ONE TIMESTEP IS ALL YOU NEED: TRAINING SPIKING NEURAL NETWORKS WITH ULTRA LOW LATENCY

Anonymous authors
Paper under double-blind review

ABSTRACT

Spiking Neural Networks (SNNs) can be energy efficient alternatives to commonly used deep neural networks (DNNs). Through event-driven information processing, SNNs can considerably reduce the compute requirements of DNNs. However, high inference latency is a significant hindrance to their deployment. Computation over multiple timesteps increases latency and incurs memory access overhead of fetching membrane potentials, both of which lessen the energy benefits of SNNs. Hence, latency reduction is pivotal to obtain SNNs with high energy efficiency. However, reducing latency can have an adverse effect on accuracy. To obtain solutions which optimize the accuracy-energy-latency trade-off, we propose an iterative training method which starts with an SNN of T (T>1) timesteps, and reduces T every iteration of training, with neuron threshold and leak as trainable parameters. This results in a continuum of SNNs, starting from an SNN trained with T timesteps, all the way up to unit latency. We use direct input encoding (analog inputs from pixels) with the first convolutional layer of the network of leaky integrate and fire (LIF) neurons acting as spike generator. We choose T=5 as our starting point, since it is the minimum reported latency to achieve satisfactory performance on ImageNet. Training SNNs directly with 1 timestep results in convergence failure due to layerwise spike vanishing and difficulty in finding optimum thresholds. The proposed iterative training approach overcomes this through enabling the learning of suitable layer-wise thresholds with backpropagation by maintaining sufficient spiking activity, starting from T timesteps up to 1. Using the proposed training algorithm, we achieve top-1 accuracy of 93.05%, 70.15% and 67.71% on CIFAR-10, CIFAR-100 and ImageNet, respectively with VGG16, in just 1 timestep. Compared to a 5 timestep SNN, the 1 timestep SNN achieves $\sim$ 5X enhancement in efficiency, with an accuracy drop of $\sim 1\%$. In addition, 1 timestep SNNs perform inference with 5X reduced latency compared to state-of-the-art SNNs, and provide 25-33X higher energy efficiency compared to DNNs, while being comparable to them in performance. The proposed method also enables training reinforcement learning agents on Cartpole and Atari pong environments which infer using 1 timestep.

1 INTRODUCTION

Deep neural networks (DNNs) have revolutionized the fields of object detection, classification and natural language processing (Krizhevsky et al., 2012; Hinton et al., 2012; Deng & Liu, 2018). However, such performance boost has been achieved at the cost of extremely energy intensive DNN architectures (Li et al., 2016). Therefore, edge deployment of such DNNs remains a challenge. One approach to counter this is to perform model compression using pruning (Han et al., 2015b; Wen et al., 2016), and data quantization (Courbariaux et al., 2016). An alternative route is using Spiking Neural Networks (SNNs) (Maass, 1997; Roy et al., 2019), which are bio-inspired learning frameworks and perform computations using binary spikes instead of analog activations used in standard networks. In this paper, standard networks are referred to as Analog Neural Networks (ANNs) in contrast to SNNs that operate with leaky integrate and fire (LIF) neurons having spike activation. As such, the sparse event-driven nature of SNNs makes them an attractive alternative to ANNs (Frenkel, 2021), specially for resource-constrained machine intelligence.

The performance of SNNs can be characterized by their accuracy, latency and energy efficiency. While reducing latency is desirable as it leads to efficiency improvements, significant latency reduction usually impacts accuracy adversely. Most of the commonly used SNNs use Poisson rate-coding (Diehl et al., 2015; Sengupta et al., 2019), where the firing rates of the spiking neurons approximate the analog activation of ANNs. Though rate-coding provides comparable results to ANNs, it suffers from high inference latency (Sengupta et al., 2019; Han et al., 2020). In rate-

coding, the discretization error caused by representing analog values with spike-trains is inversely proportional to the number of timesteps. Hence, a large number of timesteps is usually required to obtain high performance (Sengupta et al., 2019). However, this requirement of 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 a large number of timesteps results in higher number of operations, thereby reducing energy-efficiency. Moreover, in contrast to standard 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 (Han et al., 2015a). As a result, reducing inference latency and energy, while achieving the required accuracy is critical for widespread deployment of SNNs.

To leverage the full potential of SNNs by optimizing the associated accuracy-energy-latency trade-offs, we propose an iterative training technique which starts with an SNN of T (T>1) timesteps, and gradually reduces T at every iteration of training using neuron threshold and leak as trainable parameters alongside the weights. At each stage of latency reduction, the network trained at previous stage with higher timestep is used as initialization for subsequent training with lower timestep. Using such an iterative process, we obtain a continuum of SNNs, starting from an SNN trained with T timesteps, eventually leading up to unit latency. Iterative training is required since we observe that training SNNs directly with very few timesteps results in convergence failure due to significant decay of spiking activity in the deeper layers. In order to perform inference with very low latency, sufficient spikes must be propagated to the final layer in only a few forward passes. To achieve that, the layerwise neuron thresholds and membrane leak factors must be adjusted properly such that the neurons can propagate spikes in just few timesteps. Hence, our approach is to learn the optimum neuron thresholds and membrane leak factors using backpropagation (BP) for any given timestep, such that enough spikes are available at the later layers for efficient learning. For our experiments, a hybrid training method (Rathi et al., 2020) is used, where first an ANN is trained, followed by ANN-SNN conversion, and SNN training with surrogate gradient based backpropagation (Neftci et al., 2019). We use direct input encoding (analog inputs from pixels) (Rueckauer et al., 2017) with the first convolutional layer of the network of leaky integrate and fire (LIF) neurons acting as spike generator. This method is adopted since direct coding with trainable threshold and leak has enabled reducing the latency of SNNs significantly (Rathi & Roy, 2020). The starting point of the proposed iterative latency reduction scheme is chosen to be an SNN trained with T=5 timesteps, since as per literature, it is the minimum required latency to perform satisfactorily on ImageNet with SNNs (Rathi & Roy, 2020; Zheng et al., 2021). The proposed gradual latency reduction scheme enables sufficient spike activity to be maintained till the final layer even for extremely low latencies (up to 1). Consequently, we obtain a continuum of SNNs with varying latency ranging from T timesteps (T>1) all the way up to 1 timestep. We validate the performance of the proposed method on image classification tasks and achieve top-1 accuracy of 93.05%, 70.15% and 67.71% on CIFAR-10, CIFAR-100 and ImageNet, respectively with VGG16, in just 1 timestep. Furthermore, we investigate the applicability of our technique to SNN-based reinforcement learning (RL) agents with low latency. This is important since SNNs can potentially provide RL solutions with high energy efficiency (Tan et al., 2020), which is pivotal for their real-world deployment. We observe that the proposed method enables training RL agents on Cartpole and Atari pong environments which can infer with unit latency while maintaining satisfactory performance.

