Spiking Brain Compression: Exploring One-Shot Post-Training Pruning and Quantization for Spiking Neural Networks

Lianfeng Shi MD24445@BRISTOL.AC.UK
Ao Li YT22389@BRISTOL.AC.UK
Benjamin Ward-Cherrier B.WARD-CHERRIER@BRISTOL.AC.UK
University of Bristol

Abstract

Spiking Neural Networks (SNNs) have emerged as a new generation of energy-efficient neural networks suitable for implementation on neuromorphic hardware. As neuromorphic hardware has limited memory and computing resources, weight pruning and quantization have recently been explored to improve SNNs' efficiency. State-of-the-art SNN pruning/quantization methods employ multiple compression and training iterations, increasing the cost for pre-trained or very large SNNs. In this paper, we propose a new one-shot post-training pruning/quantization framework, Spiking Brain Compression (SBC), that extends the Optimal Brain Compression (OBC) method to SNNs. SBC replaces the current-based loss found in OBC with a spike train-based objective whose Hessian is cheaply computable, allowing a single backward pass to prune or quantize synapses and analytically rescale the rest. Our experiments on models trained with neuromorphic datasets (N-MNIST, CIFAR10-DVS, DVS128-Gesture) and large static datasets (CIFAR-100, ImageNet) show state-of-the-art results for one-shot post-training compression methods on SNNs, with single-digit to double-digit accuracy gains compared to OBC. SBC also approaches the accuracy of costly iterative methods, while cutting compression time by 2-3 orders of magnitude.

Keywords: Second-order methods, neuromorphic computing, spiking neural networks

1. Introduction

Spiking Neural Networks (SNNs) have garnered significant attention as a promising power-efficient alternative to traditional Artificial Neural Networks (ANNs). However, deploying large SNNs on neuromorphic hardware faces computational and memory constraints. Neural pruning and quantization emerged as essential methods for reducing neuron activations and weight storage for SNNs. Existing SNN pruning [2, 12, 21] and quantization [9, 17, 22] methods often rely on expensive iterative compression techniques, involving multiple pruning and training iterations. This is computationally expensive, especially for pre-trained and deep spiking neural networks. With the adaptation of the transformer architecture to SNNs [23, 24], the size of future state-of-the-art SNN models could cause the retraining cost to become prohibitively large.

In this paper, we introduce Spiking Brain Compression (SBC), a layer-wise compression method using a second-order approximation of SNNs to prune and/or quantize an SNN in one shot, without retraining, inspired by the Optimal Brain Compression (OBC) algorithm [8]. We discovered that the naive application of OBC to SNNs does not yield the best result, due to the disconnect between OBC's objective function and SNNs' layer-wise output spike train. We propose an innovative layer-wise objective function derived from the Van Rossum Distance (VRD) [20] of output spike trains, referred to as surrogate membrane potential (SMP). By statistically estimating SMP's Hessian from a calibration dataset, SBC maintains OBC's computational cost while providing tighter compression guarantees. We validate SBC on both neuromorphic benchmarks and large-scale static datasets, demonstrating efficient, accurate pruning and quantization on SNNs.

We summarize our contribution as follows:

- We formalize a VRD–based loss for layer-wise SNN compression and derive an efficient Hessian that can be sampled from a calibration dataset.
- We develop *Spiking Brain Compression (SBC)*: a one-shot second-order SNN pruning and quantization algorithm that works with complex SNNs like Spiking ResNets and Spiking Transformers.
- We validate SBC empirically across domains, demonstrating aggressive compression (97%) with minimal accuracy loss ($\leq 1\%$) on neuromorphic tasks and scalability by compressing SEW-ResNet 152 on the ImageNet benchmark.

2. Preliminary

2.1. Leaky-Integrate-and-Fire (LIF) neuron

We use the discrete LIF neuron model described in Fang et al. [5], with $V_{reset} = 0$. In a typical SNN layer with D LIF neurons with T time steps, U, S, and V are matrices with dimension $T \times D$. $U[t] = U_{t,:}$ and $V[t] = V_{t,:}$ represent the neuron membrane potentials after neuron dynamics, and after the trigger of a spike at time t, respectively. $I[t] = I_{t,:}$ denotes the pre-activation input at time t, which we call external input current. $S[t] = S_{t,:}$ denotes the output spike train. Θ is the Heaviside step function, and τ_m is the membrane time constant, which governs the rate of decay for the membrane potential.

$$U[t] = \left(1 - \frac{1}{\tau_m}\right) V[t-1] + \frac{1}{\tau_m} I[t] \tag{1}$$

$$S[t] = \Theta(U[t] - V_{th}) \tag{2}$$

$$V[t] = U[t](1 - S[t]) \tag{3}$$

As a recursive system, the initial condition assumes $U[0] = \frac{1}{\tau_m} I[0]$, with $I = WX$, where $X_{T\times D}$ are the layer input spike trains over T time steps.

3. Methodology

Module-wise compression Because small current errors accumulate through LIF dynamics, we measure distortion at the output spike train. To achieve this, we compress SNNs *module-wise*: Every module is Linear/Conv(+BN) $\rightarrow$ LIF; BN and Conv are folded into a single linear map, so each module becomes Linear(W) $\rightarrow$ LIF. (Appendix A visualizes the module concept in an SNN). For an input spike tensor $X \in \{0,1\}^{T\times d_{\text{input}}}$ and weights $W \in \mathbb{R}^{d_{\text{input}}\times d_{\text{output}}}$, the module produces a spike train $S = f(X, W) \in \{0,1\}^{T\times d_{\text{output}}}$.

