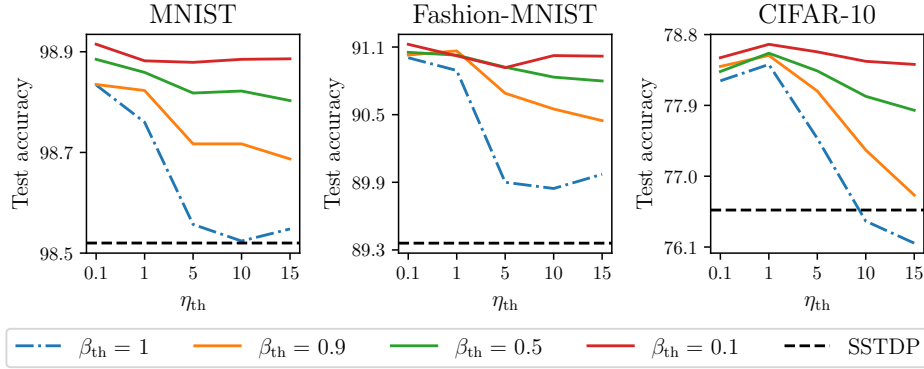



Figure 7: Accuracy of SSTDP+NCG with different threshold annealing β_{th} against the threshold learning rate η_{th}. The features are extracted with SoftHebb-CNN.

3.4 Ablation Study

Table 2: Ablation study on SSTDP+NCG. *M* is the number of neurons per class, *CR* is competition regulation with 1 or 2 thresholds, and *Drop* is dropout.

(a) Fashion-MNIST

Method	Accuracy (Mean±Std %)	
	STDP-CSNN	SoftHebb-CNN
M-1	85.26 ± 0.17	89.36 ± 0.24
M-5	85.13 ± 1.20	91.02 ± 0.14
M-5+CR-1	86.79 ± 0.14	90.92 ± 0.22
M-5+CR-2	$\mathbf{87.59 \pm 0.11}$	$\mathbf{91.06 \pm 0.10}$
M-5+Drop	86.40 ± 0.27	90.90 ± 0.14

(b) CIFAR-10

Method	Accuracy (Mean±Std %)	
	STDP-CSNN	SoftHebb-CNN
M-1	60.87 ± 0.53	76.57 ± 0.58
M-5	62.33 ± 0.14	77.51 ± 0.23
M-5+CR-1	63.50 ± 0.28	78.09 ± 0.34
M-5+CR-2	$\mathbf{64.05 \pm 0.48}$	$\mathbf{78.53 \pm 0.32}$
M-5+Drop	62.46 ± 0.19	78.12 ± 0.22

In the main paper, we conduct an ablation study on S2-STDP+NCG to show the individual accuracy gain brought by each component of our methods. We provide, in Table 2, a similar study on SSTDP+NCG. *M-1* and *M-5* represent SSTDP+NCG with $M = 1$ (one neuron per class, which is similar to SSTDP) and $M = 5$, without competition regulation. *CR-1* denotes our competition regulation mechanism with a single threshold per neuron (i.e. $\theta' = \theta$). Thresholds are not clipped nor reset between epochs, and the learned values are used for inference. *CR-2* denotes our competition regulation with two-compartment thresholds. *Drop* is dropout on the output neurons (an alternative competition regulation mechanism used with R-STDP). Neuron labeling is not employed because

7

SSTDP uses error clipping, and neuron specialization is only relevant without error clipping [4]. Overall, these results are consistent with the study conducted on S2-STDP+NCG. Competition regulation improves class separation (cf. *M-5*), except on Fashion-MNIST with SoftHebb-CNN, where the accuracy improvement is solely attributed to intra-class WTA. Our competition regulation mechanism (cf. *CR-2*) outperforms the existing threshold adaptation with one threshold (cf. *CR-1*), as well as dropout (cf. *Drop*).

