Towards Ultra Low Latency Spiking Neural Networks for Vision and Sequential Tasks Using Temporal Pruning

Sayeed Shafayet Chowdhury[1], Nitin Rathi[1], and Kaushik Roy[1]

Prurdue University, West Lafayette IN 47907, USA
{chowdh23, rathi2, kaushik}@purdue.edu

Abstract. Spiking Neural Networks (SNNs) can be energy efficient alternatives to commonly used deep neural networks (DNNs). However, computation over multiple timesteps increases latency and energy and incurs memory access overhead of membrane potentials. Hence, latency reduction is pivotal to obtain SNNs with high energy efficiency. But, reducing latency can have an adverse effect on accuracy. To optimize the accuracy-energy-latency trade-off, we propose a temporal pruning method which starts with an SNN of T timesteps, and reduces T every iteration of training, with threshold and leak as trainable parameters. This results in a continuum of SNNs from T timesteps, all the way up to unit timestep. Training SNNs directly with 1 timestep results in convergence failure due to layerwise spike vanishing and difficulty in finding optimum thresholds. The proposed temporal pruning overcomes this by enabling the learning of suitable layerwise thresholds with backpropagation by maintaining sufficient spiking activity. Using the proposed algorithm, we achieve top-1 accuracy of 93.05%, 70.15% and 69.00% on CIFAR-10, CIFAR-100 and ImageNet, respectively with VGG16, in just 1 timestep. Note, SNNs with leaky-integrate-and-fire (LIF) neurons behave as Recurrent Neural Networks (RNNs), with the membrane potential retaining information of previous inputs. The proposed SNNs also enable performing sequential tasks such as reinforcement learning on Cartpole and Atari pong environments using only 1 to 5 timesteps.

Keywords: Spiking Neural Networks, Unit timestep, Energy efficiency, Temporal pruning, Reinforcement learning

1 Introduction

Deep neural networks (DNNs) have revolutionized the fields of object detection, classification and natural language processing [21,14,5]. However, such performance boost comes at the cost of extremely energy intensive DNN architectures [23]. Therefore, edge deployment of such DNNs remains a challenge. One approach to counter this is to use bio-inspired Spiking Neural Networks (SNNs) [25,33], which perform computations using spikes instead of analog activations used in standard networks. In this paper, standard networks are referred to as

Artificial Neural Networks (ANNs) in contrast to SNNs having spike activation. As such, the sparse event-driven nature of SNNs makes them an attractive alternative to ANNs [9].

The applicability of SNNs was initially limited due to the unavailability of suitable training algorithms. At first, ANN-SNN conversion methods [7,36] were adopted, but incurred high latency. Recently, direct training using surrogate gradients [27,44] has resulted in low latency. Most of the commonly used SNNs use Poisson rate-coding [7,36] where a large number of timesteps[1] is usually required to obtain high performance [36,11]. However, multi-timestep processing causes twofold challenges in SNNs - (i) too long a latency might be unsuitable for real-time applications, (ii) the need for accumulation of membrane potential (V_{mem}) over numerous timesteps results in higher number of operations, thereby reducing efficiency. Moreover, in contrast to ANNs, additional memory is required to store the intermediate V_{mem} and memory access cost is incurred for fetching V_{mem} at each timestep. Note, the memory access cost can be significantly higher compared to floating point add operations [12]. So, reducing inference latency is critical for widespread deployment of SNNs.

To leverage the full potential of SNNs, we propose a temporal pruning technique to obtain a continuum of SNNs optimizing the associated accuracy-energy-latency trade-offs. SNNs, unlike ANNs have a temporal dimension. As a result, low latency SNNs might be obtained if suitable compression can be performed along the temporal axis, which we refer to as 'temporal pruning' here. Such temporal compression is required since training SNNs directly with very few timesteps results in convergence failure due to significant decay of spiking activity in the deeper layers. To infer with very low latency, sufficient spikes must be propagated till the final layer in only a few forward passes. To achieve that, the layerwise neuron thresholds and leaks must be adjusted properly. The optimum approach to set the thresholds and leaks is learning them using backpropagation (BP). However, without having enough spikes at the output, the optimization gets stuck and no learning can occur for extremely low latency. To circumvent this issue, we adopt a temporal pruning method which starts with an SNN of T (T>1) timesteps, and gradually reduces T at every iteration of training using threshold and leak as trainable parameters alongside the weights. At each stage of timestep reduction, the network trained at previous stage with higher timestep is used as initialization for subsequent training with lower timestep. Using such a pruning process, we obtain a continuum of SNNs, starting from T timesteps, eventually leading up to unit timestep. Note, under unit timestep, if a neuron receives an input, it updates V_{mem} and in the event of crossing a given threshold, it outputs a spike in a single shot, similar to binary activated ANNs [35,1].

As mentioned, the proposed temporal pruning starts from an initial SNN trained for T timesteps. We obtain this initial SNN using a hybrid training method [32]. First an ANN is trained, followed by ANN-SNN conversion, and SNN training with surrogate gradient based backpropagation [27]. The initial design using the above approach leads to an SNN with T=5 [31] and our proposed

[1] 1 timestep is defined as the time taken to perform 1 forward pass through the network

approach reduces the timestep to 1. We use direct input encoding [34], with the first convolutional layer of the network acting as spike generator. We evaluate the performance of the proposed method on image classification and achieve top-1 accuracy of 93.05%, 70.15% and 69.00% on CIFAR-10, CIFAR-100 and ImageNet, respectively with VGG16, in just 1 timestep. These results demonstrate that for static vision tasks, SNNs can perform comparable to ANNs using single shot inference. However, a key distinction between ANNs and SNNs is the time axis which can potentially enhance the performance of SNNs for sequential tasks if the temporal information can be leveraged suitably using the membrane potential of neurons. To validate this hypothesis, we apply our technique to SNN-based reinforcement learning (RL) agents on Cartpole and Atari pong environments. Though the proposed method can provide RL agents with unit timestep, performance improves significantly for SNNs with larger timesteps. For cartpole, we obtain mean reward of 38.7 and 52.2 for 1 and 3 timesteps, respectively. Similarly, the mean reward increased from 17.4 to 19.4 as the timesteps are increased from 1 to 5 for Atari pong. It is noteworthy that even unit timestep SNNs can handle dynamic inputs. However, increasing timesteps leads to enhancement in performance for sequential tasks. Overall, the proposed continuum of SNNs is able to provide optimum solutions for both static and dynamic tasks. While SNNs with unit timestep obtain satisfactory performance with highest efficiency for static inputs, choosing SNNs having only a few timesteps from the continuum of SNNs can provide ANN-like performance for dynamic RL tasks with much lower energy cost. To summarize, the main contributions of this work are-