Problem setup The goal of module-wise compression is to find a ”compressed” W, which we denote $\hat{W}$, that performs as similarly to the original weights as possible. More formally, the compressed weights $\hat{W}$ should minimize some loss function $\mathcal{L}$ while satisfying some constraints on $\hat{W}$, expressed as $\mathcal{C}(\hat{W}) > C$:

$$argmin_{\hat{W}} E_X[\mathcal{L}(f(X,W), f(X,\hat{W}))], \text{ s.t. } \mathcal{C}(\hat{W}) > C \tag{4}$$

The expectation is approximated using N calibration samples. We treat modules independently and sweep them greedily, largely following the method of Frantar et al. [8].

3.1. Spiking Brain Compression (SBC)

This paper proposes a framework for finding an appropriate loss function $\mathcal{L}$ and obtaining the loss function's Hessian $\boldsymbol{H}$ for general SNNs. We call it Spiking Brain Compression, following the naming convention of its ANN counterpart, Optimal Brain Compression (OBC) [8].

3.1.1. LOSS FUNCTION $\mathcal{L}$

We name the post-compression output spike train $\hat{S} = f(X, \hat{W})$. Due to the discrete nature of S and $\hat{S}$, using the squared L2 Norm like OBC can only tell the number of spike differences between two spike train, but not the spike timing. Following the SNN convention, we used the squared VRD between the pre- and post-compression output spike trains as the loss function. Here we represent the VRD decaying exponential kernel $k(t)$ as a function of time constant τ:

$$k[t] = \begin{cases} (1-\frac{1}{\tau})^t \frac{1}{\tau} & \text{if } t \geq 0 \\ 0 & \text{if } t < 0 \end{cases} \tag{5}$$

The loss can then be expressed as:

$$\mathcal{L}(W) = VRD(S, \hat{S}) = ||S(t) * k(t) - \hat{S}(t) * k(t)||_2^2 = ||MS - M\hat{S}||_2^2, \tag{6}$$

where the square matrix M of size $d_{time} \times d_{time}$ is the convolution matrix of kernel $k(t)$ for d_{time} timesteps. Since the output spike train of an LIF neuron depends only on its own external input current, and $k(t)$ is a convolution on the time dimension, the VRD of an LIF neuron only depends on the weights feeding into the neuron. This means we can write the loss in Eq. 6 as the sum of the square VRD of each neuron in the layer.

$$||MS - M\hat{S}||_2^2 = \sum_{j=1}^{d_{output}} ||MS_{:,j} - M\hat{S}_{:,j}||_2^2 \tag{7}$$

This means we can calculate the Hessian for the synaptic connections to each LIF neuron (each column of weights in W) separately, as there are no inter-neuron dependencies. In the following section, we analyze the loss function of an individual neuron, with lower case s, $\hat{s}$, and w represents the spiketrains and synaptic connections of an individual LIF neuron: $\mathcal{L} = ||Ms - M\hat{s}||_2^2 = ||Mf(X,w) - Mf(X,\hat{w})||_2^2$.

3.1.2. SURROGATE MEMBRANE POTENTIAL

Here, we provide SMP, an approximation of the exact Hessian that is computationally efficient and performs well empirically across both small and large SNNs.

We first provide the Hessian $\mathcal{H}$ of the loss function $\mathcal{L}$ under the OBS framework. Detailed derivation can be found in Appendix B:

$$\mathcal{H}_{ij} = 2(M\frac{\partial s}{\partial u}x_j)^T M\frac{\partial s}{\partial u}x_i + 2(Ms - M\hat{s})^T Mx_i\frac{\partial^2 s}{\partial u^2}x_j \tag{8}$$

Where u and s stand for the layer spike train and layer exact membrane potential, and $\frac{\partial s}{\partial u}$ is the $d_{times} \times d_{times}$ Jacobian, which we name h'.

We observe that the spike at time t $s[t] = \Theta(u[t] - V_{th})$ (Eq. 2) only depends on the membrane potential at time t. This means h' is a diagonal matrix. Inspired by surrogate gradient [18], we replace the non-differentiable Heaviside function Θ with a differentiable gradient function g, thus h' is a diagonal matrix where $h'_{ii} = g(u_i)$.

There are many different ways to design a surrogate gradient function g, and we intend to explore them in future work. Currently, we found that a constant function $g(u) = c$, works well for layer-wise compression, as supported by empirical evidence in section 5. g can be seen as a *shallow rectangle surrogate gradient* where every u falls in the active window. Since the OBS framework [10] works with the relative magnitude of the Hessian matrix, c cancels out. Without loss of generality, we set $c = 1$. Let $\boldsymbol{H} = E_X[\mathcal{H}]$ be the expectation of the Hessian over a distribution of input X. We now get the Hessian for SMP:

$$\boldsymbol{H}_{SMT} = E_X[\mathcal{H}_{SMT}] = E_X[2(MIX)^T MIX] = E_X[2(MX)^T MX] \tag{9}$$

In fact, $\boldsymbol{H}_{SMT}$ is the exact Hessian of the least square form $||MXw - MX\hat{w}||_2^2$, which is the square L2 Loss on the membrane potential of a spiking neuron with the input current Xw in the absence of spikes.

