# OPTIMIZING THE CONSUMPTION OF SPIKING NEURAL NETWORKS WITH ACTIVITY REGULARIZATION

*Simon Narduzzi*[⋆†] *Siavash A. Bigdeli*[⋆] *Shih-Chii Liu*[†] *L. Andrea Dunbar*[⋆]

[⋆]CSEM, Neuchâtel, Switzerland [†]Institute of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland

## ABSTRACT

Reducing energy consumption is a critical point for neural network models running on edge devices. In this regard, reducing the number of multiply-accumulate (MAC) operations of Deep Neural Networks (DNNs) running on edge hardware accelerators will reduce the energy consumption during inference. Spiking Neural Networks (SNNs) are an example of bio-inspired techniques that can further save energy by using binary activations, and avoid consuming energy when not spiking. The networks can be configured for equivalent accuracy on a task through DNN-to-SNN conversion frameworks but their conversion is based on rate coding therefore the synaptic operations can be high. In this work, we look into different techniques to enforce sparsity on the neural network activation maps and compare the effect of different training regularizers on the efficiency of the optimized DNNs and SNNs.

***Index Terms***— Sparsity, Regularization, Spiking Neural Networks, Deep Neural Networks

## 1. INTRODUCTION

Today, deep learning models are regularly deployed at the edge, allowing local real-time decision-making, efficient preprocessing, and privacy-preserving applications. Optimizations have been developed in the past few years to allow the deployment of these networks within restricted resource environments; quantization [1], pruning [2], distillation [3], are some of them, which are applied either during training or post-training of the neural network. Great emphasis is also put on the development of efficient accelerators, that reach competitive performance compared to CPUs and GPUs. Recent hardware accelerators include optimization techniques such as computational reduction by zero-skipping [4, 5]. These solutions have been developed to skip zero weight computation in convolution layers [5], fully-connected layers [4] and activations [6]. Therefore, to make the most out of this hardware, it is necessary to train and deploy sparse neural networks.

Biological neurons use discrete spikes to compute and transmit information using spike timing and spike rates to encode information. SNNs, inspired by biological neurons, imitate this behavior and are thus closer to biological systems than conventional deep neural networks. Additionally, the sparse nature of spikes makes SNNs more suitable for low-power inference. The interest of such systems is attested by the development of neuromorphic hardware supporting SNNs, which is an active area of academic and industrial research: Intel Loihi, Synsense DynapCNN, and IBM TrueNorth are some of the recent chips developments that consume only $1/10'000$th of the energy of traditional microprocessors.

While SNNs show attractive power trade-offs for machine learning algorithms running on the edge, methods to train them efficiently are still behind conventional techniques: their binary nature makes them untrainable using backpropagation (BP) algorithms. Variations of the BP algorithm have been recently developed, allowing precise spike-timing learning, but the accuracy of DNNs is still not obtained [7, 8]. Therefore, conversion techniques have been developed to allow the transformation of BP-trained DNNs to SNNs [9, 10], bringing SNNs on par with state-of-the-art deep neural networks.

As a key property of SNNs, recent techniques have looked into incorporating sparsity inside the pre-conversion training for SNNs. This is mainly enforced by an activation regularization during the DNN training. It has been shown empirically that this regularization leads to higher sparsity of spikes after the DNN-to-SNN conversion [10]. Due to complex optimizations in the conversion step, the exact effect of the regularizer still remains unknown in the final implementation of the SNN. Moreover, the effect of the regularization function itself has not been investigated in the context of stochastic optimization of the neural networks. In this work, we investigate different regularization techniques for sparsity and their effects on training DNNs. Additionally, we look into their influence in sparsity for SNNs after the conversion.

The rest of the paper is organized as follows: Section 2 presents a brief overview of prior work in sparsity training methods for neural networks. Section 3 describes the regularization techniques and metrics used in our setup. Section 4 introduces our experimental scheme and results. Section 5 summarizes the work of this paper and describes future research directions.