3.5 Impact of Competition Regulation

(a) **w/o** competition regulation

(b) **w/** competition regulation

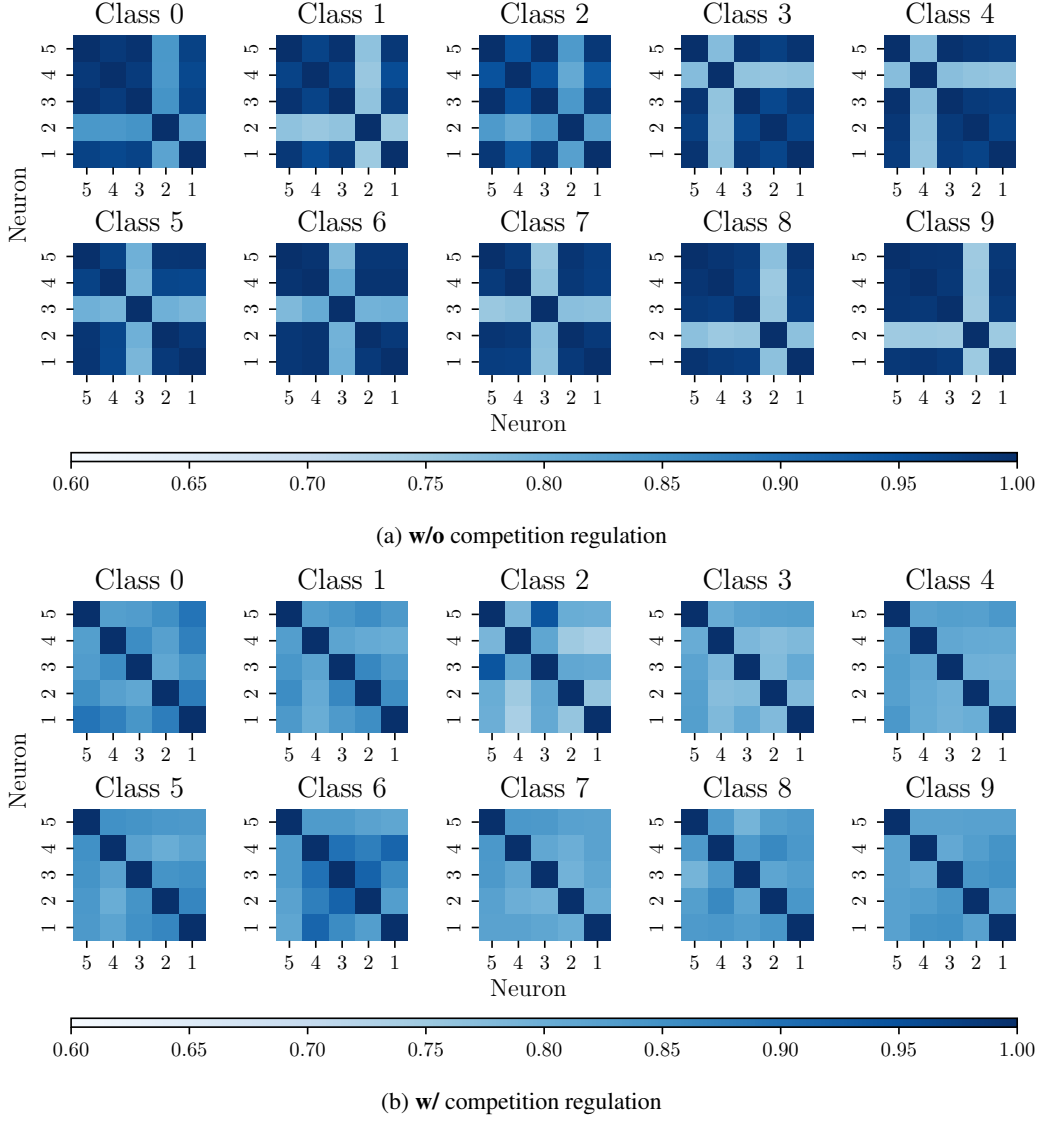



Figure 8: Heatmap of intra-class weight cosine similarities, after S2-STDP+NCG training on CIFAR-10. The features are extracted with STDP-CSNN.

In the main paper, we suggest through t-SNE visualizations that competition regulation enables the learning of various patterns per class. In this section, we further analyze the weights learned with and without competition regulation to support this claim. Figure 8 illustrates intra-class weight cosine similarities, after S2-STDP+NCG training on CIFAR-10. Without competition regulation, the neuron with the lowest score (such as neuron 2 of class 0) is the neuron receiving the majority of the target updates during training (see Figure 9a). All the other neurons exhibit high scores, indicating significant similarity in their weights. Competition regulation successfully decreases the similarity

8

score between neurons of each NCG (i.e. class), which provides further evidence that it enables the learning of various class-specific patterns.