- We present a temporal pruning technique to obtain a continuum of SNNs with varying latency, starting from T timesteps up to 1. To the best of our knowledge, this is the first SNN work to achieve competitive classification performance (top-1 accuracy of 69%) on ImageNet using unit timestep.
- Our approach does not incur the memory access cost of accumulated membrane potentials, unlike previously proposed SNNs.
- One timestep SNNs infer with up to 5X lower latency compared to state-of-the-art SNNs, while achieving comparable accuracy. This also leads to efficient computation, resulting in SNNs which are up to 33X more energy efficient compared to ANNs with equivalent architecture.
- The proposed method enables deep-Q reinforcement learning on Cartpole and Atari Pong with SNNs having few (1-5) timesteps, thereby showing the efficacy of multi-timestep SNNs (as RNNs) for sequential tasks. Compared to unit timestep, 3-5 timesteps enhance the performance of SNNs considerably for such tasks by leveraging the inherent recurrence of spiking neurons.

2 Related Works

ANN-SNN Conversion. A widely used approach for training deep SNNs involves training an ANN and converting to SNN for fine-tuning [2,7,36]. Proper layerwise threshold adjustment is critical to convert ANNs to SNNs successfully.

One approach is choosing the thresholds as the maximum pre-activation of the neurons [36]. While it provides high accuracy, the associated drawback is high inference latency (about 1000 timesteps). Alternatively, [34] suggests choosing a certain percentile of the pre-activation distribution as the threshold to reduce latency. However, these methods [34,11] still require few hundred timesteps.

Backpropagation from Scratch and Hybrid Training. An alternate route of training SNNs with reduced latency is learning from scratch using backpropagation (BP). To circumvent the non-differentiability of the spike function, surrogate gradient based optimization has been proposed [27] to implement BP in SNNs [16,22]. A related approach is using surrogate gradient based BP on membrane potential [47,38]. Overall, these approaches obtain SNNs with high accuracy, but the latency is still significant ($\sim$100-125 timesteps). Recently, surrogate gradient-based deep residual learning has been employed to directly train SNNs with just 4 timesteps [8]. A hybrid approach is proposed in [32] where a pre-trained ANN is used as initialization for subsequent SNN learning. Such a hybrid approach improves upon conversion by reducing latency and speeds up convergence of direct BP from scratch method.

Temporal Encoding. Temporal coding schemes such as phase [18] or burst [28] coding attempt to capture temporal information into learning; a related method is time-to-first-spike (TTFS) coding [29], where each neuron is allowed to spike just once. While these techniques enhance efficiency by reducing the spike count, issues regarding high latency and memory access overhead persist.

Direct Encoding. The analog pixels are directly applied to the 1^{st} layer of the network in direct encoding [34,31,49]. Using direct coding and utilizing the first layer as spike generator, authors in [31] achieve competitive performance on ImageNet with 5 timesteps. Threshold-dependent batch normalization is employed with direct encoding by [49] to obtain high performing SNNs on ImageNet with 6 timesteps. Inspired by such performance, we adopt the direct encoding method. The difference between our work and [31] is the temporal pruning aspect, which enables to reduce the timestep to lowest possible limit while maintaining performance on complex datasets. This was infeasible for state-of-the-art SNNs [8,31,49], even with direct encoding. Moreover, our approach is able to enhance performance by incorporating batch-normalization unlike [31,32].

Binary Neural Networks. Unit timestep SNN is closely related to binary neural networks (BNN) [30,40], as both infer in a single shot with binary activation. However, there are quite a few distinctions. While BNNs binarize the outputs as ± 1, SNNs give spike (i.e., $\{0, 1\}$) output, which leads to higher sparsity. Additionally, our model uses LIF neurons with trainable thresholds, whereas, BNNs employ a Heaviside activation where the firing threshold is zero. The activation of the proposed SNN reduces to Heaviside function with tunable threshold for $T \rightarrow 1$. However, the use of LIF neurons enables us to use the same model for sequential processing using V_{mem}, which is non-trivial in BNNs.

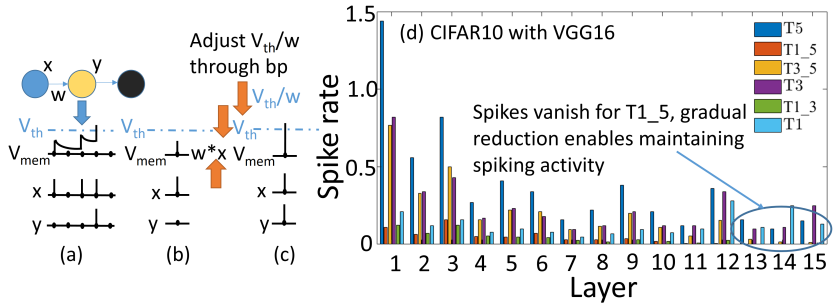


Fig. 1. (a) Schematic of an SNN with dynamics shown for yellow neuron with x as input and y as output, (b) No output spike for direct transition from 5 to 1 timestep, (c) Output spike in just 1 timestep through V_{th}/w lowering, (d) Layerwise spike rates; Tx represents an SNN trained with 'x' timesteps; Tx_y represents an SNN trained with 'x' timesteps but initialized with an SNN that was trained for 'y' timesteps.