## 2. RELATED WORK

Efforts have been made toward the sparsification of deep neural networks to reduce the memory footprint of the models deployed at the edge. Weight sparsification is achieved mainly through pruning methods [11]. Pruning entire feature maps have also been studied [12, 13, 14] to remove redundant information and subsequently reduce network computes. In SNNs, spikes and synaptic computation reduction are mostly exploited through temporal and spatial sparsity. Temporal sparsity of SNNs have inspired training techniques in deep learning [15, 16], targeting time-series applications.

Recently, regularization techniques have been applied to SNN training [17, 18] to increase spatial sparsity, but these do not rely on the regularization of BP-trained DNNs prior to SNN conversion. Sorbaro et al.[10] proposes a loss to optimize the number of synaptic operations (SynOps), applied to a DNN, acting as $L_1$ regularization on activation and weights of each neuron. Spike count reduction using $L_2$ regularization on activity maps have been studied by [19] to reduce the number of spikes of the converted models. However, true sparsity should be obtained using $L_0$ regularization, which expresses the exact number of zeros in the neural network activation map. In our work, we look at various sparsity objectives (Section 3.2), including surrogate and Hoyer [20] approximations and compare their performance on two DNNs and SNNs.

## 3. METHODS

### 3.1. Metrics

The efficacy of a DNN is expressed as Effective FLOPS (EFLOPS). Assuming that smart and efficient hardware platforms exist and perform computation based only on non-zero activations and non-zero weights, the EFLOPS metric describes the exact number of MAC operations performed. Therefore, the energy savings of running the network on this system instead of on a classical accelerator will be proportional to the level of sparsity in the network. Biases are included in the EFLOPS if they are non-zero. We can then express the number of effective FLOPS used to compute an activation of the layer $\ell$ of a network as:

$$EFLOPS_\ell = \phi(W_\ell) \times \phi(A_{\ell-1}) + \phi(B_\ell), \qquad (1)$$

where $W_\ell$ is the weight matrix, $A_{\ell-1}$ is the input activation map, $B_\ell$ is the bias, and function $\phi(x) := x \neq 0$ outputs a mask with ones, where its input has non-zero values.

### 3.2. Regularizers

We measure the activity of the network of each layer, which correlates with the SynOps emitted by the network when converted to a spiking version [10, 19]. We denote the activation value of neuron $i$ in a layer as $x_i$. Several regularizers are tested to evaluate their capacity to generate sparse activation maps. The regularization is applied using normalization of the activation. We selected a few regularizers, whose 2D landscapes are displayed in Fig. 1:

$\boldsymbol{L_2}$ : Also known as the Euclidian norm, has already demonstrated its effectiveness in the reduction of the number of spikes [19]. While the gradients of this norm are good at high values, they tend to be reduced as the values approach zero.

$\boldsymbol{L_1}$ : Also known as the Manhattan Distance, $L_1$ norm is the sum of all neurons values in the output activation map. Gradients of $L_1$ are the same everywhere and tend to have better sparsifying power than $L_2$ as the penalty for high and low values is the same.

$\boldsymbol{L_0}$ : Indicates the number of non-zero elements in a vector. Therefore, reducing this value of the norm yield to more zero in the activation map. We hypothesize that this norm has the best sparsifying power. However, it has non-informative gradients, making it unusable during the optimization.

$\boldsymbol{L_{p<1}}$ : We use a surrogate $L_0$ in the form of $L_p$, where $p$ is a positive number smaller than one. This regularization has the advantage of being differentiable everywhere, and the speed of approach (gradient) towards 0-values can be controlled using the value $p$. $L_p$ regularization is defined as:

$$L_p(X) = \sum_i |x_i|^p. \qquad (2)$$

**Hoyer ($H$)** and **Hoyer-Squares ($H_S$)** : The Hoyer regularization is the ratio between $L_1$ and $L_2$ norm. Hoyer-Squares normalization is an approximation of the $L_0$ norm which has the advantage of being scale invariant as for the $L_0$ norm:

$$H(X) = \frac{\sum_i |x_i|}{\sqrt{\sum_i x_i^2}}, \quad \text{and} \quad H_S(X) = \frac{(\sum_i |x_i|)^2}{\sum_i x_i^2}. \qquad (3)$$

Finally, we define our loss function as:

$$\mathcal{L} = CE + \lambda_{reg} \sum_\ell \psi(X_\ell), \qquad (4)$$

where $CE$ is the cross-entropy loss, $\lambda_{reg}$ controls the regularization weight, $X_\ell$ is the activation map of layer $\ell$ in the neural network, and $\psi \in \{L_1, L_2, L_p, H, H_S\}$ indicates the chosen regularization function. In this work, we evaluate the reduction in number of computes relative to the value $\lambda_{reg}$.