To summarize, the main contributions of this work are as follows,

- We present an iterative training technique that enables training a continuum of SNNs with varying latency, starting from T timesteps all the way up to 1; the 1 timestep SNN providing the highest efficiency and lowest latency but with a slight accuracy drop compared to the T=5 timestep SNN. To the best of our knowledge, this is the first SNN work to achieve competitive classification performance (top-1 accuracy of 67.71%) on ImageNet using unit latency.
- Since inference can be performed in a single forward pass, our approach does not require the memory access cost of accumulated membrane potentials, unlike previously proposed SNNs. However, the trade-off is a slight accuracy degradation compared to SNNs with higher timesteps.
- One timestep SNNs are able to infer with 5X lower latency compared to state-of-the-art SNNs (requiring latency of 5 timesteps), while achieving comparable or even better accuracy.
- One timestep inference also provides efficient computation due to low spike rate, resulting in SNNs which are up to 33X energy efficient compared to ANNs with equivalent architecture.

- The proposed method enables deep-Q reinforcement learning with 1 timestep SNNs, suitable for edge deployment, as verified on simple tasks like Cartpole and Atari Pong.

2 Related Works

ANN-SNN Conversion. A widely used approach for training deep SNNs involves training an ANN and converting to SNN for finetuning (Cao et al., 2015; Diehl et al., 2015; Sengupta et al., 2019). To minimize conversion loss, the ANNs are usually trained without bias, batch-norm or maxpooling layers, though some works bypass these constraints using custom layers (Rueckauer et al., 2017). Proper layerwise threshold adjustment is critical to convert ANNs to SNNs successfully. One approach of threshold balancing is to choose the layerwise thresholds as the maximum pre-activation of the neurons (Sengupta et al., 2019). While such assignment provides high accuracy, the associated drawback is high inference latency (about 1000 timesteps). Alternatively, Rueckauer et al. (2017) suggest choosing a certain percentile of the pre-activation distribution as the threshold to reduce inference latency and improve robustness. However, these conversion based methods (Rueckauer et al., 2017; Han et al., 2020) still require few hundred timesteps to perform satisfactorily.

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 (Neftci et al., 2019) to implement BP in SNNs (Huh & Sejnowski, 2018; Lee et al., 2020). Zenke & Ganguli (2018) propose surrogate gradient based BP on membrane potential. Shrestha & Orchard (2018) perform BP based on the gradients calculated using the difference between the membrane potential and the threshold. In Wu et al. (2018), BP is performed on SNNs with a surrogate gradient defined on the continuous-valued membrane potential. Overall, surrogate-gradient based BPTT training has resulted in SNNs with high accuracy, but the training is compute intensive compared to conversion techniques and the latency is still significant ($\sim$100-125 timesteps). Rathi et al. (2020) propose a hybrid approach where a pre-trained ANN is used as initialization for subsequent surrogate gradient based 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 (Kim et al., 2018) or burst (Park et al., 2019) coding attempt to capture temporal information into the spike trains; a related method is time-to-first-spike (TTFS) coding (Park et al., 2020), where each neuron is allowed to spike just once. While such input coding techniques enhance computational efficiency by reducing the spike count, the high latency issue persists and incurs high memory access overhead.

Direct Encoding. The analog pixels are directly applied to the 1^{st} layer in direct encoding (Rueckauer et al., 2017; Rathi & Roy, 2020; Zheng et al., 2021). Using direct coding and utilizing the first layer as spike generator, Rathi & Roy (2020) achieve competitive performance on ImageNet with 5 timesteps. Threshold-dependent batch normalization is employed with direct encoding by Zheng et al. (2021) to obtain high performing SNNs on ImageNet with 6 timesteps. Inspired by such performance, we adopt the direct encoding method. However, even with direct coding, it has been challenging for state-of-the-art SNNs (Rathi & Roy, 2020; Zheng et al., 2021) to infer with unit latency while maintaining high performance on complex datasets, which we aim to address here.

3 Background

Spiking Neuron Model. The LIF neuron model (Hunsberger & Eliasmith, 2015) 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 representing weighted sum of spikes, time constant for membrane potential decay, leakage path resistance, firing threshold and resting potential, respectively. We employ a discretized version of Eqn. 1 given as-

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

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

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. As indicated by Eqn. 2, we implement a soft-reset, so, u is reduced by v upon crossing the threshold. The detailed methodology of training SNN with a certain inference timestep is provided in appendix section A.1. In particular, Algorithm 2 depicts the training scheme for one iteration.

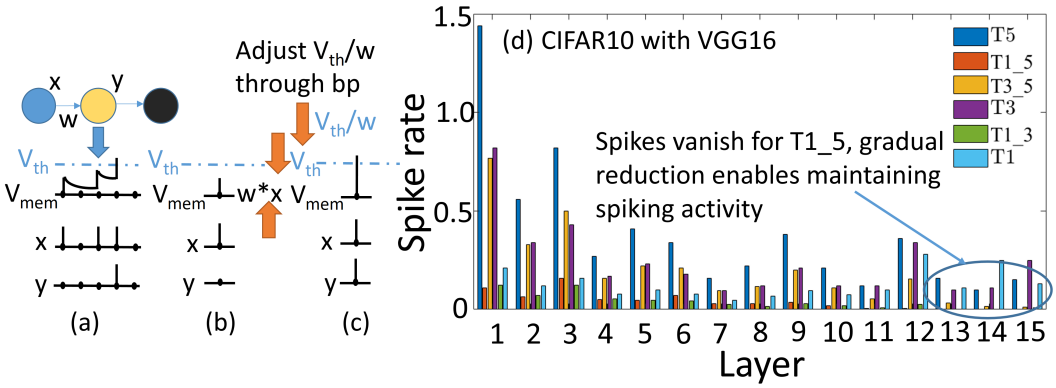


Figure 1: (a) Schematic of an SNN with dynamics shown for yellow neuron with x as input and y as output, (b) No output spike for direct transition from 5 to 1 timestep, (c) Output spike in just 1 timestep through V_{th}/w lowering, (d) Layerwise spike rates, Tx represents a converged SNN with 'x' timestep,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

4 Proposed Latency Reduction Method

The starting point of our training pipeline follows the proposed methodology in Rathi & Roy (2020), difference being we utilize batch-norm (BN) during ANN training and subsequently the BN parameters are fused with the layerwise weights as done in Rueckauer et al. (2017). With this 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 (T$>$ 1) using BP which serves as our baseline; training steps are detailed in Algorithm 2. Our starting point is a T=5 timesteps trained network, since it is lowest latency that previous works have reported for SNN ImageNet training (Rathi & Roy, 2020; Zheng et al., 2021) with high performance. Going below 5 results in convergence failure due to spike vanishing at deeper layers.

Direct Inference with Unit Timestep. As mentioned in Section 1, our goal is to obtain SNNs with unit latency to harness the maximum efficiency 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 perform inference with 1 timestep (Fig. 1(b)), there is no output spike as the V_{mem} is unable to reach V_{th} instantly. Hence, w and V_{th} need to be adjusted such that information can propagate even within 1 step. However, balancing the thresholds properly is critical for SNNs to perform well. If V_{th} is too high, spikes vanish, but lowering V_{th} too much results in too many spikes, leading to unsatisfactory performance (Zheng et al., 2021). 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). For the rest of this paper, with Tx and Tx_y, we denote converged SNN with 'x' timestep and an SNN with 'x' timestep initialized with a 'y' timestep trained SNN, respectively. 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 end layer are 0, thus BP fails to start training. This happens since the initial V_{th} and w are trained for 5 timesteps, so the drastic transition to 1 timestep hinders spikes to propagate till the end.