3.2. Comparing SBC with OBC

Naive implementation of OBC on SNN would have a loss function directly on the output of the Linear layer. It has the Hessian:

$$\boldsymbol{H}_{OBC} = E_X[\mathcal{H}_{OBC}] = E_X[2X^T X] \tag{10}$$

Recall exact Hessian Eq.8, $M\frac{\partial s}{\partial u} = Mh'$ always preserves the lower triangle structure (for spike trains with spikes) because M is lower triangular, and h' is diagonal. This means simplification $Mh' = I$ leading to $\mathcal{H} = 2(IX)^T IX$, i.e. $\boldsymbol{H}_{OBC}$, is weaker than keeping M, i.e. $\boldsymbol{H}_{SMT}$.

4. SBC Complexity and future works

We provide per-module SBC complexity here. Under unstructured pruning, the complexity of SBC is very similar to that of ExactOBS from OBC, with key differences: SBC's unstructured pruning uses d_{out} Hessian matrices, one for each output dimension in memory, instead of 1, to achieve true unordered, unstructured pruning. This modification also prepares our code and the algorithm

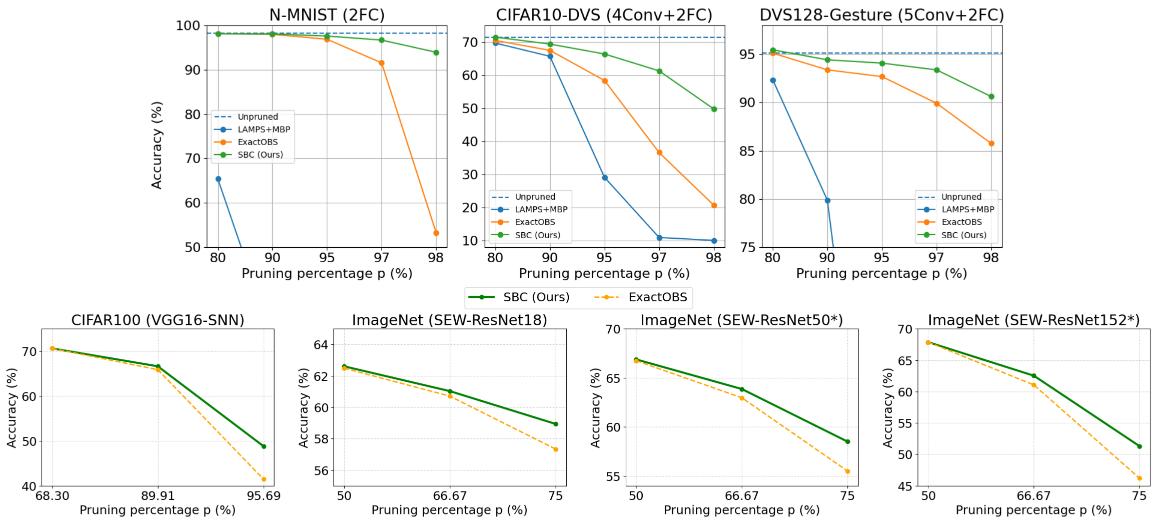

Figure 1: Neuromorphic (top row) and static (bottom row) dataset pruning results. Tables: Appx. F

for heterogeneous Hessians in the future. To summarise, per-module SBC has space complexity $\mathcal{O}(B\, d_{\text{in}}^2)$ and time complexity $\mathcal{O}(\frac{d_{out}}{B} d_{in}^3)$, with d_{in}, d_{out} refer to the input and output dimension of the linearized parameterized layer in the module. Batch size B is adjusted to trade time and space according to hardware availability. For detailed derivation, refer to Appendix C. SBC quantization will follow GPTQ [7] and has lower complexity than unstructured pruning.

In future work, we want to explore more sophisticated surrogate gradients g', Mh', or even a general $T \times T$ matrix in place of M in $\boldsymbol{H}_{SMP}$. These matrices can be customized to each LIF neuron based on properties of the neuron that do not change easily during compression, such as spike rate. These Hessians will be easy to calculate and can be used for unstructured pruning and quantization of large SNNs.

Another key property of our compression method is its ability to handle different membrane time constants τ within an LIF neuron layer. However, to the best of our knowledge, no mainstream training method exists for such LIF neuron layers. SBC can also handle compression of deep SNNs like various Spiking ResNets [6] and Spiking Transformer [23, 24] architectures. Appendix D provides implementation details.

5. Experimental Results

We conducted experiments with SNNs trained on neuromorphic datasets (N-MNIST [19], CIFAR10-DVS [3], and DVS128-Gesture [1]) and static datasets (CIFAR100 [13] and ImageNet [4]) to quantify the effectiveness of SBC empirically. Model and dataset details can be found in Appendix E.

SNNs can leverage unstructured pruning to reduce compute [21] on neuromorphic hardware, unlike ANNs on GPUs. This is why we focus the experiments on pruning. We use Layer-Adaptive sparsity for the Magnitude-based Pruning Score (LAMPS) [15] to determine per-layer pruning percentage from a global pruning target.