### 3.3. Network architecture and training procedure

To assess the properties of regularization algorithms on the computational complexity of neural networks, two models were trained: LeNet-5, a convolutional neural network, and a 784-300-100-10 multi-layer perceptron (MLP), both with

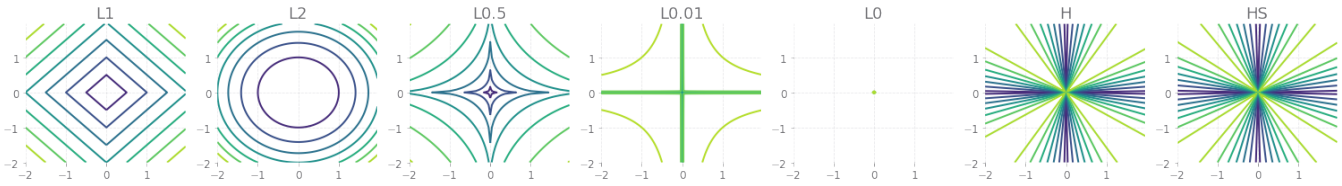


**Fig. 1**: Loss landscape for regularization methods used in our experiment.

ReLU activations in hidden layers. The models were trained with different regularizers on the MNIST dataset with Adam optimizer, a learning rate of $1e-4$, and batch-size of 128. 5000 images from the training set are kept for validation. The convergence criterion is set to be the smallest validation loss, with the patience of 20 epochs. The model is then converted to a SNN using the procedure described in [9]. The SNN model is calibrated on the full training set and the accuracy is reported on the entire test set. The converted model is simulated for 100 time-steps, and the test accuracy at the end of the simulation is reported as the SNN accuracy. The reported EFLOPS, number of spikes and SynOps are averaged per each sample over the course of the simulation. We repeated the experiments with several random seeds and obtained similar results. Depending on the hardware, the consumption might be impacted by both spike emission and SynOps. Therefore, we report these two values independently. SynOps are however considered to be the dominant source of the power. We demonstrate that our approach can be applied to other datasets by training and testing LeNet-5 on the CIFAR-10 dataset with the same procedure, and exploring a few $\lambda_{reg}$ values. In order to get results comparative to DNNs, we increase the simulation time-steps to 1000 in this experiment.

## 4. RESULTS

### 4.1. Activity regularization effect on number of SynOps

In Fig. 2, we show the results of the regularization on the different metrics using the MLP trained on MNIST. The regularizers have different dynamics, but we observe a general trend toward the reduction of activity with increasing values of the regularization constraint, $\lambda_{reg}$. When $\lambda_{reg}$ is too high, we observe a significant accuracy drop, as the network can not learn anymore. The results of the best trade-off models are represented in Table 2. For each regularization method, we take the best model with less than $0.5\%$ and $5\%$ difference in accuracy after conversion for MNIST and CIFAR-10 respectively, with the least number of SynOps. We observe that activity regularization greatly reduces the number of spikes and SynOps of the converted network, irrespective of the chosen regularizer. On MNIST, the total number of spikes is reduced by more than $90\%$ and the number of SynOps by $96\%$ on the MLP using $L_{0.5}$. For the LeNet-5 architecture, $L_1$ achieves the best reduction in both SynOps and number of spikes. $L_2$ still reduces the number of SynOps by $90\%$ while keeping the accuracy of the converted SNN the same. We observe that $L_{0.01}$ might be to aggressive for the models, and is worse than $L_1$ in both models. The reduction of SynOps in CIFAR-10 experiment is less significant than in MNIST. This can be due to the input (natural images) being less sparse than MNIST samples. We did not test Hoyer regularizers on CIFAR-10 because of its poor performance on MNIST.