Gradual Latency Reduction. To mitigate the above issue, we adopt a gradual latency reduction approach. We observe that for T3_5, though layerwise spike activity decays significantly, there is still some spikes that reach the final layer. 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 initialization and retraining method which enables training a continuum of SNNs starting from T timesteps (T$>$ 1) up to 1. 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. While training with a certain latency, the network is initialized with the higher timestep network trained at previous training stage.

Table 1: Top-1 classification accuracy (%), Tx denotes SNN 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	68.62	67.71

Latency Reduction as Form of Compression. SNNs with multiple timesteps are trained with BP through time (BPTT) (Neftci et al., 2019), like RNNs. If we unroll SNNs in time, it becomes obvious that each timestep adds a hidden state to a spiking neuron. Thus, reducing latency compresses the SNNs in the temporal axis and the eventual unit timestep network consists of a single forward pass, with no internal states. From a related perspective, we can perceive gradual latency reduction as sequential temporal distillation, where at each step, the network with higher timesteps acts as the teacher, while the student network with reduced latency learns from it. Similar sequential distillation (training by generations) has been implemented in ANNs (Furlanello et al., 2018; Yang et al., 2019) to obtain better performing students. However, in our case, there is no explicit distillation loss and the sequential training is a requirement for convergence instead of being just a performance enhancement tool. Also, unlike ANN sequential distillation, the student architecture remains same spatially throughout all generations in the proposed method and compression occurs in the temporal domain of SNNs.

Algorithm 1 Pseudo-code of training.

Input: Trained SNN with N timesteps (TN), timesteps reduced per reduction step (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 2
  end for
  // Initialize another iso-architecture SNN with parameters of above trained network
  // Timestep reduction
  $T_r = T_r - b$
end while

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. Appendix A.2 includes architectural details and training hyper-parameters. The code is submitted as part of the supplementary material.

Results on CIFAR and ImageNet. The experimental results using the proposed scheme for CIFAR and ImageNet datasets 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 architecture, with just 1 timestep (T1); results with ResNet20 are also shown in the Table. Next, to investigate the scalability of the proposed algorithm, we experiment with ImageNet where we obtain 67.71% top-1 accuracy with T1. As we sequentially reduce the timestep from 5 to 1, there is a slight accuracy degradation, however that is due to the inherent accuracy versus latency trade-off in SNNs. Notably, the proposed technique allows us to bring the SNN latency down to lowest possible limit (1 timestep) while maintaining comparable accuracy to corresponding ANN as well as the 5 timestep SNN. Additionally, this provides the option to choose the SNN model suitable for application in hand from a range of timesteps; if latency and efficiency is the priority, T1 would be preferred, albeit with slightly lower accuracy compared to T5.

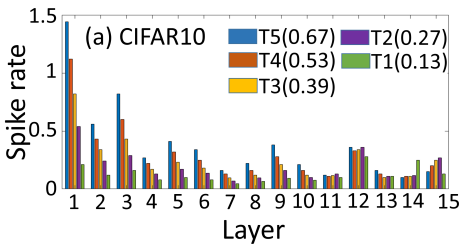
Performance Comparison. Next, we compare our performance with different state-of-the-art SNNs in Table 2. T1 SNN performs better than or comparably to all these methods, while achieving significantly lower inference latency. In particular, note that previously it was challenging to obtain satisfactory performance with low latency on ImageNet, with lowest reported latencies of 5 (Rathi & Roy, 2020) and 6 (Zheng et al., 2021). In contrast, we report 67.71% top-1 accuracy on ImageNet using T1. Furthermore, Wu et al. (2019b) achieve 90.53% accuracy on CIFAR-10 using 12 timesteps with direct input encoding (similar to us), but the proposed method enables us to obtain 93.05% accuracy on CIFAR-10 using T1. For CIFAR100, Rathi & Roy (2020) report 69.67% accuracy with 5 timesteps, whereas T1 achieves 70.15%. Overall, T1 SNN demonstrates 5-2500X improvement in inference latency compared to other works while maintaining iso or better

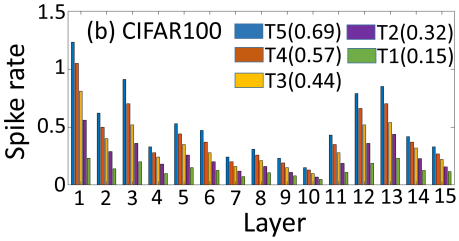
Table 2: Comparison of T1 SNN to other reported results. SGB, hybrid and TTFS denote surrogate-gradient based backprop, pretrained ANN followed by SNN fine-tuning, and time-to-first-spike scheme, respectively and (qC, dL) denotes an architecture with q conv layers and d linear layers.

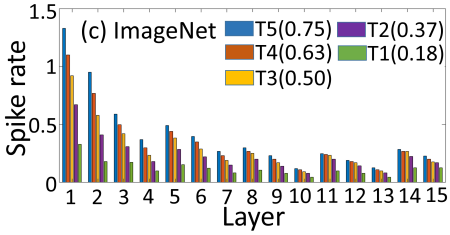
Reference	Dataset	Training	Architecture	Accuracy(%)	Timesteps
(Hunsberger & Eliasmith, 2015)	CIFAR10	Conversion	2C, 2L	82.95	6000
(Cao et al., 2015)	CIFAR10	Conversion	3C, 2L	77.43	400
(Sengupta et al., 2019)	CIFAR10	Conversion	VGG16	91.55	2500
(Lee et al., 2020)	CIFAR10	SGB	VGG9	90.45	100
(Rueckauer et al., 2017)	CIFAR10	Conversion	4C, 2L	90.85	400
(Rathi et al., 2020)	CIFAR10	Hybrid	VGG9	90.5	100
(Park et al., 2020)	CIFAR10	TTFS	VGG16	91.4	680
(Park et al., 2019)	CIFAR10	Burst-coding	VGG16	91.4	1125
(Kim et al., 2018)	CIFAR10	Phase-coding	VGG16	91.2	1500
(Wu et al., 2018)	CIFAR10	SGB	2C, 2L	50.7	30
(Wu et al., 2019b)	CIFAR10	SGB	5C, 2L	90.53	12
(Wu et al., 2019a)	CIFAR10	Tandem Learning	5C, 2L	90.98	8
(Zhang & Li, 2020)	CIFAR10	SGB	5C, 2L	91.41	5
(Rathi & Roy, 2020)	CIFAR10	Hybrid	VGG16	92.70	5
(Zheng et al., 2021)	CIFAR10	STBP-tdBN	ResNet-19	93.16	6
This work (T1)	**CIFAR10**	**Hybrid**	**VGG16**	**93.05**	**1**
(Lu & Sengupta, 2020)	CIFAR100	Conversion	VGG15	63.2	62
(Rathi et al., 2020)	CIFAR100	Hybrid	VGG11	67.9	125
(Park et al., 2020)	CIFAR100	TTFS	VGG16	68.8	680
(Park et al., 2019)	CIFAR100	Burst-coding	VGG16	68.77	3100
(Kim et al., 2018)	CIFAR100	Phase-coding	VGG16	68.6	8950
(Rathi & Roy, 2020)	CIFAR100	Hybrid	VGG16	69.67	5
This work (T1)	**CIFAR100**	**Hybrid**	**VGG16**	**70.15**	**1**
(Sengupta et al., 2019)	Imagenet	Conversion	VGG16	69.96	2500
(Rueckauer et al., 2017)	Imagenet	Conversion	VGG16	49.61	400
(Rathi et al., 2020)	Imagenet	Hybrid	VGG16	65.19	250
(Wu et al., 2019a)	Imagenet	Tandem Learning	AlexNet	50.22	10
(Lu & Sengupta, 2020)	Imagenet	Conversion	VGG15	66.56	64
(Rathi & Roy, 2020)	Imagenet	Hybrid	VGG16	69.00	5
(Zheng et al., 2021)	Imagenet	STBP-tdBN	ResNet-34	67.05	6
This work (T1)	**Imagenet**	**Hybrid**	**VGG16**	**67.71**	**1**