3 Background

Spiking Neuron Model. The LIF neuron model [17] is described as-

$$\tau_m \frac{dU}{dt} = -(U - U_{rest}) + RI, \quad U \leq V_{th} \tag{1}$$

where U, I, τ_m, R, V_{th} and U_{rest} denote membrane potential, input, time constant for membrane potential decay, leakage resistance, firing threshold and resting potential, respectively. We employ a discretized version of Eqn. 1-

$$u_i^t = \lambda_i u_i^{t-1} + \sum_j w_{ij} o_j^t - v_i o_i^{t-1}, \tag{2}$$

$$z_i^{t-1} = \frac{u_i^{t-1}}{v_i} \quad \text{and} \quad o_i^{t-1} = \begin{cases} 1, & \text{if } u_i^{t-1} > v_i \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where u is the membrane potential, subscripts i and j represent the post and pre-neuron, respectively, t denotes timestep, λ is the leak constant= $e^{\frac{-1}{\tau_m}}$, w_{ij} represents the weight between the i-th and j-th neurons, o is the output spike, and v is the threshold. The detailed methodology of training SNN with a certain timestep is provided in supplementary section 1. In particular, Algorithm S1 of supplementary depicts the training scheme for one iteration.

4 Proposed Latency Reduction Method

We start by training an ANN with batch-norm (BN) and subsequently the BN parameters are fused with the layerwise weights as done in [34]. With such pretrained ANN, the weights are copied to an iso-architecture SNN and we

select the 90.0 percentile of the pre-activation distribution at each layer as its threshold. Then the SNN is trained for T timesteps using BP which serves as our baseline; training steps are detailed in Algorithm S1 of supplementary. Our starting point is a T=5 timesteps trained network, since 4-6 is the minimum latency range that state-of-the-art (SOTA) SNNs have reported for ImageNet training [8,31,49] with high performance. However, our method is generic and can be applied to SNNs with higher T as starting point too, but that increases training overhead which motivates our choice of starting point as T=5.

Direct Inference with Unit Timestep. As mentioned in Section 1, our goal is to reduce latency of SNNs as much as possible. To that effect, next we explore the feasibility of directly reducing latency from T=5 to 1. Fig. 1(a) schematically depicts an SNN with 3 neurons (one per layer), we focus on the yellow neuron. Suppose it receives x as input and y is its output. With the weight (w) and threshold (V_{th}) trained for 5 timesteps, there is enough accumulation of membrane potential (V_{mem}) to cross V_{th} and propagate spikes to next layer within that 5 timestep window. However, when we try to infer with 1 timestep (Fig. 1(b)), there is no output spike as the V_{mem} is unable to reach V_{th} instantly. Therefore, w and V_{th} need to be adjusted such that information can propagate even within 1 step. Balancing the thresholds properly is critical for SNNs to perform well [49]. Hence, our goal is adjusting V_{th}/w through learning using BP, so that only neurons salient for information propagation are able to spike in 1 timestep (Fig. 1(c)), while other neurons remain dormant.

Direct Transition to Training with Unit Timestep. Next, we begin training with 1 timestep initialized with the 5 timestep trained SNN. However, the network fails to train. To investigate this, we plot the layerwise spike rates for a VGG16 on CIFAR10 in Fig. 1(d). Here, with Tx denotes an SNN trained with spike based back propagation for x timesteps, and Tx_y denotes an SNN trained with x timesteps, starting from an initial SNN that was trained for y timesteps ($y > x$). While T5 has sufficient spiking activity till final layer, for T1_5, spikes die out in the earlier layers. Due to such spike vanishing, all outputs at the deeper layers (layers 11-16) are 0, thus BP fails to start training. This happens since the initial V_{th} and w are trained for 5 timesteps, and retraining directly for 1 timestep hinders spike propagation to the later layers.

Gradual Temporal Pruning. To mitigate the above issue, we propose a gradual latency reduction approach. We observe that despite significant layer-wise spike activity decay, some spikes still reach the final layer for T3_5. Hence, we start training with 3 timesteps using T5 as initialization and training is able to converge through BP. The spiking activity is recovered when learning converges as shown in Fig. 1(d), case T3. Subsequently, we train a network with just 1 timestep by initializing it with T3, and successfully attain convergence. The results for this case is shown as T1 in Fig. 1(d). Motivated by these observations, we propose an iterative temporal pruning method which enables training a continuum of SNNs starting from T timesteps (T> 1) up to 1. Since in our case, we compress the temporal dimension of SNNs, it is termed as 'temporal

Algorithm 1 Pseudo-code of training.

Input: Trained SNN with N timesteps (TN), timesteps reduction step size (b), number of epochs to train (e)
Initialize: new SNN initialized with trained parameters of TN, reduced latency $T_r = N - b$
while $T_r > 0$ **do**
  // **Training Phase**
  for *epoch* $\leftarrow 1$ **to** e **do**
    //Train network with T_r timesteps using algorithm S1 of supplementary
  end for
  // Initialize another iso-architecture SNN with parameters of above trained network
  // Temporal pruning
  $T_r = T_r - b$
end while

pruning'. A pseudo-code of training is given in Algorithm 1. Beginning with T5, we gradually reduce the latency by 1 at each step and train till convergence.

How does temporal pruning help in learning? In this section, we investigate the effect of the temporal pruning on SNN learning. Without temporal pruning, spikes may not propagate to the final layers in few timesteps. As a result, no gradient can be propagated back to train the SNN. In terms of the optimization landscape, the neural network parameters remain stuck at their initialization point since the gradient updates remain zero. However, using gradual temporal pruning, we facilitate the propagation of spikes (and suitable gradient flow is enabled) by properly learning the threshold and leak parameters, leading to training convergence with very few timesteps. This can be visualized as a form of curriculum training, where the SNN is first trained with a comparatively easier learning case (higher T) and then more stringent constraints (training with lower T) are incorporated gradually. Similar curriculum training has been applied for ANNs [20] where directly imposing a tougher learning scenario leads to optimization failure; however, gradual training alleviates the problem. The effect of temporal pruning can also be analyzed by comparing it to spatial pruning in ANNs [13,12]. In case of spatial pruning, a more complex network is first trained and then spatial compression is performed while maintaining performance. Our proposed approach is similar with the exception that we leverage the time axis of SNNs to perform pruning. SNNs with multiple timesteps are trained with BP through time (BPTT) [27], like RNNs. If we unroll SNNs in time, it becomes obvious that each timestep adds a new hidden state. In essence, temporal pruning helps latency reduction in SNNs through compression. From a related perspective, we can also perceive gradual temporal pruning as training by generations. Similar sequential compression (training by generations) has been implemented in ANNs [10,46] to obtain better performing compressed networks.