We observed that sometimes, applying regularization results in better accuracy than the baseline. However, on edge devices, an improvement of $0.2\%$ in accuracy is less desirable than an $\geq 94\%$ improvement in energy consumption due to the number of computes, as edge devices are designed for low-power and high speed rather than accuracy. While the number of SynOps is reduced, there is no correlation with the number of EFLOPS computed in the model. This suggests that SynOps reduction via pre-conversion training is more complex than only optimizing a DNN network and that artefacts are introduced during the conversion.

| | **Spikes** | **EFLOPS** | **SynOps** | **Accuracy** (DNN/SNN) |
|---|---|---|---|---|
| MLP baseline | 326'185 | 67'183 | 22'374'991 | 97.91% / 97.98% |
| **Reg. @ $\lambda_{reg}$** | **%Spikes** | **%EFLOPS** | **%SynOps** | **ΔAccuracy** |
| L2 @ 9e-04 | -67.29% | -27.20% | -87.29% | -0.29%/-0.40% |
| L1 @ 7e-04 | -86.65% | -29.46% | -94.61% | -0.34%/-0.44% |
| L0.5 @ 9e-04 | **-90.93%** | **-30.42%** | **-96.37%** | -0.30%/-0.43% |
| L0.01 @ 7e-03 | -83.83% | -28.70% | -91.15% | -0.38%/-0.44% |
| H @ 7e-03 | -76.33% | -4.91% | -86.14% | 0.14%/-0.45% |
| Hs @ 3e-05 | -70.29% | -3.98% | -82.94% | 0.10%/-0.25% |
| LeNet-5 baseline | 4'600'497 | 307'599 | 258'831'814 | 98.72% / 98.39% |
| **Reg. @ $\lambda_{reg}$** | **%Spikes** | **%EFLOPS** | **%SynOps** | **ΔAccuracy** |
| L2 @ 1e+00 | -83.89% | -77.85% | -90.14% | 0.03%/0.10% |
| L1 @ 1e-02 | **-89.72%** | -77.43% | **-93.97%** | 0.12%/-0.47% |
| L0.5 @ 3e-04 | -89.03% | -78.94% | -91.18% | 0.09%/0.18% |
| L0.01 @ 3e-04 | -87.86% | **-79.24%** | -88.10% | -0.18%/-0.43% |
| H @ 9e-03 | -60.32% | -43.63% | -60.37% | 0.36%/-0.18% |
| Hs @ 1e-05 | -37.58% | -36.76% | -42.89% | 0.28%/0.44% |

**Table 1**: Results for MLP and LeNet-5 on MNIST.

| | **Spikes** | **EFLOPS** | **SynOps** | **Accuracy** (DNN/SNN) |
|---|---|---|---|---|
| LeNet-5 baseline | 61'231'514 | 613'508 | 3'425'593'226 | 57.92% / 51.74% |
| **Reg. @ $\lambda_{reg}$** | **%Spikes** | **%EFLOPS** | **%SynOps** | **ΔAccuracy** |
| L2 @ 1e-01 | **-76.48%** | -21.24% | **-81.48%** | 2.29%/2.76% |
| L1 @ 1e-03 | -68.98% | -15.64% | -72.36% | 4.79%/5.11% |
| L0.5 @ 1e-05 | -15.28% | -1.17% | -15.22% | 1.16%/1.94% |
| L0.01 @ 1e-03 | -61.72% | **-23.14%** | -57.65% | -4.01%/-3.60% |