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 (Sengupta et al., 2019; Hunsberger & Eliasmith, 2015). Surrogate-gradient based BP (Wu et al., 2019b; Lee et al., 2020) reduced it to few tens to hundred, but scaling up to ImageNet was challenging. Hybrid training (Rathi et al., 2020) 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 (Rathi & Roy, 2020; Zheng et al., 2021). The proposed method leverages all these previously proposed techniques and improves upon them by incorporating the sequential training approach to obtain unit latency. We also achieve better performance compared to different SNN encoding schemes such as TTFS (Park et al., 2020), phase (Kim et al., 2018), burst (Park et al., 2019); detailed comparison is provided in Appendix A.3.

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 ($0.9pJ$) (Horowitz, 2014) in 45nm CMOS technology (as shown in Fig. 2(e)). Appendix A.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 $\#\text{SNN}_{\text{ops, q}} = \text{spike rate}_{\text{q}} \times \#\text{ANN}_{\text{ops, q}}$; spike rate_{q} denoting 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 considerably lower operation count in T1 compared to T5. The overall average spike rates using T1 for CIFAR-10, CIFAR-100 and ImageNet on VGG16 are 0.13, 0.15 and 0.18, respectively; all significantly below 5.1 (relative cost of MAC to addition), indicating the energy benefits of T1 SNN over the corresponding ANN. The first layer







(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

Figure 2: Layerwise spike rates for a VGG16 across T5 to T1 (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 technology (Horowitz, 2014).

with direct input encoded SNNs receive analog inputs, hence the operations are same as an ANN at this layer. Considering it, we compute 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 (Han et al., 2015a) compared to FP adds as shown in Fig. 2(e). In multi-timestep SNNs, weights and membrane potentials have to be fetched at every timestep, incurring significant data movement cost. However, most previous works (Park et al., 2020; Rathi & Roy, 2020; Zheng et al., 2021; Rathi et al., 2020) 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 for all neurons, unlike ANNs. However, the memory requirements for T1 SNNs are same as ANNs; T1 SNNs neither require any extra memory to store $\mathrm{V_{mem}}$ for future timesteps, nor do they have any extra memory accesses compared to ANNs as inference is single shot. The actual improvements in energy due to this 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 Rathi & Roy (2020). More generally, our scheme with T1 reduces the number of memory accesses by approximately T× compared to an SNN trained with T timesteps.

Comparison with binary activated ANNs. A key distinction between ANNs and SNNs is the notion of time. However, T1 SNN and ReLU activated ANN both infer in single shot, difference being the neuronal functionality and output precision. This difference reduces further with ANNs using binary activations (ANNb) which also have 0/1 output and full precision weights, like SNNs. Thus, it is interesting to compare the performance of T1 SNN and ANNb as shown in Table 3. We observe that T1 SNN outperforms other ANNb approaches in most cases; our results are slightly lower on CIFAR100 compared to Deng & Gu (2020), however in their case, the ANN output is thresholded to saturate at 2, but the values are analog. Note, in all cases, weights are full precision.

Table 3: Comparison of T1 SNN to ANNb

Reference	Dataset	Accuracy(%)
(Sakr et al., 2018)	CIFAR10	89.6
(Bengio et al., 2013)	CIFAR10	85.2
(Deng & Gu, 2020)	CIFAR10	91.88
This work (T1)	CIFAR10	93.05
(Deng & Gu, 2020)	CIFAR100	70.43
This work (T1)	CIFAR100	70.15

Efficacy of iterative training with shallow networks. As mentioned in section 4, gradual training is required due to spike vanishing at later layers for deep SNNs if direct transition from T5 to T1 is attempted. However, for shallow networks, if there is some propagation of spikes till the

Table 4: Accuracy(%) on CIFAR10 with VGG6

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_BN and D_nBN denote dataset D with and without batch-norm respectively

T	CIFAR10_BN	CIFAR10_nBN	CIFAR100_BN	CIFAR100_nBN
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

end, training with T1 following direct conversion from ANN might be possible. To validate this, we experiment with a VGG6 on CIFAR10 and the results are shown in Table 4. In this table, VGG6g denotes networks trained with gradual latency reduction and VGG6d denotes networks directly converted from ANN and trained using that particular timestep, so the result in the last row of Table 4 is obtained by converting an ANN to SNN and directly training with 1 timestep. We observe that proposed gradual training scheme provides slightly higher accuracy compared to direct training in case of shallower networks, though it increases the training overhead. This is consistent with ANN domain results (Furlanello et al., 2018; Han et al., 2015b), where the authors achieve better performing networks using sequential model compression compared to direct compression.

Proposed method with and without batch-norm. Recent works have achieved low latency by adopting batch-normalization (BN) suitably in SNNs (Zheng et al., 2021; Kim & Panda, 2020). To disentangle the effect of BN from the proposed gradual latency reduction 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 are able to perform training with T5 to T1 irrespective of using BN during ANN training. Using BN enhances accuracy, but sequential latency reduction can be performed independently from it. Also, Zheng et al. (2021) report that using threshold-dependent BN allows reducing latency up to a minimum of 6 timesteps, but we can go up to 1. Note, in our case, BN is used only during ANN training and the BN parameters are fused with the weights during ANN-SNN conversion as proposed in Rueckauer et al. (2017); BN is not used in SNN domain training, so the activations at the output of each layer remain binary.

Skipping intermediate latency reduction steps. Since the proposed scheme increases training overhead due to sequential retraining, it is worth investigating if this cost can be reduced by skipping in between latency 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 latencies 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, indicating if the end goal is obtaining T1 SNN, training overhead can be reduced by skipping intermediate steps. However, in this paper, we report the whole spectrum of accuracy-energy-latency trade-off from T5 to T1.