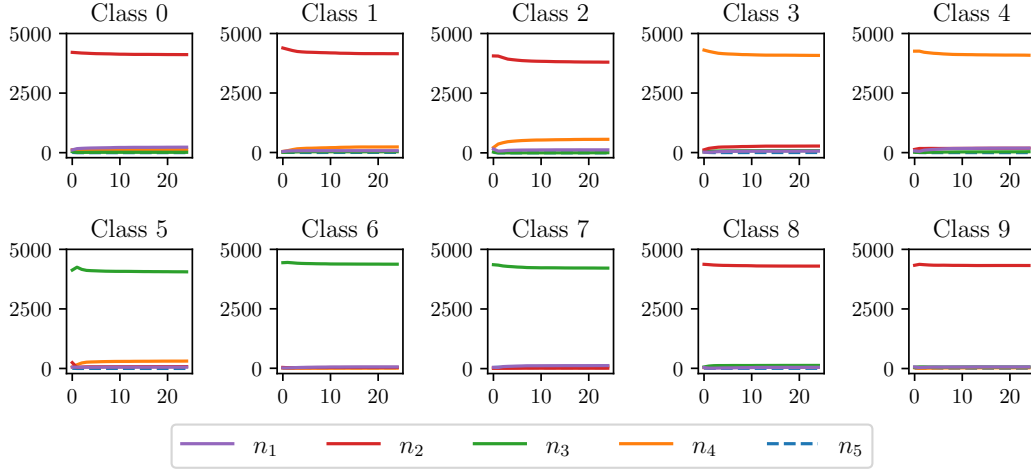
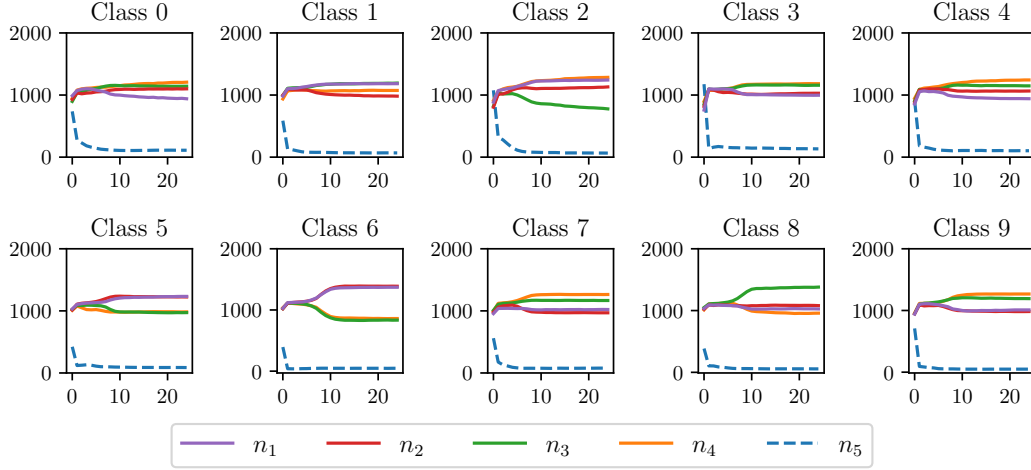
We present additional figures to show that competition regulation is crucial for ensuring balanced competition. Figures 9 and 10 illustrate the number of target weight updates received by neurons trained with S2-STDP+NCG on CIFAR-10 and Fashion-MNIST, respectively. Figures 11 and 12 present similar plots with SSTDP+NCG. Without competition regulation, one neuron of each NCG (i.e. class) receives the majority of the target updates. Our competition regulation mechanism successfully balances the number of updates received by neurons, which promotes, through intra-class WTA, the learning of various patterns per class. Note that competition for some classes is not completely balanced, particularly on Fashion-MNIST. However, absolute balance in the competition is not expected, given that the training set does not necessarily ensure equal representation of all different patterns.

(a) **w/o** competition regulation

(b) **w/** competition regulation

Figure 9: Number of target weight updates per epoch received by neurons of each class on CIFAR-10, with S2-STDP+NCG training. n_1 to n_4 are labeled as target neurons and n_5 is labeled as non-target. The features are extracted with STDP-CSNN.