**Table 2**: Results for LeNet-5 on CIFAR-10.

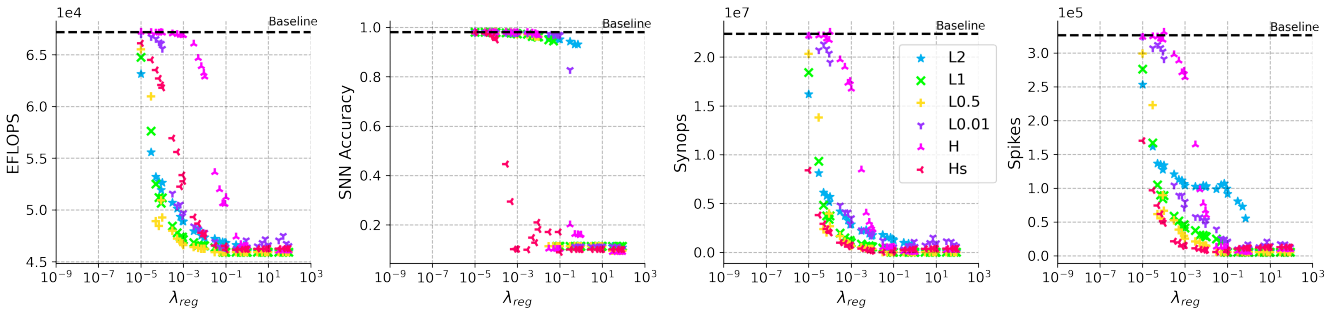


**Fig. 2**: Metrics relative to the regularization constant $\lambda_{reg}$ in the converted SNN MLP architecture.

| Architecture | L2 | L1 | L0.5 | L0.01 | H | Hs |
|---|---|---|---|---|---|---|
| LeNet-5 | 0.944/0.945 | 0.966/0.966 | **0.972/ 0.973** | 0.854/0.937 | 0.807/0.809 | 0.844/0.844 |
| MLP | 0.965/0.965 | 0.973/0.973 | **0.974/ 0.975** | 0.936/0.970 | 0.922/0.928 | 0.936/0.939 |
| Average | 0.954/0.955 | 0.970/0.970 | **0.973/0.974** | 0.895/0.954 | 0.865/0.869 | 0.890/0.891 |

**Table 3**: Area under the curve (AUC) for both architectures. The values are formatted as (AUC (original) / AUC (smoothed)).

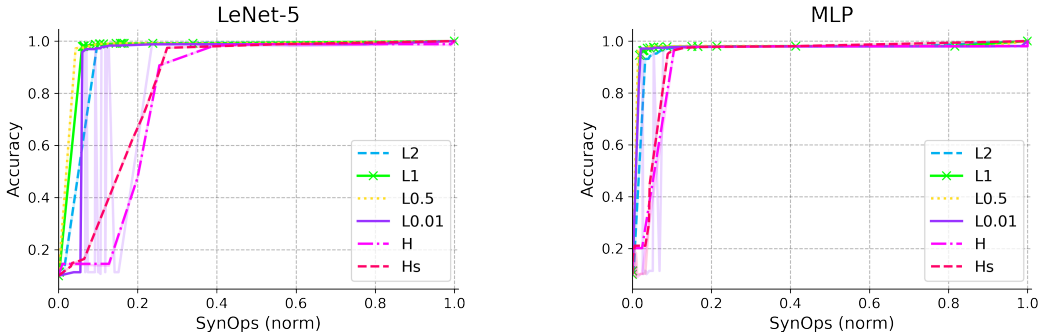


**Fig. 3**: Accuracy vs SynOps curves for each model. SynOps are normalized to the maximum value across the models.

### 4.2. $L_p$ vs. Hoyer regularization

To further compare the performance of each metric against the other, we plot the accuracy versus the number of SynOps. As the SynOps reduction is not always monotonic (especially in the LeNet-5 architecture), the smoothed version of the curves is computed. The x-axis is normalized to the maximum number of SynOps obtained by the regularized models. Fig. 3 shows the curves for the models trained on MNIST, and Table 3 provides the corresponding Area Under the Curve (AUC) values of the original and smoothed version of the curves. We observe that Hoyer-based regularizers produce more SynOps than $L_p$ for the same accuracy in both models. $L_{p>0.01}$ have almost identical sparsifying power. We also observe the instability of the $L_{0.01}$ regularizer, where the accuracy oscillates when the SynOps become too small, suggesting that the regularization was too strong. Regularization seems to be less effective on the LeNet-5 architecture trained on CIFAR-10, suggesting that it requires more SynOps to express the features of this dataset.