Table 1. Top-1 classification accuracy (%), Tx denotes SNN trained with 'x' timestep

Architecture	Dataset	ANN	T5	T4	T3	T2	T1
VGG6	CIFAR10	91.59	90.61	90.52	90.40	90.05	89.10
VGG16	CIFAR10	94.10	93.90	93.87	93.85	93.72	93.05
ResNet20	CIFAR10	93.34	92.62	92.58	92.56	92.11	91.10
VGG16	CIFAR100	72.46	71.58	71.51	71.46	71.43	70.15
ResNet20	CIFAR100	65.90	65.57	65.37	65.16	64.86	63.30
VGG16	ImageNet	70.08	69.05	69.03	69.01	69.00	69.00

5 Experiments and Results

Datasets and Models. We perform experiments on CIFAR10, CIFAR100 and ImageNet using VGG16 and ResNet20, with some studies involving VGG6. The proposed method is also evaluated on reinforcement learning (RL) using Cartpole and Atari-pong. Supplementary section 2 includes architectural details and hyperparameters. The code is submitted as part of the supplementary material.

Results on CIFAR and ImageNet. The experimental results using the proposed scheme are shown in Table 1. We achieve top-1 accuracy of 93.05% and 70.15% on CIFAR-10 and CIFAR-100, respectively using VGG16, with just 1 timestep (T1); results with ResNet20 are also shown in this Table. With reduction of latency from 5 to 1, there is a slight accuracy degradation, however that is due to the inherent accuracy versus latency trade-off in SNNs. Next, to investigate the scalability of the proposed algorithm, we experiment with ImageNet where we obtain 69.00% top-1 accuracy with T1. Notably, the proposed technique allows us to reduce the SNN latency to lowest possible limit.

Performance Comparison. Next, we compare our performance with different state-of-the-art SNNs in Table 2. T1 SNN performs better than or comparably to all these methods, while achieving significantly lower inference latency. In particular, previously it was challenging to obtain satisfactory performance with low latency on ImageNet, with lowest reported timesteps of 4 [8], 5 [31] and 6 [49]. In contrast, we report 69.00% top-1 accuracy on ImageNet using T1. Overall, T1 SNN demonstrates 4-2500X improvement in inference latency compared to other works while maintaining iso or better classification performance. Table 2 also demonstrates the gradual progression of SNN training from ANN-SNN conversion to T1 SNN. Initial ANN-SNN conversion methods required latency on the order of thousands [36,17]. Surrogate-gradient based BP [44,22] reduced it to few tens to hundred. Next, hybrid training [32] combined these two methods to bring the latency down to few hundreds on ImageNet. Subsequently, direct input encoding enabled convergence on ImageNet with latency of ~ 5 [31,49]. The proposed method leverages all these previously proposed techniques and improves upon them by incorporating the temporal pruning approach. We also achieve better performance compared to different SNN encoding schemes such as TTFS [29], phase [18], burst [28]; detailed comparison with these methods is provided in supplementary section 3.

Table 2. Comparison of T1 with other SNN results. SGB, hybrid and TTFS denote surrogate-gradient based backprop, pretrained ANN followed by SNN fine-tuning, and time-to-first-spike, respectively and (qC, dL) denotes q conv layers and d linear layers.

Method	Dataset	Architecture	Accuracy(%)	Timesteps (T)
ANN-SNN conversion [17]	CIFAR10	2C, 2L	82.95	6000
ANN-SNN conversion [2]	CIFAR10	3C, 2L	77.43	400
ANN-SNN conversion [36]	CIFAR10	VGG16	91.55	2500
SGB [22]	CIFAR10	VGG9	90.45	100
ANN-SNN conversion [34]	CIFAR10	4C, 2L	90.85	400
Hybrid [32]	CIFAR10	VGG9	90.5	100
TTFS [29]	CIFAR10	VGG16	91.4	680
Burst-coding [28]	CIFAR10	VGG16	91.4	1125
Phase-coding [18]	CIFAR10	VGG16	91.2	1500
SGB [43]	CIFAR10	2C, 2L	50.7	30
SGB [44]	CIFAR10	5C, 2L	90.53	12
Tandem Learning [42]	CIFAR10	5C, 2L	90.98	8
SGB [48]	CIFAR10	5C, 2L	91.41	5
Direct Encoding [31]	CIFAR10	VGG16	92.70	5
STBP-tdBN [49]	CIFAR10	ResNet-19	93.16	6
Temporal pruning (ours)	**CIFAR10**	**VGG16**	**93.05**	**1**
ANN-SNN conversion [24]	CIFAR100	VGG15	63.2	62
Hybrid [32]	CIFAR100	VGG11	67.9	125
TTFS [29]	CIFAR100	VGG16	68.8	680
Burst-coding [28]	CIFAR100	VGG16	68.77	3100
Phase-coding [18]	CIFAR100	VGG16	68.6	8950
Direct Encoding [31]	CIFAR100	VGG16	69.67	5
Temporal pruning (ours)	**CIFAR100**	**VGG16**	**70.15**	**1**
ANN-SNN conversion [36]	ImageNet	VGG16	69.96	2500
ANN-SNN conversion [34]	ImageNet	VGG16	49.61	400
Hybrid [32]	ImageNet	VGG16	65.19	250
Tandem Learning [42]	ImageNet	AlexNet	50.22	10
ANN-SNN conversion [24]	ImageNet	VGG15	66.56	64
Direct Encoding [31]	ImageNet	VGG16	69.00	5
SGB [8]	ImageNet	ResNet-34	67.04	4
STBP-tdBN [49]	ImageNet	ResNet-34	67.05	6
Temporal pruning (ours)	**ImageNet**	**VGG16**	**69.00**	**1**