Is training following direct conversion from ANN with T1 thresholds feasible? Though the extra training overhead in our method does not affect our primary goal (inference efficiency) , it would be better if direct conversion from ANN to T1 would be feasible. This is challenging due to layerwise spike decay as discussed in detail in section 4. In this part, we revisit this from a different angle. First, we obtain the layerwise V_{th} for T1. Then, using this set of V_{th}, we convert a VGG16 ANN to SNN, on CIFAR10 and CIFAR100 to investigate whether these thresholds trained for T1 can enable training with 1 timestep directly following ANN-SNN conversion. However, training failure occurs in this case too, since during T1 training, the weights (w) get modified in addition to the thresholds, and layerwise spike propagation depends on both V_{th} and w. Even if training had converged in this case, the challenge would remain how to obtain the suitable V_{th} for T1 without the proposed iterative process, however it would indicate the existence of deep T1 SNNs which can be trained by ANN-SNN conversion followed by direct SNN domain T1 training. But in our experiments, we are unable to find such networks, further validating the need of iterative initialization and retraining.

T1 SNN with magnitude based weight pruning. With T1 SNN, we can obtain maximum compression possible in the temporal domain of SNNs, as 1 forward pass is the minimum for any network to produce output. Additionally, we hypothesize that T1 networks might be amenable to spatial pruning as there is significant redundancy in the layerwise weights of DNNs (Han et al., 2015b;a). To investigate this, we perform experiments on T1 with magnitude based weight pruning (Han et al., 2015a). Our result indicate that we can remove 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.

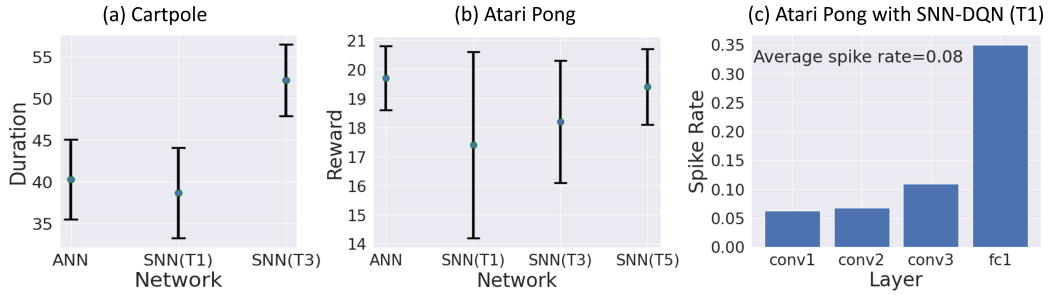


Figure 3: Average reward (errorbars depict mean±std), using DQN with ANN and SNN on (a) Cartpole and (b) Atari Pong, (c) layerwise spike rate with SNN-DQN (T1) on Atari Pong.

This provides evidence that spatial pruning techniques can be combined with T1 SNNs to achieve holistic spatio-temporal compression, albeit with a slight hit to classification accuracy.

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 Tan et al. (2020) 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. Experiments are performed using deep Q-networks (DQN) (Mnih et al., 2015) 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 with 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 SNN outperforms the ANN. Similar performance improvement over ANN using SNN for some RL tasks has been reported in Tan et al. (2020). Next, we experiment with a more complex task, Atari pong game; in this case, T1 SNN achieves reward of 17.4±3.2 compared to ANN based DQN's 19.7±1.1. However, with T5 SNN, we obtain comparable reward (19.4±1.3) to ANN. Notably, we can obtain SNNs for RL task (pong) with significantly lower latency compared to prior art. On pong, Tan et al. (2020) reports reward of 19.8±1.3 using 500 timesteps, whereas, we obtain 19.4±1.3 reward using just 5 timesteps. We believe our training method enables SNNs for RL tasks with comparable performance to ANNs with ~ 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. The details of network topologies used, training setup and additional results are given in appendix A.5.

6 Conclusion

Bio-plausible SNNs hold promise as energy efficient alternatives to ANNs. However, to make SNNs suitable for edge deployment, mitigating high inference latency is critical. To that end, we propose an iterative initialization and retraining approach which enables SNNs to infer in a single shot while maintaining comparable performance to ANNs. We first train an ANN followed by ANN-SNN conversion. Then SNN domain training with surrogate gradient based backpropagation is performed with T timesteps (T>1) using direct input encoding and using neuron threshold and leak as trainable parameters along with weights. At each subsequent stage of training, latency is reduced gradually using the network trained at previous stage as initialization. This iterative training process results in a continuum of SNNs from higher timesteps (T>1) up to unit latency (T1). We observe that T1 SNNs provide the best efficiency and latency; however, the associated trade-off is a slightly lower classification performance with T1 compared to SNNs with higher timesteps. In particular, the T1 SNNs enhance computational energy efficiency by 25X on Imagenet compared to ANNs with similar accuracy. Furthermore, in comparison with state-of-the-art SNNs, T1 SNNs reduce latency by 5X without significant performance degradation on Imagenet. Moreover, T1 SNNs are able to eliminate the memory access overhead associated with fetching membrane potentials and improve compute efficiency by reducing the cumulative spike count significantly compared to SNNs with high latency. We also investigate the applicability of the proposed technique to SNN-based reinforcement learning (RL) with low latency and observe that SNN based DQNs can infer with unit latency while maintaining satisfactory performance on tasks like Cartpole and Atari pong game.

Reproducibility Statement. We submit our implementation as part of the supplementary material. Also, the experimental details including the hyperparameters and architectural descriptions are provided in Appendix section A.2.

REFERENCES

Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. In *Advances in Neural Information Processing Systems*, pp. 787–797, 2018.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Yongqiang Cao, Yang Chen, and Deepak Khosla. Spiking deep convolutional neural networks for energy-efficient object recognition. *International Journal of Computer Vision*, 113(1):54–66, 2015.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Li Deng and Yang Liu. *Deep learning in natural language processing*. Springer, 2018.

Shikuang Deng and Shi Gu. Optimal conversion of conventional artificial neural networks to spiking neural networks. In *International Conference on Learning Representations*, 2020.

Peter U Diehl, Daniel Neil, Jonathan Binas, Matthew Cook, Shih-Chii Liu, and Michael Pfeiffer. 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.

Charlotte Frenkel. Sparsity provides a competitive advantage. *Nature Machine Intelligence*, 3(9): 742–743, 2021.

Tommaso Furlanello, Zachary Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. In *International Conference on Machine Learning*, pp. 1607–1616. PMLR, 2018.

Bing Han, Gopalakrishnan Srinivasan, and Kaushik Roy. 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.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015b.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, 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.

Mark Horowitz. 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.

Dongsung Huh and Terrence J Sejnowski. Gradient descent for spiking neural networks. In *Advances in Neural Information Processing Systems*, pp. 1433–1443, 2018.

Eric Hunsberger and Chris Eliasmith. Spiking deep networks with lif neurons. *arXiv preprint arXiv:1510.08829*, 2015.

Jaehyun Kim, Heesu Kim, Subin Huh, Jinho Lee, and Kiyoung Choi. Deep neural networks with weighted spikes. *Neurocomputing*, 311:373–386, 2018.

Youngeun Kim and Priyadarshini Panda. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *arXiv preprint arXiv:2010.01729*, 2020.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Chankyu Lee, Syed Shakib Sarwar, Priyadarshini Panda, Gopalakrishnan Srinivasan, and Kaushik Roy. Enabling spike-based backpropagation for training deep neural network architectures. *Frontiers in Neuroscience*, 14, 2020.