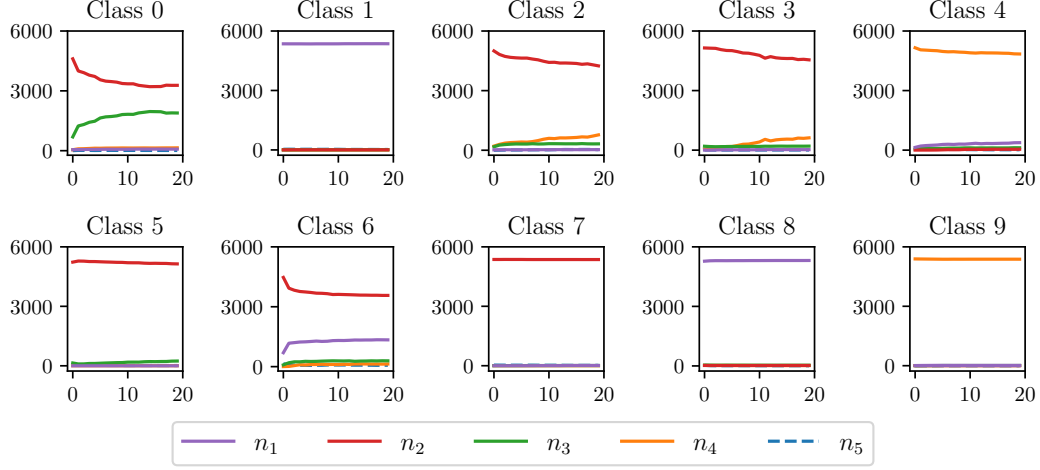
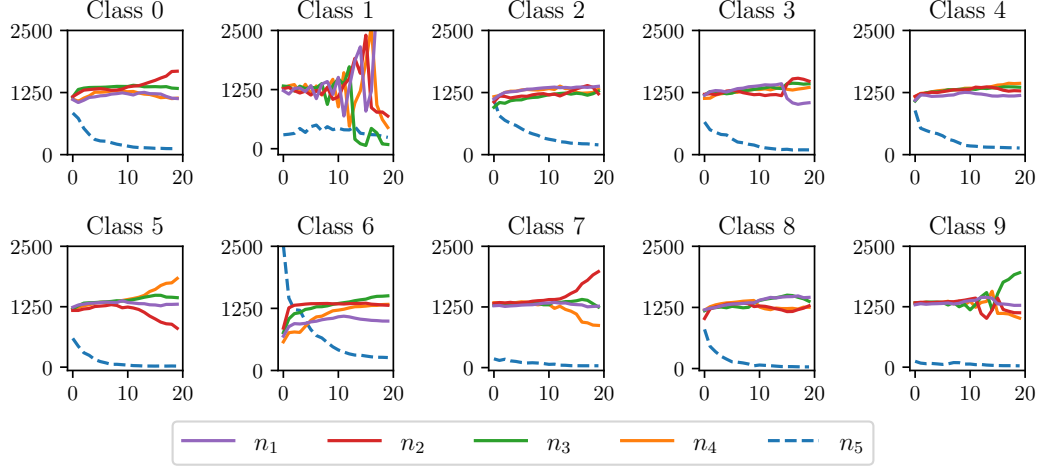
9

(a) **w/o** competition regulation

(b) **w/** competition regulation

Figure 10: Number of target weight updates per epoch received by neurons of each class on Fashion-MNIST, with S2-STDP+NCG training. n_1 to n_4 are labeled as target neurons and n_5 is labeled as non-target. The features are extracted with STDP-CSNN.