Inference Efficiency. Next, we compare the energy efficiency of T1 SNNs with ANNs and multi-timestep SNNs. In SNNs, the floating-point (FP) additions replace the FP MAC operations. This results in higher compute efficiency as the cost of a MAC ($4.6pJ$) is $5.1\times$ to an addition ($0.9pJ$) [15] in 45nm CMOS technology (as shown in Fig. 2(e)). Supplementary section 4 contains the equations of computational cost in the form of operations per layer in an ANN, $\#\text{ANN}_{\text{ops}}$. For an SNN, the number of operations is given as $\#\text{SNN}_{\text{ops, q}} = \text{spike rate}_{\text{q}} \times \#\text{ANN}_{\text{ops, q}}$; where $\text{spike rate}_{\text{q}}$ denotes the average number of spikes per neuron per inference over all timesteps in layer q. The layerwise spike rates across T5 to T1 are shown in Fig. 2(a-c). Note, the spike rates decrease significantly with latency reduction from 5 to 1, leading to considerable reduction in operation count. The overall average spike rates using T1 for CIFAR10, CIFAR100 and ImageNet on VGG16 are 0.13, 0.15 and 0.18, respectively; all significantly below 5.1 (relative cost of MAC to addition).

The first layer with direct input encoded SNNs receive analog inputs, hence the operations are same as an ANN at this layer. Considering it, we calculate the compute energy benefits of T1 SNN over ANN, α as,

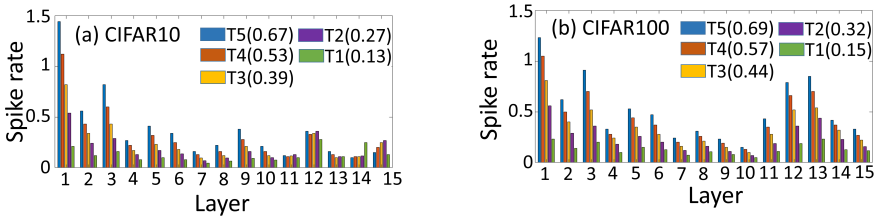
$$\alpha = \frac{\mathrm{E_{ANN}}}{\mathrm{E_{SNN}}} = \frac{\sum_{q=1}^{L} \#\mathrm{ANN_{ops,q}} * 4.6}{\#\ \mathrm{SNN_{ops,1}} * 4.6 + \sum_{q=2}^{L} \#\ \mathrm{SNN_{ops,q}} * 0.9}. \tag{4}$$

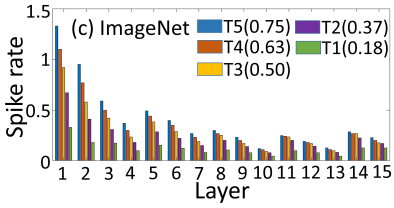
The values of α for different datasets and architectures are given in Fig. 2(d). For VGG16, we obtain α of 33.0, 29.24 and 24.61 on CIFAR-10, CIFAR-100 and ImageNet, respectively. Besides compute energy, another significant overhead in SNNs occurs due to memory access costs which can be significantly higher [12] compared to FP adds as shown in Fig. 2(e). However, most previous works [29,31,49,32] did not consider this cost while comparing the energy benefits of SNN to ANN. To obtain a fairer comparison, we analyze the costs taking the memory access issue into consideration. For multi-timestep SNNs, in addition to the weights, $\mathrm{V_{mem}}$ needs to be stored and fetched at each timestep. However, the memory requirements for T1 SNNs are same as ANNs. The actual improvements in energy depends on the hardware architecture and system configurations. Hence, we compare the reduction in terms of number of memory access. For a VGG16, the proposed T1 reduces the number of memory accesses by 5.03× compared to 5 timestep SNN of [31]. More generally, our scheme with T1 reduces the number of memory accesses by approximately T× compared to an SNN trained with T timesteps.

Table 3. Comparison of T1 SNN with BNN

Method	Dataset	Accuracy(%)
Binary activation [35]	CIFAR10	89.6
STE-BNN [1]	CIFAR10	85.2
BN-less BNN [3]	CIFAR10	92.08
Trained binarization[45]	CIFAR10	92.3
BBG-Net [37]	CIFAR10	92.46
CI-BCNN [41]	CIFAR10	92.47
Binary activation [6]	CIFAR10	91.88
This work (T1)	CIFAR10	93.05
Binary activation [6]	CIFAR100	70.43
BN-less BNN [3]	CIFAR100	68.34
BBG-Net [37]	CIFAR100	69.38
This work (T1)	CIFAR100	70.15

Comparison with binary activated ANNs. A key distinction between ANNs and SNNs is the notion of time. However, T1 SNN and binary neural networks (BNNs) both infer in single shot using binary activations. But, there are some differences which have been discussed in section 2. Here, we compare the performance of T1 SNN and BNNs. We observe that T1 SNN performs on par or better than other BNN approaches on CIFAR10 and CIFAR100 datasets, as depicted in Table 3. Furthermore, training large scale datasets such as ImageNet using BNN methods [35,1,4,3,45,37] has been challenging, while the proposed method scales well to ImageNet. Notably, Xnor-net [30] and Dorefa-net [50] achieve 44.2% and 43.6% top-1 accuracy on ImageNet, respectively, with





(d) ANN vs T1 SNN compute energy

Network	Dataset	α
VGG16	CIFAR10	33.0
VGG16	CIFAR100	29.24
VGG16	ImageNet	24.61
ResNet20	CIFAR10	16.32
ResNet20	CIFAR100	15.35

(e) Energy Table

Operation (32 bits)	Energy (pJ)	Relative Cost
FP add	0.9	1
FP mult	3.7	4.1
FP MAC	4.6	5.1
SRAM Cache	5	5.6
DRAM	640	711.1

Fig. 2. Layerwise spike rates for a VGG16 (average spike rate in parenthesis) on (a) CIFAR10, (b) CIFAR100, (c) ImageNet, (d) relative cost of compute between ANN and T1, (e) operation-wise energy consumption in 45nm CMOS [15].

1-bit activations, while T1 achieves 69%. However, SNNs like some of the BNNs in Table 3 [35,6] use real-valued weights, while other BNNs use binary weights [4,30,50]. Although SNNs use full precision weights, they provide computational efficiency like BNNs by replacing MACs with adds, while maintaining accuracy. Note, we performed energy comparison of T1 with ANNs following [32,31]. However, BNNs infer using XNOR and popcount ops instead of adds (used in SNNs), which might further improve efficiency. Therefore, the proposed SNNs can provide suitable trade-off between accuracy and efficiency between ANN and BNNs.