We compared the performance of the naive magnitude-based pruning (MBP) shown in LAMPS (only on neuromorphic datasets), the naive ExactOBS (OBC's pruning algorithm) implementation on SNN, and SBC with surrogate membrane potential. Note that all three methods pruned the same

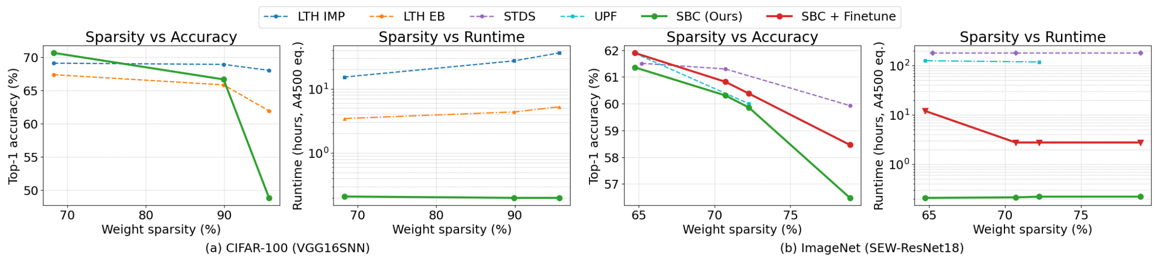

Figure 2: SBC (Ours) vs. PAT methods accuracies and runtimes. Tables: Appx. F

proportion of weights from each module; the difference is in how each module prunes its weights. Results are presented in Fig. 1.

In addition, we compared the accuracy and pruning time of SBC, SBC with post-pruning fine-tune, and SOTA SNN pruning-aware-training (PAT) algorithms (Lottery Ticket Hypothesis (LTH) [12], UPF [21], STDS [2]), with accuracies and pruning time recorded in Fig. 2.

5.1. Results and Analysis

Across neuromorphic datasets, SBC achieves the highest post-pruning accuracy of the three methods, and its advantage grows with sparsity; the same trend holds on larger static datasets.

Surprisingly, SBC outperforms LTH (+1.55%) at 68.30% sparsity, and Early Bird (EB) version of LTH (+0.82%) at 98.91% sparsity while providing 1 to 2 orders of magnitude pruning speed up. In ImageNet experiments, the SBC method is competitive with PAT methods and offers a 3-order-of-magnitude improvement in pruning time. With fine-tuning, SBC outperforms the accuracy of UPF methods, while falling short of the STDS results. These experiments demonstrate SBC as a cost-effective alternative to PAT methods, offering flexibility for further accuracy improvement through fine-tuning.

Notably, the drastically lower computation cost enables SBC to prune large models, such as SEW ResNet50/152, when PAT methods become too costly. To the best of our knowledge [2, 12, 21], this experiment is the first time SNNs with more than 34 layers have been pruned, and the first SNNs trained on an ImageNet-scale dataset with more than 19 layers have been pruned (Kim [12]'s experiment on Spiking ResNet-34 was trained on Tiny-ImageNet [14]). To the best of our knowledge, SEW-ResNet152 is also the largest and deepest SNN model that has been pruned unstructuredly by any algorithm to date. This reaffirms SBC as the SOTA one-shot post-training compression method for SNNs.

6. Conclusion and Future Works

In conclusion, this paper proposes a one-shot, efficient, post-training compression framework for SNNs, utilizing a second-order approximation of the per-layer spike train loss to compress and compensate for the compression dynamically. Through theoretical analysis and empirical demonstration, this paper shows SBC's effectiveness in compressing SNNs trained for neuromorphic and static datasets, outperforming current SOTA on SNN post-training one-shot methods, and competitive with the accuracy of iterative retraining compression methods. In future work, we will explore more sophisticated Hessians for spike train loss functions, as well as experiments on compressing even deeper/larger SNNs (e.g., Spikformers [25] and SpikeGPT [26]).

References

[1] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha. A low power, fully event-based gesture recognition system. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7388–7397, 2017.

[2] Yanqi Chen, Zhaofei Yu, Wei Fang, Zhengyu Ma, Tiejun Huang, and Yonghong Tian. State transition of dendritic spines improves learning of sparse spiking neural networks. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 3701–3715. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/chen22ac.html.

[3] Wensheng Cheng, Hao Luo, Wen Yang, Lei Yu, and Wei Li. Structure-aware network for lane marker extraction with dynamic vision sensor, 2020.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[5] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothee Masquelier, Tiejun Huang, and Yonghong Tian. Incorporating learnable membrane time constant to enhance learning of spiking neural networks, 2021.

[6] Wei Fang, Zhaofei Yu, Yanqi Chen, Tiejun Huang, Timothée Masquelier, and Yonghong Tian. Deep residual learning in spiking neural networks, 2022. URL https://arxiv.org/abs/2102.04159.

[7] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers, 2023.

[8] Elias Frantar, Sidak Pal Singh, and Dan Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning, 2023.

[9] Steve Furber, Chen Li, and Lei Ma. Quantization framework for fast spiking neural networks. *Frontiers in Neuroscience*, 16:1–13, July 2022. ISSN 1662-4548.

[10] Babak Hassibi and David Stork. Second order derivatives for network pruning: Optimal brain surgeon. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1992.

[11] Yangfan Hu, Huajin Tang, and Gang Pan. Spiking deep residual networks. *IEEE Transactions on Neural Networks and Learning Systems*, 34(8):5200–5205, 2023. doi: 10.1109/TNNLS.2021.3119238.

[12] Youngeun Kim, Yuhang Li, Hyoungseob Park, Yeshwanth Venkatesha, Ruokai Yin, and Priyadarshini Panda. Exploring lottery ticket hypothesis in spiking neural networks, 2022.

[13] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical Report 0, University of Toronto, Toronto, Ontario, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

[14] Ya Le and Xuan S. Yang. Tiny imagenet visual recognition challenge. 2015. URL https://api.semanticscholar.org/CorpusID:16664790.