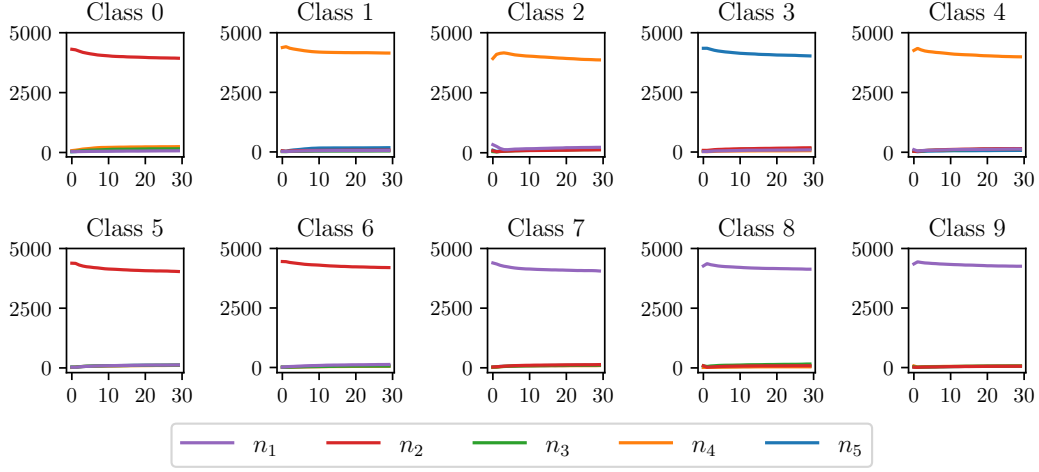
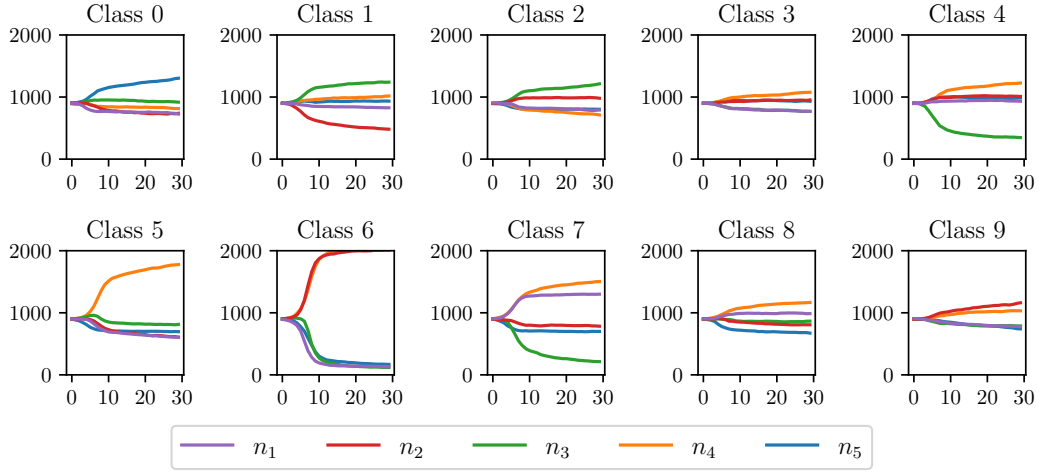
10

(a) **w/o** competition regulation

(b) **w/** competition regulation

Figure 11: Number of target weight updates per epoch received by neurons of each class on CIFAR-10, with SSTDP+NCG training. The features are extracted with STDP-CSNN.

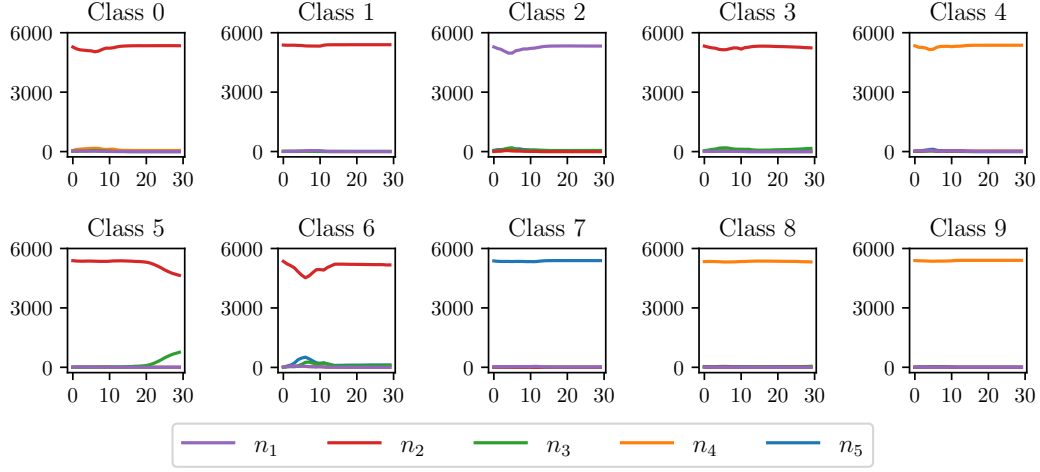
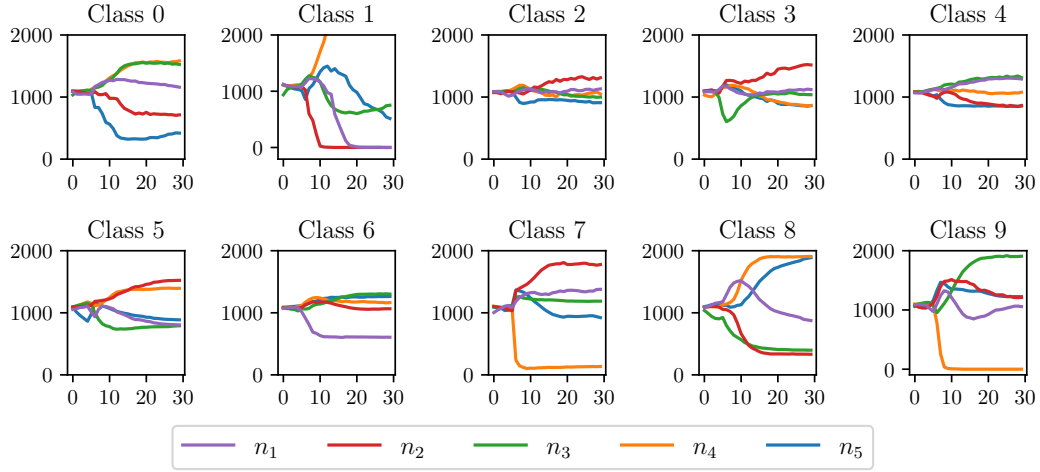
11