Efficacy of temporal pruning with shallow networks. Temporal pruning is required due to spike vanishing at later layers if direct transition from T5 to T1 is attempted. However, for shallow networks, training with T1 following direct conversion from ANN might be possible. To investigate this, we experiment with a VGG6 on CIFAR10 and the results are shown in Table 4, where VGG6g denotestrained with temporal pruning and VGG6d denotes directly converted from ANN and trained using that particular timestep. The proposed scheme provides slightly higher accuracy compared to direct training in case of shallower networks. This is consistent with [10,13], where the authors achieve better performing networks using sequential model compression.

Proposed method with and without batch-norm. Recent works have achieved low latency by adopting batch-normalization (BN) suitably in SNNs [49,19]. To disentangle the effect of BN from the proposed scheme and ensure that achieving convergence with 1 timestep is orthogonal to using BN, we perform ablation studies as shown in Table 5. For both CIFAR10 and CIFAR100, we

Table 4. Accuracy(%) on CIFAR10

T	VGG6g	VGG6d
5	90.61	90.15
4	90.52	90.08
3	90.40	89.91
2	90.05	89.35
1	89.10	88.64

Table 5. Accuracy(%) with VGG16, D_w and D_wo denote dataset D with and without batch-norm respectively

T	CIFAR10_w	CIFAR10_wo	CIFAR100_w	CIFAR100_wo
5	93.90	92.15	71.58	69.86
4	93.87	91.95	71.51	69.84
3	93.85	91.90	71.46	69.76
2	93.72	91.88	71.43	69.30
1	93.05	91.03	70.15	67.76

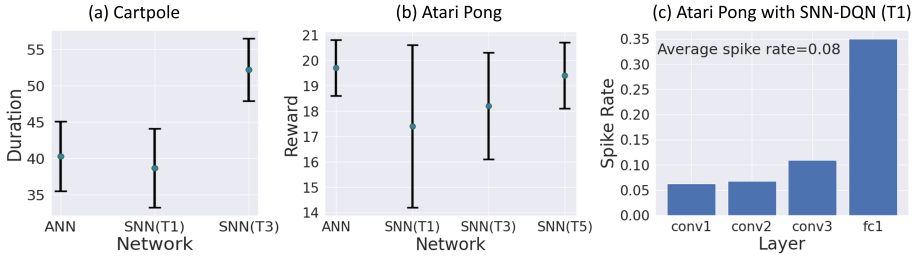


Fig. 3. Average reward (errorbars depict mean±std), using DQN with ANN and SNN on (a) Cartpole and (b) Atari Pong, (c) layerwise spike rate with T1 on Atari Pong.

are able to perform training with T5 to T1 irrespective of using BN during ANN training. Using BN enhances accuracy, but sequential temporal pruning can be performed independently from it. Also, [49] reports that using threshold-dependent BN allows reducing latency up to a minimum of 6 timesteps, but we can go up to 1. Note, BN is used only during ANN training and the BN parameters are fused with the weights during ANN-SNN conversion as proposed in [34]; BN is not used in the SNN domain.

Skipping intermediate timestep reduction steps. Since the proposed scheme increases training overhead due to sequential temporal pruning, it is worth investigating if this cost can be reduced by skipping in between timestep reduction steps. To that effect, we experiment with 2 cases- (i) training T5 followed by T3_5, followed by T1_3, (ii) including all intermediate timesteps with the sequence- T5, T4_5, T3_4, T2_3 and T1_2. Interestingly, both these cases perform comparably; for CIFAR10, we obtain 93.01% and 93.05% accuracy, respectively with 1 timestep for cases (i) and (ii). These values for CIFAR100 are 69.92% and 70.15%, respectively, and for ImageNet, the values are 68.98% and 69%, respectively. This indicates that if the end goal is obtaining T1 SNN, training overhead can be reduced by skipping intermediate steps.

T1 with spatial pruning. With T1, we obtain maximum compression in the temporal domain of SNNs. Additionally, we hypothesize that T1 might be amenable to spatial pruning as there is redundancy in the weights of DNNs [13,12]. To investigate this, we perform experiments on T1 with magnitude based

weight pruning [12]. Our result indicate that we can remove up to 90% of the total weights of a VGG16 without large drop in performance. Using VGG16 (T1), we obtain 91.15% accuracy on CIFAR10 and 68.20% accuracy on CIFAR100, while retaining just 10% of the original spatial connections. This provides evidence that spatial pruning techniques can be combined with T1 SNNs.

Reinforcement learning (RL) with proposed SNNs. Due to their inherent recurrence, SNNs with multiple timesteps might be more useful in sequential decision-making (such as RL tasks) than static image classification. However, application of SNNs in RL may be limited if the latency is too high, since the agent has to make decisions in real-time. The authors in [39] obtain high performing RL agents on Atari games with SNNs, but with 500 timesteps. In this section, we investigate if our technique can enable training SNNs for RL tasks with low latency. The training pipeline is similar to that used for image classification tasks (hybrid SNN training with temporal pruning). Experiments are performed using deep Q-networks (DQN) [26] with SNNs (SNN-DQN) on cartpole and Atari pong environments. As shown in Fig. 3(a) and (b), for both cases, we can train SNNs up to 1 timestep. The rewards are obtained by averaging over 20 trials and plotted with error-bars showing mean±std. In (a), the reward is the duration of the cartpole remaining balanced. DQN with ANN, SNN(T1) and SNN(T3) achieve 40.3±4.8, 38.7±5.4, 52.2±4.3 reward, respectively. While T1 SNN performs slightly worse compared to ANN-based DQN, T3 outperforms the ANN. Similar performance improvement over ANN using SNN for some RL tasks has been reported in [39]. Next, we experiment with a more complex task, Atari pong game. In this case, T1 achieves reward of 17.4±3.2 compared to ANN based DQN's 19.7±1.1. However, with T5, we obtain comparable reward (19.4±1.3) to ANN. These results demonstrate that even T1 can handle dynamic inputs, albeit with lower accuracy than T3 or T5 (Fig. 3). Number of timestep is defined as the number of forward passes through the model. As such, T1 does not prevent usage with temporal inputs, rather each frame is processed just once, like an RNN. However, for sequential tasks, SNNs with multiple timesteps indeed provide enhanced performance using the inherent memory of membrane potential in neurons. Notably, we can obtain SNNs for RL task (pong) with significantly lower latency compared to prior art. On pong, [39] reports reward of 19.8±1.3 using 500 timesteps, whereas, we obtain 19.4±1.3 reward using just 5 timesteps. Our training method enables SNNs for RL tasks with comparable performance to ANNs with $\sim$ 5 timesteps, and up to 1 timestep with slightly lower performance. Moreover, such latency reduction translates to considerable energy savings which is critical for agents operating in the real world. The layerwise spike rates for SNN-DQN (T1) are shown in Fig. 3(c), the average spike rate is 0.08, which results in 7.55X higher computational energy efficiency compared to ANN-DQN. Furthermore, if we compare iso-performing networks to ANN-DQN, SNN-DQN (T5) infers with average spike rate of 0.42, thus providing 5.22X higher computational energy efficiency compared to ANN-DQN. Details of training and additional results are given in supplementary section 6.