[15] Jaeho Lee, Sejun Park, Sangwoo Mo, Sungsoo Ahn, and Jinwoo Shin. Layer-adaptive sparsity for the magnitude-based pruning, 2021.

[16] Yudong Li, Yunlin Lei, and Xu Yang. Spikeformer: A novel architecture for training high-performance low-latency spiking neural network, 2022. URL https://arxiv.org/abs/2211.10686.

[17] Hin Wai Lui and Emre Neftci. Hessian aware quantization of spiking neural networks, 2021.

[18] Emre O. Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks, 2019. URL https://arxiv.org/abs/1901.09948.

[19] Garrick Orchard, Ajinkya Jayawant, Gregory K. Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, Volume 9 - 2015, 2015.

[20] M. C. W. van Rossum. A novel spike distance. *Neural Computation*, 13(4):751–763, 2001.

[21] Xinyu Shi, Jianhao Ding, Zecheng Hao, and Zhaofei Yu. Towards energy efficient spiking neural networks: An unstructured pruning framework. In *The Twelfth International Conference on Learning Representations*, 2024.

[22] Wenjie Wei, Malu Zhang, Zijian Zhou, Ammar Belatreche, Yimeng Shan, Yu Liang, Honglin Cao, Jieyuan Zhang, and Yang Yang. Qp-snn: Quantized and pruned spiking neural networks, 2025.

[23] Man Yao, Jiakui Hu, Zhaokun Zhou, Li Yuan, Yonghong Tian, Bo Xu, and Guoqi Li. Spike-driven transformer, 2023.

[24] Zhaokun Zhou, Yuesheng Zhu, Chao He, Yaowei Wang, Shuicheng Yan, Yonghong Tian, and Li Yuan. Spikformer: When spiking neural network meets transformer, 2022.

[25] Zhaokun Zhou, Kaiwei Che, Wei Fang, Keyu Tian, Yuesheng Zhu, Shuicheng Yan, Yonghong Tian, and Li Yuan. Spikformer v2: Join the high accuracy club on imagenet with an snn ticket, 2024. URL https://arxiv.org/abs/2401.02020.

[26] Rui-Jie Zhu, Qihang Zhao, Guoqi Li, and Jason K. Eshraghian. Spikegpt: Generative pre-trained language model with spiking neural networks, 2024. URL https://arxiv.org/abs/2302.13939.

Appendix A. Visualization of modules in a typical SNN

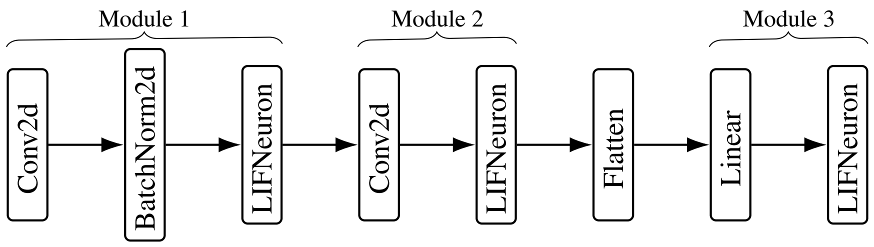

Figure 3: Example modules in a sample SNN. Each module ends with a LIF neuron layer and includes the linear, convolution, or batch normalization layers before it.

Appendix B. Deriving exact Hessian for individual LIF neuron

We first derive the first-order derivative of loss function $\mathcal{L}$ against weight w_i, with key assumption on optimality of w, Hessian invariance under perturbation, and differentiability of spike train s w.r.t to membrane potential u.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_i} = \frac{\partial \mathcal{L}}{\partial s}\frac{\partial s}{\partial w_i} = \frac{||Ms - M\hat{s}||_2^2}{\partial s}\frac{\partial s}{\partial w_i} \\ = 2(Ms - M\hat{s})^T M \frac{\partial s}{\partial w_i}\end{aligned} \tag{11}$$

As is customary in OBS framework, with a well-trained neural network, we assume $\nabla_{w_j}\mathcal{L} = 0$ for all weights w_i. Now we can get the second order derivative:

$$\begin{aligned}\mathcal{H}_{ij} &= \frac{\partial^2 \mathcal{L}}{\partial w_i \partial w_j} \\ &= \frac{\partial}{\partial w_j}(2(Ms - M\hat{s})^T M \frac{\partial s}{\partial w_i}) \quad \text{(by (11))} \\ &= 2[\frac{\partial}{\partial w_j}(a^T b)] \quad \text{where } a = (Ms - M\hat{s}), b = M\frac{\partial s}{\partial w_i} \\ &= 2[(\frac{\partial a}{\partial w_j})^T b + a^T(\frac{\partial b}{\partial w_j})] \quad \text{(product rule)} \\ &= 2[(\frac{\partial (Ms - M\hat{s})}{\partial w_j})^T M \frac{\partial s}{\partial w_j} + (Ms - M\hat{s})^T \frac{\partial^2 Ms}{\partial w_i \partial w_j}] \quad \text{expand a and b} \\ &= 2[(\frac{\partial (Ms)}{\partial w_j})^T M \frac{\partial s}{\partial w_j} + (Ms - M\hat{s})^T M \frac{\partial^2 s}{\partial w_i \partial w_j}] \\ &= 2(M\frac{\partial s}{\partial w_j})^T M \frac{\partial s}{\partial w_i} + 2(Ms - M\hat{s})^T M \frac{\partial^2 s}{\partial w_i \partial w_j}\end{aligned} \tag{12}$$