(a) **w/o** competition regulation

(b) **w/** competition regulation

Figure 12: Number of target weight updates per epoch received by neurons of each class on Fashion-MNIST, with SSTDP+NCG training. The features are extracted with STDP-CSNN.

12

3.6 Application to Gradient-Based Learning Rules

Table 3: Accuracy of S2-STDP+NCG, with and without long-term depression.

Dataset	Long-term depression	Accuracy (Mean±Std %)	
		STDP-CSNN	SoftHebb-CNN
MNIST	no	98.71 ± 0.14	99.15 ± 0.05
	yes	$\mathbf{98.92 \pm 0.07}$	$\mathbf{99.17 \pm 0.07}$
Fashion-MNIST	no	88.36 ± 0.12	91.54 ± 0.21
	yes	$\mathbf{88.72 \pm 0.23}$	$\mathbf{91.86 \pm 0.14}$
CIFAR-10	no	65.91 ± 0.38	$\mathbf{79.57 \pm 0.24}$
	yes	$\mathbf{66.41 \pm 0.17}$	79.55 ± 0.23

To train the spiking classification layers, we employ an error-modulated additive STDP in which the weight change is the product of the error and the learning rate (see Equation 3, page 4, in the main paper). The learning rate is positive for long-term potentiation (i.e. when the input neuron fires before the output neuron) and negative for long-term depression (i.e. when the input neuron fires after the output neuron). Given the simplicity of this STDP model, the weight updates, if long-term depression is ignored, resemble a gradient-based rule or delta rule[1]. To better understand the relevance of STDP with respect to these types of rules, we examine, in Table 3, the accuracy of S2-STDP+NCG with and without long-term depression. Results indicate that incorporating long-term depression generally leads to a slight improvement in accuracy. This improvement may be due to STDP considering input spikes that reach the neuron both before and after the output spike. In addition, long-term depression enables faster training convergence by increasing the number of weight updates per sample. The number of epochs with long-term depression is reduced by an average of 15% for STDP-CSNN and 4% for SoftHebb-CNN. As a result, S2-STDP enables more effective training of the NCGs than a gradient-based rule that uses a squared error loss function and the same method for defining the desired firing times. However, it should be noted that the method for computing the errors (specifically, the desired firing times) is more important than the method for updating the weights (gradient-based or STDP-based).

Table 4: Accuracy of spiking classification layers trained with a gradient-based method, on top of Hebbian-based unsupervised feature extractors.

Dataset	Method	Neurons per class	Accuracy (Mean±Std %)	
			STDP-CSNN	SoftHebb-CNN
MNIST	S4NN	1	97.86 ± 0.11	98.86 ± 0.14
	S4NN+NCG *(ours)*	5	$\mathbf{98.33 \pm 0.10}$	$\mathbf{98.99 \pm 0.09}$
Fashion-MNIST	S4NN	1	86.55 ± 0.13	90.38 ± 0.32
	S4NN+NCG *(ours)*	5	$\mathbf{87.87 \pm 0.24}$	$\mathbf{90.98 \pm 0.14}$
CIFAR-10	S4NN	1	57.86 ± 0.15	76.92 ± 0.26
	S4NN+NCG *(ours)*	5	$\mathbf{62.55 \pm 0.20}$	$\mathbf{77.62 \pm 0.27}$
CIFAR-100	S4NN	1	24.38 ± 0.44	38.51 ± 1.00
	S4NN+NCG *(ours)*	5	$\mathbf{28.68 \pm 0.61}$	$\mathbf{39.84 \pm 0.54}$