Is direct training feasible? Though the overhead due to iterative training does not affect our primary goal (inference efficiency), we explore the feasibility of directly obtaining T1 SNNs here. We reproduce the results of [35,1] where the networks are trained as BNN; but as shown in Table 3, the iterative process provides better results. Furthermore, it is challenging to scale these networks with purely binary activation to ImageNet. More importantly, the continuum of SNNs cannot be obtained using this process, which is required to have optimal SNN solutions for both static and dynamic tasks. Additionally, we investigate if direct conversion from ANN to T1 is feasible and find that this case leads to convergence failure (details provided in supplementary section 5).

Limitations. A limitation of the proposed scheme is the training overhead due to sequential training. However, this occurs only in training (can be performed offline) and our end goal is inference which is not impacted by this iterative process. Moreover, this cost can be reduced by skipping latency reduction steps. Furthermore, if the end goal is T1, training can be performed with equivalent overhead to T5 by early stopping training of parent SNNs. In this case, we do not train the SNNs with higher timesteps till full convergence. Rather, we keep the number of overall training epochs fixed and divide the epochs equally among the temporal pruning stages. Let, the number of training epochs used for training only T5 is v. Then, for the case of training T5 followed by T3_5, followed by T1_3, we use (v/3) number of training epochs at each stage. Interestingly, this causes negligible performance drop ($< 0.3\%$) for CIFAR10 with VGG16.

6 Conclusion

Bio-plausible SNNs hold promise as energy efficient alternatives to ANNs. However, mitigating high inference latency is critical for their edge deployment. To that end, we propose a temporal pruning approach which results in a continuum of SNNs from T timesteps up to unity (T1). The proposed low-latency SNNs use direct input coding with threshold and leak as trainable parameters along with weights. This leads to a spectrum of optimum SNN solutions for both static and sequential tasks. For static vision applications, T1 SNNs provide the best efficiency and latency. In particular, the T1 SNNs enhance computational efficiency by 25X on ImageNet compared to ANNs using VGG16 and are able to reduce the memory access overhead compared to SNNs with higher T. On the other hand, the presented approach also provides SNN based solutions for sequential tasks using few (1-5) timesteps. Notably, these SNN based deep Q-networks (DQNs) perform comparably to ANNs while providing significantly higher efficiency. Thus, the proposed technique enables training highly efficient SNNs on static tasks (up to unit timestep) as well as on sequential tasks (with few timesteps) while maintaining satisfactory performance.

Acknowledgement. The work was supported in part by, Center for Brain-inspired Computing (C- BRIC), a DARPA sponsored JUMP center, Semiconductor Research Corporation, National Science Foundation, Intel Corporation, the DoD Vannevar Bush Fellowship and U.S. Army Research Laboratory.

References

1. Bengio, Y., Léonard, N., Courville, A.: Estimating or propagating gradients through stochastic neurons for conditional computation. arXiv preprint arXiv:1308.3432 (2013)
2. Cao, Y., Chen, Y., Khosla, D.: Spiking deep convolutional neural networks for energy-efficient object recognition. International Journal of Computer Vision **113**(1), 54–66 (2015)
3. Chen, T., Zhang, Z., Ouyang, X., Liu, Z., Shen, Z., Wang, Z.: " bnn-bn=?": Training binary neural networks without batch normalization. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 4619–4629 (2021)
4. Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., Bengio, Y.: Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830 (2016)
5. Deng, L., Liu, Y.: Deep learning in natural language processing. Springer (2018)
6. Deng, S., Gu, S.: Optimal conversion of conventional artificial neural networks to spiking neural networks. In: International Conference on Learning Representations (2020)
7. Diehl, P.U., Neil, D., Binas, J., Cook, M., Liu, S.C., Pfeiffer, M.: Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In: 2015 International Joint Conference on Neural Networks (IJCNN). pp. 1–8. ieee (2015)
8. Fang, W., Yu, Z., Chen, Y., Huang, T., Masquelier, T., Tian, Y.: Deep residual learning in spiking neural networks. arXiv preprint arXiv:2102.04159 (2021)
9. Frenkel, C.: Sparsity provides a competitive advantage. Nature Machine Intelligence **3**(9), 742–743 (2021)
10. Furlanello, T., Lipton, Z., Tschannen, M., Itti, L., Anandkumar, A.: Born again neural networks. In: International Conference on Machine Learning. pp. 1607–1616. PMLR (2018)
11. Han, B., Srinivasan, G., Roy, K.: Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 13558–13567 (2020)
12. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149 (2015)
13. Han, S., Pool, J., Tran, J., Dally, W.J.: Learning both weights and connections for efficient neural networks. arXiv preprint arXiv:1506.02626 (2015)
14. Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., et al.: Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal processing magazine **29**(6), 82–97 (2012)
15. Horowitz, M.: 1.1 computing's energy problem (and what we can do about it). In: 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC). pp. 10–14. IEEE (2014)
16. Huh, D., Sejnowski, T.J.: Gradient descent for spiking neural networks. In: Advances in Neural Information Processing Systems. pp. 1433–1443 (2018)
17. Hunsberger, E., Eliasmith, C.: Spiking deep networks with lif neurons. arXiv preprint arXiv:1510.08829 (2015)