Da Li, Xinbo Chen, Michela Becchi, and Ziliang Zong. 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.

Sen Lu and Abhronil Sengupta. Exploring the connection between binary and spiking neural networks. *arXiv preprint arXiv:2002.10064*, 2020.

Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. Surrogate gradient learning in spiking neural networks. *IEEE Signal Processing Magazine*, 36:61–63, 2019.

Seongsik Park, Seijoon Kim, Hyeokjun Choe, and Sungroh Yoon. 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.

Seongsik Park, Seijoon Kim, Byunggook Na, and Sungroh Yoon. T2fsnn: Deep spiking neural networks with time-to-first-spike coding. *arXiv preprint arXiv:2003.11741*, 2020.

Nitin Rathi and Kaushik Roy. Diet-snn: Direct input encoding with leakage and threshold optimization in deep spiking neural networks. *arXiv preprint arXiv:2008.03658*, 2020.

Nitin Rathi, Gopalakrishnan Srinivasan, Priyadarshini Panda, and Kaushik Roy. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=B1xSperKvH`.

Kaushik Roy, Akhilesh Jaiswal, and Priyadarshini Panda. Towards spike-based machine intelligence with neuromorphic computing. *Nature*, 575(7784):607–617, 2019.

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*, 11:682, 2017.

Charbel Sakr, Jungwook Choi, Zhuo Wang, Kailash Gopalakrishnan, and Naresh Shanbhag. 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.

Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. Going deeper in spiking neural networks: Vgg and residual architectures. *Frontiers in neuroscience*, 13:95, 2019.

Sumit Bam Shrestha and Garrick Orchard. Slayer: Spike layer error reassignment in time. In *Advances in Neural Information Processing Systems*, pp. 1412–1421, 2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Weihao Tan, Devdhar Patel, and Robert Kozma. Strategy and benchmark for converting deep q-networks to event-driven spiking neural networks. *arXiv preprint arXiv:2009.14456*, 2020.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29:2074–2082, 2016.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Jibin Wu, Yansong Chua, Malu Zhang, Guoqi Li, Haizhou Li, and Kay Chen Tan. A tandem learning rule for efficient and rapid inference on deep spiking neural networks. *arXiv*, pp. arXiv–1907, 2019a.

Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, and Luping Shi. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12:331, 2018.

Yujie Wu, Lei Deng, Guoqi Li, Jun Zhu, Yuan Xie, and Luping Shi. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 1311–1318, 2019b.

Chenglin Yang, Lingxi Xie, Siyuan Qiao, and Alan L Yuille. Training deep neural networks in generations: A more tolerant teacher educates better students. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 5628–5635, 2019.

Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541, 2018.

Wenrui Zhang and Peng Li. Temporal spike sequence learning via backpropagation for deep spiking neural networks. *Advances in Neural Information Processing Systems*, 33, 2020.

Hanle Zheng, Yujie Wu, Lei Deng, Yifan Hu, and Guoqi Li. Going deeper with directly-trained larger spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11062–11070, 2021.

A APPENDIX

A.1 TRAINING METHODOLOGY

A.1.1 SURROGATE-GRADIENT BASED LEARNING

To train deep SNNs, surrogate-gradient based backpropagation through time (BPTT) (Werbos, 1990) is used to perform temporal as well as spatial error credit assignment. Spatial credit assignment is achieved by spatial error distribution across layers, while the network is unrolled in time for temporal credit assignment. The output layer neuronal dynamics is governed by-

$$u_i^t = u_i^{t-1} + \sum_j w_{ij} o_j^t, \tag{5}$$

here u_i corresponds to the membrane potential of i-th neuron of final (L-th) layer. The final layer neurons just accumulate the potential over time for classification purpose, without emitting spikes. We pass these accumulated final layer outputs through a softmax layer which gives the class-wise probability distribution and then the loss is calculated using the cross-entropy between the true output and the network's predicted distribution. The governing equations are-

$$Loss = -\sum_i y_i log(s_i), \tag{6}$$

$$s_i = \frac{e^{u_i^T}}{\sum_{k=1}^{N} e^{u_k^T}}, \tag{7}$$

where Loss represents the loss function, y denotes true label, s is the predicted label, T is the total number of timesteps and N is the number of classes. The derivative of the loss w.r.t. the membrane potential of the neurons in the final layer is-

$$\frac{\partial Loss}{\partial \boldsymbol{u}_L^T} = \boldsymbol{s} - \boldsymbol{y}, \tag{8}$$

and the weight updates at the output layer are done as-

$$w_{ij,L} = w_{ij,L} - \eta \Delta w_{ij,L}, \tag{9}$$

$$\Delta w_{ij,L} = \sum_t \frac{\partial Loss}{\partial w_{ij,L}^t} = \sum_t \frac{\partial Loss}{\partial \boldsymbol{u}_L^T} \frac{\partial \boldsymbol{u}_L^T}{\partial w_{ij,L}^t} = \frac{\partial Loss}{\partial \boldsymbol{u}_L^T} \sum_t \frac{\partial \boldsymbol{u}_L^T}{\partial w_{ij,L}^t}, \tag{10}$$

where η is the learning rate, and $w_{ij,L}^t$ denotes the weight between i-th neuron at layer L and j-th neuron at layer $L-1$ at timestep t. Output layer neurons are non-spiking, hence the non-differentiability issue is not involved here. The hidden layer parameter update is given by-

$$\Delta w_{ij,k} = \sum_t \frac{\partial Loss}{\partial w_{ij,k}^t} = \sum_t \frac{\partial Loss}{\partial o_{i,k}^t} \frac{\partial o_{i,k}^t}{\partial u_{i,k}^t} \frac{\partial u_{i,k}^t}{\partial w_{ij,k}^t}, \quad k = 2, 3, ...L-1 \tag{11}$$

where $o_{i,k}^t$ is the spike-generating function (Eqn. 3), k is layer index. We approximate the gradient of this function w.r.t. its input using the linear surrogate-gradient (Bellec et al., 2018) as-

$$\frac{\partial o}{\partial u} = \gamma max\{0, 1 - |\frac{u-v}{v}|\}, \tag{12}$$

where γ is a hyperparameter chosen as 0.3 in this work. The layerwise threshold is updated using-

$$v_l = v_l - \eta \Delta v_l, \tag{13}$$

$$\Delta v_l = \sum_t \frac{\partial Loss}{\partial v_l} = \sum_t \frac{\partial Loss}{\partial \boldsymbol{o}_l^t} \frac{\partial \boldsymbol{o}_l^t}{\partial \boldsymbol{z}_l^t} \frac{\partial \boldsymbol{z}_l^t}{\partial v_l} = \sum_t \frac{\partial Loss}{\partial \boldsymbol{o}_l^t} \frac{\partial \boldsymbol{o}_l^t}{\partial \boldsymbol{z}_l^t} \left(\frac{-v_l \boldsymbol{o}_l^{t-1} - \boldsymbol{u}_l^t}{v_l^2} \right) \tag{14}$$

and during training stages where more than one timestep is involved, the leak is updated as-

$$\lambda_l = \lambda_l - \eta \Delta \lambda_l, \quad \text{and} \quad \Delta \lambda_l = \sum_t \frac{\partial Loss}{\partial \lambda_l} = \sum_t \frac{\partial Loss}{\partial \boldsymbol{o}_l^t} \frac{\partial \boldsymbol{o}_l^t}{\partial \boldsymbol{u}_l^t} \frac{\partial \boldsymbol{u}_l^t}{\partial \lambda_l} = \sum_t \frac{\partial Loss}{\partial \boldsymbol{o}_l^t} \frac{\partial \boldsymbol{o}_l^t}{\partial \boldsymbol{u}_l^t} \boldsymbol{u}_l^{t-1} \tag{15}$$

Algorithm 2 Procedure of spike-based learning with backpropagation for an iteration.