We can further derive $\frac{\partial s}{\partial w_i}$ and $\frac{\partial^2 s}{\partial w_i \partial w_j}$:

$$\begin{aligned}\frac{\partial s}{\partial w_i} &= \frac{\partial s}{\partial u}\frac{\partial u}{\partial w_i} = \frac{\partial s}{\partial u}x_i \\ \frac{\partial^2 s}{\partial w_i \partial w_j} &= \frac{\partial}{\partial w_j}(\frac{\partial s}{\partial u}x_i) = x_i\frac{\partial}{\partial w_j}(\frac{\partial s}{\partial u}) = x_i[\frac{\partial}{\partial u}(\frac{\partial s}{\partial u})]\frac{\partial u}{\partial w_j} = x_i\frac{\partial^2 s}{\partial u^2}x_j\end{aligned} \tag{13}$$

Where x_i, x_j denotes time-varying pre-synaptic spike trains to synaptic connections w_i and w_j, i.e. $x_i = X_{:,i}$. We can now further simplify the exact Hessian Eq. 12:

$$\mathcal{H}_{ij} = 2(M\frac{\partial s}{\partial u}x_j)^T M\frac{\partial s}{\partial u}x_i + 2(Ms - M\hat{s})^T M x_i \frac{\partial^2 s}{\partial u^2}x_j \tag{14}$$

Under the key OBS assumption that Hessian H remains unchanged under the weight perturbation, the second half of Eq. 14 can be removed as $(Ms - M\hat{s}) = 0$ at $s = \hat{s}$. We can now write the full simplified Hessian:

$$\mathcal{H} = 2(M\frac{\partial s}{\partial u}X)^T M\frac{\partial s}{\partial u}X \tag{15}$$

Appendix C. SBC complexity analysis

C.0.1. THE SBC PRUNING ALGORITHM

Here, we introduce the module-wise unstructured pruning framework for SNNs, utilizing SMP. We start by obtaining a per-module pruning target with Layer-Adaptive sparsity for the Magnitude-based Pruning Score (LAMPS) [15] to determine per-module pruning percentage from a global pruning target. LAMPS takes $O(d \cdot log(d))$, where d represents the number of trainable parameters in the SNN.

Step 1: Weight ordering. To efficiently determine the pruning order of weights in each module, we record each weight's loss by performing OBS per neuron with a small batch size B_{in}. Each iteration, we take weights with B_{in} smallest loss and prune them together, then update the inverse Hessian with Woodbury Matrix Identity [10]. The per-neuron full algorithm is given in Algorithm 1. For small B_{in}, this takes $O(\frac{d_{\text{in}}^3}{B_{in}})$ time and $O(d_{in})$ space.

In actuality, we can batch B_{out} neurons in parallel, given the available GPU resources. This produces a loss for each weight in the module, which we then sort and create a mask $\mathbf{M}$ of pruned weights according to the module pruning target. This takes $O(\frac{d_{out}}{B_{out}} \cdot \frac{d_{in}^3}{B_{in}})$ time and $O(B_{out} \cdot d_{in}^2)$ space.

Step 2: Weight pruning. With a mask $\mathbf{M}$, we can directly update weights of a neuron according to its local set of weights to remove $\mathbb{P} = \mathbf{M}_{:,i}$ via the group OBS formula $\delta_{\mathbb{P}} = -\mathbf{H}^{-1}_{:,\mathbb{P}}((\mathbf{H^{-1}})_{\mathbb{P}})^{-1}\mathbf{W}_{:,i}$. This takes $O(\frac{d_{out}}{B_{out}} \cdot d_{in}^3)$ time and $O(B_{out} \cdot d_{in}^2)$ space, but it is generally faster than the weight ordering step.

Complexity. To summarise, per-module SBC has space complexity $O(B_{out} \cdot d_{\text{in}}^2)$ and time complexity $O(\frac{d_{out}}{B_{out}} \cdot \frac{d_{in}^3}{B_{in}})$, with d_{in}, d_{out} refer to the input and output dimension of the linearized parameterized layer in the module. Batch sizes B_{in} and B_{out} are adjusted to balance time and space according to hardware availability.

Algorithm 1 Losses $\mathbf{L}$ for weights $\mathbf{w}$ of neuron with $\mathbf{H}^{-1} = (2(\mathbf{MX})^\top\mathbf{MX})^{-1}$, in $O(\frac{d_{\text{in}}^3}{B_{in}})$ time.

```
$\mathbb{M} \leftarrow \{1, \dots, d_{\text{in}}\}$
$\mathbf{L} \leftarrow \{\}$
for $i = 1, 1 + B_{in}, \dots, d_{in}$ do
    $s_p \leftarrow \frac{w_p^2}{[\mathbf{H}^{-1}]_{pp}} \ \forall p \in \mathbb{M}$
    $\mathbb{P} \leftarrow$ indices of the $B_{\text{in}}$ smallest $\{s_p\}_{p \in \mathbb{M}}$
    $\mathbf{L}[p] \leftarrow s_p \ \ \forall p \in \mathbb{P}$
    $\mathbf{w_P} \leftarrow \mathbf{w}_p \ \ \forall p \in \mathbb{P}$
    $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{H}^{-1}_{:,\mathbb{P}}((\mathbf{H}^{-1})_{\mathbb{P}})^{-1} \cdot \mathbf{w}_{\mathbb{P}}$
    $\mathbf{H}^{-1} \leftarrow \mathbf{H}^{-1} - \mathbf{H}^{-1}_{:,\mathbb{P}}((\mathbf{H}^{-1})_{\mathbb{P}})^{-1}\mathbf{H}^{-1}_{\mathbb{P},:}$
    $\mathbb{M} \leftarrow \mathbb{M} \setminus \{\mathbb{P}\}$
end for
```