18. Kim, J., Kim, H., Huh, S., Lee, J., Choi, K.: Deep neural networks with weighted spikes. Neurocomputing **311**, 373–386 (2018)
19. Kim, Y., Panda, P.: Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. arXiv preprint arXiv:2010.01729 (2020)
20. Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., Mahoney, M.W.: Characterizing possible failure modes in physics-informed neural networks. Advances in Neural Information Processing Systems **34** (2021)
21. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
22. Lee, C., Sarwar, S.S., Panda, P., Srinivasan, G., Roy, K.: Enabling spike-based backpropagation for training deep neural network architectures. Frontiers in Neuroscience **14** (2020)
23. Li, D., Chen, X., Becchi, M., Zong, Z.: Evaluating the energy efficiency of deep convolutional neural networks on cpus and gpus. In: 2016 IEEE international conferences on big data and cloud computing (BDCloud), social computing and networking (SocialCom), sustainable computing and communications (SustainCom)(BDCloud-SocialCom-SustainCom). pp. 477–484. IEEE (2016)
24. Lu, S., Sengupta, A.: Exploring the connection between binary and spiking neural networks. arXiv preprint arXiv:2002.10064 (2020)
25. Maass, W.: Networks of spiking neurons: the third generation of neural network models. Neural networks **10**(9), 1659–1671 (1997)
26. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. nature **518**(7540), 529–533 (2015)
27. Neftci, E.O., Mostafa, H., Zenke, F.: Surrogate gradient learning in spiking neural networks. IEEE Signal Processing Magazine **36**, 61–63 (2019)
28. Park, S., Kim, S., Choe, H., Yoon, S.: Fast and efficient information transmission with burst spikes in deep spiking neural networks. In: 2019 56th ACM/IEEE Design Automation Conference (DAC). pp. 1–6. IEEE (2019)
29. Park, S., Kim, S., Na, B., Yoon, S.: T2fsnn: Deep spiking neural networks with time-to-first-spike coding. arXiv preprint arXiv:2003.11741 (2020)
30. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: Xnor-net: Imagenet classification using binary convolutional neural networks. In: European conference on computer vision. pp. 525–542. Springer (2016)
31. Rathi, N., Roy, K.: Diet-snn: Direct input encoding with leakage and threshold optimization in deep spiking neural networks. arXiv preprint arXiv:2008.03658 (2020)
32. Rathi, N., Srinivasan, G., Panda, P., Roy, K.: Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In: International Conference on Learning Representations (2020), `https://openreview.net/forum?id=B1xSperKvH`
33. Roy, K., Jaiswal, A., Panda, P.: Towards spike-based machine intelligence with neuromorphic computing. Nature **575**(7784), 607–617 (2019)
34. Rueckauer, B., Lungu, I.A., Hu, Y., Pfeiffer, M., Liu, S.C.: Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. Frontiers in neuroscience **11**, 682 (2017)
35. Sakr, C., Choi, J., Wang, Z., Gopalakrishnan, K., Shanbhag, N.: True gradient-based training of deep binary activated neural networks via continuous binarization. In: 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 2346–2350. IEEE (2018)

36. Sengupta, A., Ye, Y., Wang, R., Liu, C., Roy, K.: Going deeper in spiking neural networks: Vgg and residual architectures. Frontiers in neuroscience **13**, 95 (2019)
37. Shen, M., Liu, X., Gong, R., Han, K.: Balanced binary neural networks with gated residual. In: ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 4197–4201. IEEE (2020)
38. Shrestha, S.B., Orchard, G.: Slayer: Spike layer error reassignment in time. In: Advances in Neural Information Processing Systems. pp. 1412–1421 (2018)
39. Tan, W., Patel, D., Kozma, R.: Strategy and benchmark for converting deep q-networks to event-driven spiking neural networks. arXiv preprint arXiv:2009.14456 (2020)
40. Wang, P., He, X., Li, G., Zhao, T., Cheng, J.: Sparsity-inducing binarized neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 34, pp. 12192–12199 (2020)
41. Wang, Z., Lu, J., Tao, C., Zhou, J., Tian, Q.: Learning channel-wise interactions for binary convolutional neural networks. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 568–577 (2019)
42. Wu, J., Chua, Y., Zhang, M., Li, G., Li, H., Tan, K.C.: A tandem learning rule for efficient and rapid inference on deep spiking neural networks. arXiv pp. arXiv–1907 (2019)
43. Wu, Y., Deng, L., Li, G., Zhu, J., Shi, L.: Spatio-temporal backpropagation for training high-performance spiking neural networks. Frontiers in neuroscience **12**, 331 (2018)
44. Wu, Y., Deng, L., Li, G., Zhu, J., Xie, Y., Shi, L.: Direct training for spiking neural networks: Faster, larger, better. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 1311–1318 (2019)
45. Xu, Z., Cheung, R.C.: Accurate and compact convolutional neural networks with trained binarization. In: 30th British Machine Vision Conference (BMVC 2019) (2019)
46. Yang, C., Xie, L., Qiao, S., Yuille, A.L.: Training deep neural networks in generations: A more tolerant teacher educates better students. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 33, pp. 5628–5635 (2019)
47. Zenke, F., Ganguli, S.: Superspike: Supervised learning in multilayer spiking neural networks. Neural computation **30**(6), 1514–1541 (2018)
48. Zhang, W., Li, P.: Temporal spike sequence learning via backpropagation for deep spiking neural networks. Advances in Neural Information Processing Systems **33** (2020)
49. Zheng, H., Wu, Y., Deng, L., Hu, Y., Li, G.: Going deeper with directly-trained larger spiking neural networks. In: Proceedings of the AAAI Conference on Artificial Intelligence. vol. 35, pp. 11062–11070 (2021)
50. Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., Zou, Y.: Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv preprint arXiv:1606.06160 (2016)