To provide additional evidence that NCGs can be trained with gradient-based rules, we conducted another experiment using S4NN [9], an established rule for single-spike neurons. S4NN uses the stochastic gradient descent algorithm (with gradient approximations) to minimize a squared error loss function, and computes the desired firing times based on the first output firing time. Table 4 presents the accuracy achieved by this rule, with and without NCGs, across various datasets and

---

[1]The delta learning rule must employ the same method for computing the errors and a Heaviside function to convert input spikes into a continuous signal.

13

feature extractors. NCGs consistently improve the performance of S4NN, aligning with the results from the main paper on STDP-based rules. Yet, further research is required to evaluate the impact of the method for computing the errors (and the desired firing times) on the effectiveness of NCGs.

4   Comparison with SOTA Methods

SOTA methods for direct training of SNNs [10, 11, 12] rely on backpropagation through time (BPTT) [13] and surrogate gradient [14]. These methods usually allow multiple spikes per neuron and support the training of very deep networks using global supervised learning. In this work, we allow one spike per neuron, train all layers with local learning (limiting our networks to shallow architectures), and use a semi-supervised training strategy, where only the last layer is trained with supervision. In terms of performance, our methods lag behind fully-supervised SOTA methods. For instance, [12] report an accuracy of 96.44% on CIFAR-10 (our best model achieves 79.55%) and 81.65% on CIFAR-100 (our best model achieves 53.49%). This decrease in accuracy can partially be attributed to the number of layers employed (4 against 19) and the use of supervision limited to the last layer. In terms of computational and memory costs, BPTT is extremely inefficient since these costs scale with the latency (i.e. the number of time steps), whereas the costs of our methods are independent of the latency. Also, a backward pass with BPTT adjusts all synapses in the network, whereas our methods adjust only the synapses of neurons that have fired. In terms of energy efficiency, our single-spike strategy may limit the number of generated spikes significantly compared to multiple-spike methods, which reduces power consumption in both training and inference. In terms of hardware suitability, BPTT is challenging to implement on neuromorphic hardware because it relies on non-local learning [15]. BPTT-based SNNs must be trained on GPUs, which is energy-intensive [16, 17], and can be deployed on chip for inference only. To fully exploit the energy-efficient capabilities of SNNs, both training and inference should be performed on chip. We target, for instance, memristive-based chips [18] for hardware implementation of our methods. They are excellent candidates for ultra-low-power applications, potentially reducing energy consumption by several orders of magnitude compared to GPUs [19, 16]. Also, STDP is inherently implemented in memristor circuits [20, 21], which facilitates on-chip training [22, 23]. There are still several challenges to address before our work can be implemented on this type of chip, such as the need for a digital module to calculate the error. This should be the focus of future work.

References

[1] Pierre Falez, Pierre Tirilly, Ioan Marius Bilasco, Philippe Devienne, and Pierre Boulet. Multi-Layered Spiking Neural Network with Target Timestamp Threshold Adaptation and STDP. In *International Joint Conference on Neural Networks*, 2019.

[2] Pierre Falez, Pierre Tirilly, and Ioan Marius Bilasco. Improving STDP-based Visual Feature Learning with Whitening. In *International Joint Conference on Neural Networks*, 2020.

[3] Simon Thorpe, Arnaud Delorme, and Rufin Van Rullen. Spike-Based Strategies for Rapid Processing. *Neural Networks*, 14:715–725, 2001.

[4] Gaspard Goupy, Pierre Tirilly, and Ioan Marius Bilasco. Paired Competing Neurons Improving STDP Supervised Local Learning in Spiking Neural Networks. *Frontiers in Neuroscience*, 18, 2024.

[5] Adam Coates, Andrew Ng, and Honglak Lee. An Analysis of Single-Layer Networks in Unsupervised Feature Learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011.

[6] Adrien Journé, Hector Garcia Rodriguez, Qinghai Guo, and Timoleon Moraitis. Hebbian Deep Learning Without Feedback. *International Conference on Learning Representations*, 2023.

[7] Milad Mozafari, Saeed Reza Kheradpisheh, Timothee Masquelier, Abbas Nowzari-Dalini, and Mohammad Ganjtabesh. First-Spike-Based Visual Categorization Using Reward-Modulated STDP. *Transactions on Neural Networks and Learning Systems*, 29:6178–6190, 2018.