## 5. CONCLUSION

Our results demonstrate that activity regularization during the training of DNNs is a simple way to reduce the number of spikes and SynOps in converted SNNs. We also show that Hoyer regularization has limited effect compared to $L_p$ regularization and that reducing $p$ does not necessarily lead to better results. This suggests that a better approximation of $L_0$ with smaller gradients than $L_{0.01}$ can give more stability and potentially even better performance. Another potential improvement is to explore the simultaneous regularization of both weights and activations to generate very efficient SNNs: EFLOPS have not been reduced much in these models, as the weights were not constrained and could have been non-zero, potentially leading to high activations. Finally, fine-tuning and regularization on the post-converted networks could further improve the ability of the presented methods to obtain further sparsity while keeping competitive accuracy.

This project was partially funded by EU H2020, through ANDANTE grant no. 876925.

## 6. REFERENCES


[1] Tailin Liang, John Glossner, Lei Wang, and Shaobo Shi, "Pruning and quantization for deep neural network acceleration: A survey," *arXiv preprint arXiv:2101.09671*, 2021.

[2] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste, "Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks," *arXiv preprint arXiv:2102.00554*, 2021.

[3] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao, "Knowledge distillation: A survey," *International Journal of Computer Vision*, vol. 129, no. 6, pp. 1789–1819, 2021.

[4] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally, "Eie: Efficient inference engine on compressed deep neural network," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 243–254, 2016.

[5] Angshuman Parashar, Minsoo Rhu, Anurag Mukkara, Antonio Puglielli, Rangharajan Venkatesan, Brucek Khailany, Joel Emer, Stephen W Keckler, and William J Dally, "Scnn: An accelerator for compressed-sparse convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 27–40, 2017.

[6] Dongyoung Kim, Junwhan Ahn, and Sungjoo Yoo, "Zena: Zero-aware neural network accelerator," *IEEE Design & Test*, vol. 35, no. 1, pp. 39–46, 2017.

[7] 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," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019.

[8] Maryam Mirsadeghi, Majid Shalchian, Saeed Reza Kheradpisheh, and Timothée Masquelier, "Stidi-bp: Spike time displacement based error backpropagation in multilayer spiking neural networks," *Neurocomputing*, vol. 427, pp. 131–140, 2021.

[9] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu, "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification," *Frontiers in neuroscience*, vol. 11, pp. 682, 2017.

[10] Martino Sorbaro, Qian Liu, Massimo Bortone, and Sadique Sheik, "Optimizing the energy consumption of spiking neural networks for neuromorphic applications," *Frontiers in neuroscience*, vol. 14, pp. 662, 2020.

[11] Christos Louizos, Max Welling, and Diederik P Kingma, "Learning sparse neural networks through $l\_0$ regularization," *arXiv preprint arXiv:1712.01312*, 2017.

[12] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2736–2744.

[13] Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh, "Inducing and exploiting activation sparsity for fast inference on deep neural networks," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5533–5543.

[14] Georgios Georgiadis, "Accelerating convolutional neural networks via activation map compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 7085–7095.

[15] Amirreza Yousefzadeh and Manolis Sifalakis, "Training for temporal sparsity in deep neural networks, application in video processing," *arXiv preprint arXiv:2107.07305*, 2021.

[16] Amirreza Yousefzadeh, Mina A. Khoei, Sahar Hosseini, Priscila Holanda, Sam Leroux, Orlando Moreira, Jonathan Tapson, Bart Dhoedt, Pieter Simoens, Teresa Serrano-Gotarredona, and Bernabe Linares-Barranco, "Asynchronous spiking neurons, the natural key to exploit temporal sparsity," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 668–678, 2019.

[17] Junhong Zhao, Jie Yang, Jun Wang, and Wei Wu, "Spiking neural network regularization with fixed and adaptive drop-keep probabilities," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[18] Thomas Pellegrini, Romain Zimmer, and Timothée Masquelier, "Low-activity supervised convolutional spiking neural networks applied to speech commands recognition," in *2021 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2021, pp. 97–103.

[19] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu, "Learning to be efficient: Algorithms for training low-latency, low-compute deep spiking neural networks," in *Proceedings of the 31st annual ACM symposium on applied computing*, 2016, pp. 293–298.

[20] Huanrui Yang, Wei Wen, and Hai Li, "Deephoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures," *arXiv preprint arXiv:1908.09979*, 2019.