Algorithm 2 SBC Pruning, with SNN $\mathcal{M}$, calibration data $\mathcal{X}$, model sparsity target $s \in [0, 1]$

```
$\mathbb{M} \leftarrow$ modules($\mathcal{M}$)          ▷ get set of modules
for all $\mathbf{m} \in \mathbb{M}$ do
    $s_m$ = LAMPS($\mathcal{M}, s, \mathbf{m}$)  ▷ module target
    $\mathbb{X} \leftarrow$ module $\mathbf{m}$ input data from $\mathcal{X}, \mathcal{M}$
    $\mathbf{H} = \frac{2}{|\mathbb{X}|}\sum_{X \in \mathbb{X}}(MX)^T MX$
    $\mathbf{W} \leftarrow$ weights of $\mathbf{M}$
    $\mathbf{L} \leftarrow$ Algorithm 1$(\mathbf{W}, \mathbf{H}^{-1})$
    $\mathbf{M} \leftarrow$ indicies of $|\mathbf{W}| \cdot s_{\mathbf{m}}$ smallest $\mathbf{L}$
    for $i = 1, \dots d_{out}$ do
        $\mathbb{P} \leftarrow \mathbf{M}_{:,\mathbf{i}}$
        $\delta_{\mathbb{P}} \leftarrow -\mathbf{H}^{-1}_{:,\mathbb{P}}((\mathbf{H}^{-1})_{\mathbb{P}})^{-1}\mathbf{W}_{:,i}$
        $\mathbf{W}_{:,i} \leftarrow \mathbf{W}_{:,i} + \delta_{\mathbb{P}}$
    end for
end for
```

Appendix D. Derivation on how to apply SBC to various Spiking ResNets and Spiking Transformer architectures

D.1. Spiking ResNets

Shortcut handling in Spiking ResNets We derive the SBC loss with surrogate membrane potential (SMP) for two popular Spiking ResNet variants: Spiking-Element-Wise (SEW) ResNet [6] and SpikingResNet [11]. In SEW ResNet, the shortcut follows the spiking layer; from SBC's perspective, the two branches are already independent modules and can be pruned separately.

In SpikingResNet, the shortcut precedes the spiking layer. Thus the external current to the next spiking layer is the sum of the convolution output and the shortcut output. Consider one neuron with linearized convolution output X_1w_1, shortcut output X_2w_2, and surrogate membrane-potential operator M. The shortcut loss is

$$L_{\text{shortcut}} = \mathbb{E}_X\left[\|M(X_1w_1 + X_2w_2) - M(X_1\hat{w}_1 + X_2\hat{w}_2)\|_2^2\right] \tag{16}$$

$$= \mathbb{E}_X\left[\left\|M\left(\begin{bmatrix} X_1 & X_2 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}\right) - M\left(\begin{bmatrix} X_1 & X_2 \end{bmatrix}\begin{bmatrix} \hat{w}_1 \\ \hat{w}_2 \end{bmatrix}\right)\right\|_2^2\right]. \tag{17}$$

Hence we may concatenate the inputs and weights of the last convolution and the shortcut in each SpikingResNet block, forming a new module with input $X = \begin{bmatrix} X_1 & X_2 \end{bmatrix}$ and weights $W = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$. Compression is then performed on this module. If the shortcut has no trainable parameters, we simply mask out the weights associated with X_2 before compression. *Notation:* $\begin{bmatrix} X_1 & X_2 \end{bmatrix}$ denotes horizontal concatenation; $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ denotes vertical concatenation.

D.2. Spiking Transformers

This section examines the applicability of **SBC** to popular *spiking–transformer* architectures, and found that it extends cleanly to Spikformer [16], Spikformer V2 [25], and the Spike-Driven Transformer (SDT) [23].

Spikformer family. In Spikformer, the weights of the Spiking Self-Attention (SSA) layers (W_Q, W_K, W_V) belong to a Linear→BN→Spiking block and can therefore be treated as *three* SBC modules. Each MLP block follows Linear → BN → Spiking → Linear → BN → Spiking, giving *two* additional modules. Spikformer V1 uses Spiking Patch Splitting (SPS), whereas V2 employs a Spiking Convolution Stem (SCS); both are sequential convolutions that fit the SBC module-wise compression definition in Section 3.

Spike-Driven Transformer (SDT). In SDT, the shortcut precedes the spiking layer, so the external current is the sum of the previous BN output and the shortcut. Let the main-path output be XW, the shortcut output U, and M the surrogate membrane-potential operator. For a single spiking layer, the shortcut loss is

$$\begin{aligned} L_{\text{shortcut}} &= \mathbb{E}_X\big[\|M(XW+U) - M(X\hat{W}+U)\|_2^2\big] \\ &= \mathbb{E}_X\big[\|MXW + MU - MX\hat{W} - MU\|_2^2\big] \\ &= \mathbb{E}_X\big[\|MXW - MX\hat{W}\|_2^2\big]. \end{aligned} \tag{18}$$

Hence, the shortcut can be ignored during compression, allowing all SDT modules to be treated exactly as in Spikformer.