14

[8] Franck Cappello, Frédéric Desprez, Michel Daydé, Emmanuel Jeannot, Yvon Jégou, Stephane Lanteri, Nouredine Melab, Raymond Namyst, Pascale Primet, Olivier Richard, Eddy Caron, Julien Leduc, and Guillaume Mornet. Grid'5000: A Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform. In *International Workshop on Grid Computing*, 2005.

[9] Saeed Reza Kheradpisheh and Timothée Masquelier. Temporal Backpropagation for Spiking Neural Networks with One Spike per Neuron. *International Journal of Neural Systems*, 30, 2020.

[10] Chaoteng Duan, Jianhao Ding, Shiyan Chen, Zhaofei Yu, and Tiejun Huang. Temporal Effective Batch Normalization in Spiking Neural Networks. *Advances in Neural Information Processing Systems*, 35, 2022.

[11] Man Yao, JiaKui Hu, Zhaokun Zhou, Li Yuan, Yonghong Tian, Bo Xu, and Guoqi Li. Spike-driven Transformer. *Advances in Neural Information Processing Systems*, 36, 2023.

[12] Yuhang Li, Tamar Geller, Youngeun Kim, and Priyadarshini Panda. SEENN: Towards Temporal Spiking Early Exit Neural Networks. *Advances in Neural Information Processing Systems*, 36, 2023.

[13] Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-Temporal Backpropagation for Training High-Performance Spiking Neural Networks. *Frontiers in Neuroscience*, 12, 2018.

[14] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate Gradient Learning in Spiking Neural Networks: Bringing the Power of Gradient-Based Optimization to Spiking Neural Networks. *Signal Processing Magazine*, 36:51–63, 2019.

[15] Friedemann Zenke and Emre Neftci. Brain-Inspired Learning on Neuromorphic Substrates. *Proceedings of the IEEE*, 109:935–950, 2021.

[16] Jiwei Li, Hui Xu, Sheng-Yang Sun, Nan Li, Qingjiang Li, Zhiwei Li, and Haijun Liu. In Situ Learning in Hardware Compatible Multilayer Memristive Spiking Neural Network. *Transactions on Cognitive and Developmental Systems*, 14:448–461, 2022.

[17] Shiya Liu, Nima Mohammadi, and Yang Yi. Quantization-Aware Training of Spiking Neural Networks for Energy-Efficient Spectrum Sensing on Loihi Chip. *Transactions on Green Communications and Networking*, 2023.

[18] Doo Seok Jeong, Kyung Min Kim, Sungho Kim, Byung Joon Choi, and Cheol Seong Hwang. Memristors for Energy-Efficient New Computing Paradigms. *Advanced Electronic Materials*, 2, 2016.

[19] Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J. Joshua Yang, and He Qian. Fully Hardware-Implemented Memristor Convolutional Neural Network. *Nature*, 577:641–646, 2020.

[20] Damien Querlioz, Olivier Bichler, and Christian Gamrat. Simulation of a Memristor-Based Spiking Neural Network Immune to Device Variations. In *International Joint Conference on Neural Networks*, pages 1775–1781, 2011.

[21] Catherine D. Schuman, Thomas E. Potok, Robert M. Patton, J. Douglas Birdwell, Mark E. Dean, Garrett S. Rose, and James S. Plank. A Survey of Neuromorphic Computing and Neural Networks in Hardware. *ArXiv*, arXiv:1705.06963 [cs.NE], 2017.

[22] Sylvain Saïghi, Christian G. Mayr, Teresa Serrano-Gotarredona, Heidemarie Schmidt, Gwendal Lecerf, Jean Tomas, Julie Grollier, Sören Boyn, Adrien F. Vincent, Damien Querlioz, Selina La Barbera, Fabien Alibart, Dominique Vuillaume, Olivier Bichler, Christian Gamrat, and Bernabé Linares-Barranco. Plasticity in Memristive Devices for Spiking Neural Networks. *Frontiers in Neuroscience*, 9, 2015.

[23] Lyes Khacef, Philipp Klein, Matteo Cartiglia, Arianna Rubino, Giacomo Indiveri, and Elisabetta Chicca. Spike-Based Local Synaptic Plasticity: A Survey of Computational Models and Neuromorphic Circuits. *Neuromorphic Computing and Engineering*, 3, 2023.

15