Input: pixel-based mini-batch of input ($\boldsymbol{X}$) - target ($\boldsymbol{Y}$) pairs, total number of timesteps (T), number of layers (L), pre-trained ANN weights ($\boldsymbol{W}$), membrane potential ($\boldsymbol{U}$), layer-wise membrane leak constants ($\boldsymbol{\lambda}$), layer-wise firing thresholds ($\boldsymbol{V}$), learning rate (η)
Initialize: $\boldsymbol{U}_l^t = 0,\ \forall l = 1, ..., L$
// Forward Phase
for $t \leftarrow 1$ **to** T **do**
  $\boldsymbol{O}_1^t = \boldsymbol{X}$;
  for $l \leftarrow 2$ **to** $L-1$ **do**
    // membrane potential integrates weighted sum of spike-inputs
    $\boldsymbol{U}_l^t = \lambda_l \boldsymbol{U}_l^{t-1} + \boldsymbol{W}_l * \boldsymbol{O}_{l-1}^t$
    if $\boldsymbol{U}_l^t > V_l$ **then**
      // if membrane potential exceeds V_l, a neuron fires a spike
      $\boldsymbol{O}_l^t = 1,\ \boldsymbol{U}_l^t = \boldsymbol{U}_l^t - V_l$
    else
      // else, output is zero
      $\boldsymbol{O}_l^t = 0$
    end if
  end for
  // final layer neurons do not fire
  $\boldsymbol{U}_L^t = \boldsymbol{U}_L^{t-1} + \boldsymbol{W}_L * \boldsymbol{O}_{L-1}^t$
end for
//calculate loss, Loss=cross-entropy($\boldsymbol{U}_L^T, \boldsymbol{Y}$)
// Backward Phase
for $t \leftarrow T$ **to** 1 **do**
  for $l \leftarrow L-1$ **to** 1 **do**
    // evaluate partial derivatives of loss with respect to the trainable parameters by unrolling the network over time
    $\triangle \boldsymbol{W}_l^t = \frac{\partial Loss}{\partial \boldsymbol{O}_l^t} \frac{\partial \boldsymbol{O}_l^t}{\partial \boldsymbol{U}_l^t} \frac{\partial \boldsymbol{U}_l^t}{\partial \boldsymbol{W}_l^t},\ \Delta V_l^t = \frac{\partial Loss}{\partial \boldsymbol{O}_l^t} \frac{\partial \boldsymbol{O}_l^t}{\partial \boldsymbol{z}_l^t} \frac{\partial \boldsymbol{z}_l^t}{\partial V_l},\ \Delta \lambda_l^t = \frac{\partial Loss}{\partial \boldsymbol{o}_l^t} \frac{\partial \boldsymbol{o}_l^t}{\partial \boldsymbol{u}_l^t} \frac{\partial \boldsymbol{u}_l^t}{\partial \lambda_l}$
  end for
end for
//update the parameters $\boldsymbol{W}_l = \boldsymbol{W}_l - \eta \sum_t \Delta \boldsymbol{W}_l^t, V_l = V_l - \eta \sum_t \Delta V_l^t, \lambda_l = \lambda_l - \eta \sum_t \Delta \lambda_l^t$

A.2 Experimental details

A.2.1 Network architecture

We modify the VGG and ResNet architectures slightly to facilitate ANN-SNN conversion. 3 plain convolutional layers of 64 filters are appended after the input layer in the ResNet architecture (Sengupta et al., 2019). In all cases, average pooling (2×2) is used and the basic block in ResNet employs a stride of 2 when the number of filters increases. 1×1 convolution is used in the shortcut path of basic blocks where the number of filters is different in input and output. The architectural details are-

VGG6: {64, A, 128, 128, A}, Linear

VGG16: {64, D, 64, A, 128, D, 128, A, 256, D, 256, D, 256, A, 512, D, 512, D, 512, A, 512, D, 512, D, 512}, Linear

ResNet20: {64, D, 64, D, 64, A, 64BB, 64BB, 128BB (/2), 128BB, 256BB (/2), 256BB, 512BB (/2), 512BB}

BB: basic block, Linear: {4096, D, 4096, D, number of classes}, D: Dropout (probability (p)-ANN: 0.5, SNN: 0.2), A: Average Pooling (kernel size = 2× 2)

A.2.2 Training Hyperparameters

Standard data augmentation techniques are applied for image datasets such as padding by 4 pixels on each side, and 32×32 cropping by randomly sampling from the padded image or its horizontally flipped version (with 0.5 probability of flipping). The original 32×32 images are used during testing.

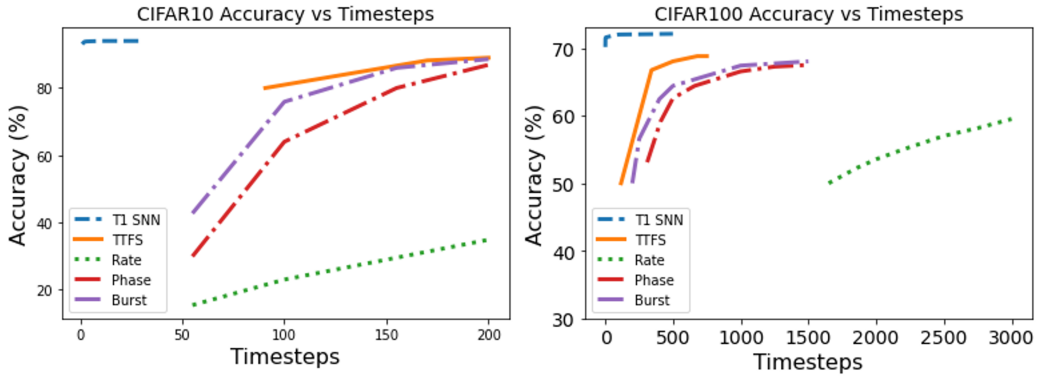


Figure 4: Accuracy versus latency curve for various SNN coding methods on VGG16, the values for TTFS (Park et al., 2020), phase (Kim et al., 2018), burst (Park et al., 2019) and rate (Rueckauer et al., 2017) have been adopted from (Park et al., 2020).

.