Appendix E. Datasets and SNN model details

Table 1: Datasets and SNN Architectures used for Compression

Dataset	Architecture	Input Shape	τ	T	Accuracy(%)
N-MNIST	2FC	34x34x2	2.0	100	98.31
CIFAR10-DVS	4Conv+2FC	64x64x2	2.0	20	71.50
DVS128-Gesture	5Conv+2FC	128x128x2	2.0	20	95.14
CIFAR100	VGG16SNN	32x32x3	1.33	5	71.05
ImageNet	SEW-ResNet Family	256x256x3	$+\infty$	4	63.12-69.18

Table 1 provides information on the datasets and models used for experiments. All three neuromorphic datasets (N-MNIST, CIFAR10-DVS, DVS128-Gesture) are trained with the Adam optimizer using $lr = 0.001$. N-MNIST was trained for 200 epochs, CIFAR10-DVS, and DVS128 were trained with Cosine Annealing for 512 epochs. The CIFAR-100 model was trained with 300 epochs, with steps at 150 and 225, using an initial learning rate of $lr_{initial} = 0.1$. SEW-ResNet family models checkpoints are retrieved from its GitHub repository.

Appendix F. Experiment Result Tables

Table 2: Neuromorphic dataset pruning result

Setting	Method	Accuracy(%)					
		$p = 0\%$	80.00%	90.00%	95.00%	97.00%	98.00%
N-MNIST 2FC	LAMPS+MBP	98.31	65.39	29.40	35.13	26.94	25.60
	ExactOBS	98.31	**98.16**	98.01	96.91	91.59	53.24
	SBC(Ours)	98.31	98.13	**98.10**	**97.63**	**96.72**	**93.97**
CIFAR10-DVS 4Conv+2FC	LAMPS+MBP	71.50	69.70	65.70	29.10	10.90	10.00
	ExactOBS	71.50	70.50	67.50	58.40	36.60	20.70
	SBC(Ours)	71.50	**71.50**	**69.40**	**66.40**	**61.30**	**49.80**
DVS128-Gesture 5Conv+2FC	LAMPS+MBP	95.14	92.36	79.86	35.13	9.03	8.33
	ExactOBS	95.14	95.13	93.40	92.7	89.93	85.76
	SBC(Ours)	95.14	**95.48**	**94.44**	**94.10**	**93.40**	**90.63**

Table 3: Static dataset pruning results: SBC vs. ExactOBS

Dataset	Arch.	Time step	Top-1 Acc. **SBC(Ours)** (%)	Top-1 Acc. ExactOBS (%)	Weight Sparsity (%)	Time (h) A4500 Eq.	Calib. size (% train)
CIFAR100	VGG16-SNN	5	**70.63**	**70.63**	68.30	0.21	20%
			66.64	65.89	89.91	0.20	20%
			48.81	41.46	95.69	0.20	20%
ImageNet	SEW-ResNet18	4	**62.61**	62.49	50.00	0.208	2%
			61.03	60.71	66.67	0.214	2%
			58.94	57.34	75.00	0.220	2%
	SEW-ResNet50†	4	**66.88**	66.75	50.00	0.346	2%
			63.86	62.97	66.67	0.344	2%
			58.50	55.50	75.00	0.341	2%
	SEW-ResNet152†	4	**67.88**	**67.88**	50.00	0.783	2%
			62.53	61.06	66.67	0.808	2%
			51.30	46.19	75.00	0.723	2%

* **Bolded** entries represent best accuracies in sparsity class

† Deepest SNN models pruned to date. No existing PAT methods have pruned them.

Table 4: Static dataset pruning results, SBC compared with Pruning-Aware-Training methods

Dataset / Model	Time step	Pruning Method	Top-1 Acc. (%)	Weight Sparsity (%)	Time (h) A4500 Eq.	Calib. size (% train)	Finetune epochs
CIFAR100 VGG16-SNN	5	LTH IMP[12]	69.08	68.30	15.23	–	–
			68.90*	89.91	27.22	–	–
			68.00*	95.69	36.22	–	–
		LTH EB[12]	67.35	68.30	3.44	–	–
			65.82	89.91	4.36	–	–
			61.90	95.69	5.24	–	–
		SBC (Ours)	**70.63***	68.30	**0.21**	20%	–
			66.64	89.91	**0.20**	20%	–
			48.81	95.69	**0.20**	20%	–
ImageNet SEW-ResNet18	4	STDS[2]	61.51	65.21	180	–	–
			61.30*	70.71	180	–	–
			59.93*	78.92	180	–	–
		UPF[21]	61.89	64.74	125	–	–
			60.00	72.26	118	–	–
		SBC (Ours)	61.36	64.74	**0.208**	2%	–
			60.31	70.71	**0.213**	2%	–
			59.86	72.26	**0.220**	2%	–
			56.48	78.92	**0.220**	2%	–
		SBC + Finetune	**61.90(+0.54†)***	64.74	12.00	2%	5.0
			60.82(+0.51†)	70.71	2.75	2%	1.0
			60.38(+0.52†)*	72.26	2.75	2%	1.0
			58.46(+1.98†)	78.92	2.75	2%	1.0

* Best accuracies in sparsity class

† Accuracy improvements compared to pre-finetuned, SBC pruned models