Both training and testing data are normalized using channel-wise mean and standard deviation calculated from training set. The ANNs are trained with cross-entropy loss with stochastic gradient descent optimization (weight decay=0.0005, momentum=0.9). We train the ANNs for 500 and 90 epochs for CIFAR and ImageNet respectively, with an initial learning rate of 0.01. The learning rate is divided by 5 at epochs of 0.45, 0.7 and 0.9 fraction of total epochs. The ANNs are trained with batch-norm (BN) and the BN parameters are fused with the layerwise weights during ANN-SNN conversion following Rueckauer et al. (2017), we do not have bias terms as BN is used. Additionally, dropout (Srivastava et al., 2014) is used as the regularizer with a constant dropout mask with dropout probability=0.5 across all timesteps while training the SNNs. Since max-pooling causes significant information loss in SNNs (Diehl et al., 2015), we use average-pooling layers to reduce the feature maps. During ANN training, the weights are initialized using He initialization (He et al., 2015). Upon conversion, at each training iteration with a certain timestep, the SNNs are trained for 300 epochs with cross-entropy loss and adam optimizer (weight decay=0). Initial learning rate is chosen as 0.0001, which is divided by 5 at epochs of 0.6, 0.8 and 0.9 fraction of total epochs. While training the SNNs, dropout probability is kept at 0.2.

A.3 Comparison of accuracy versus latency trade-off with various encoding methods

In this part, we compare our results with various SNN encoding schemes in terms of timesteps required to reach convergence for a VGG16 network and the results are shown in Fig. 4. The figure demonstrates the results of "T2FSNN" encoding (Park et al., 2020), where the timing of spikes carries information and other rate and temporal coding schemes including "Rate" (Rueckauer et al., 2017), "Phase" (Kim et al., 2018), and "Burst" coding (Park et al., 2019). The left plot in Fig. 4 shows $\sim$ 200 timesteps is needed for the fastest convergence among these encoding methods for CIFAR10, whereas, we achieve 93.05% accuracy in just 1 timestep. Similarly, we obtain 70.15% accuracy in 1 timestep with VGG16 on CIFAR100. The graph on the right in Fig. 4 shows the results for VGG16 on CIFAR100 using "T2FSNN", "Burst", "Phase" and "Rate". As can be seen, "T2FSNN" reaches 68% roughly at 500 steps, "Burst" at 1500, "Phase" at 2000, and "Rate" fails to cross 60% even at 3000 timesteps. Notably, we are not only able to reduce latency by 2 to 3 orders of magnitude compared to these works, but also outperform them in top-1 accuracy by $\sim 2\%$ using unit inference latency. So, T1 SNN enhances the performance on both ends of the accuracy-latency trade-off compared to these other approaches. Moreover, while the temporal methods enhance computational efficiency by reducing the spike count, the high latency issue persists which incurs high memory access overhead due to the requirement of fetching membrane potential of each neuron at every timestep. Since our proposed scheme enables one step inference, the neurons are automatically limited to maximum one spike per neuron like "T2FSNN" (Park et al., 2020), but our inference latency is significantly lower and no extra memory access operations are involved for fetching membrane potentials of previous timesteps.

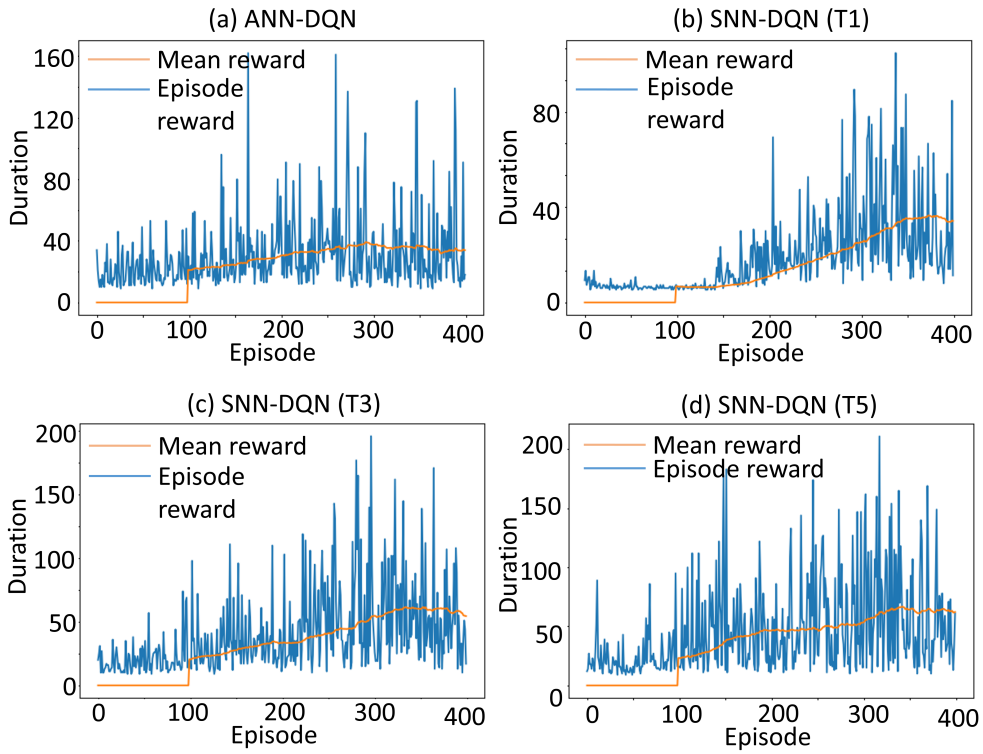


Figure 5: Rewards during training on Cartpole environment with- (a) ANN-DQN and (b) SNN-DQN (T1), (c) SNN-DQN (T3), and (d) SNN-DQN (T5).

A.4 Computational Cost

The number of operations in an ANN layer is given as-

$$\#\text{ANN}_{\text{ops}} = \begin{cases} k_w \times k_h \times c_{in} \times h_{out} \times w_{out} \times c_{out}, & \text{Conv layer} \\ n_{in} \times n_{out}, & \text{Linear layer} \end{cases}$$

where $k_w(k_h)$ denote filter width (height), $c_{in}(c_{out})$ is number of input (output) channels, $h_{out}(w_{out})$ is the height (width) of the output feature map, and $n_{in}(n_{out})$ is the number of input (output) nodes.

A.5 Additional Results on Reinforcement Learning

In this section, we provide supplementary results on our experiments on RL tasks (Cartpole and Atari pong). The details of network architectures, training hyperparameters and training dynamics are described in the following.

A.5.1 Results on Cartpole

For the cartpole task, we use a small network with 3 convolutional and a fully connected layer. The first convolutional layer has 16 5X5 filters with stride 2, second and third convolutional layers have 32 5X5 filters with stride 2 each. The fully connected layer has 2 neurons, corresponding to the number of actions. We use RMSProp as optimizer for training. The networks are trained with batch size of 128, discount factor (γ) of 0.999 and replay memory of 10000. Figure 5 shows the results for the cartpole environment. We repeat all experiments for 400 episodes, in this game, the duration for which the agent is able to continue the game (keep the balance), is the reward; so higher game duration means better performance. The blue trajectory shows the rewards at each episode, which has some variation across episodes, the yellow curve shows the accumulated average reward. As can be seen, the SNN outperforms the ANN in terms of reward with just 3 timesteps. For SNN-DQN with 1 timestep (T1), the SNNs performs slightly worse compared to ANN-DQN. As we increase the simulation time-window (number of timesteps), the SNN performance improves further

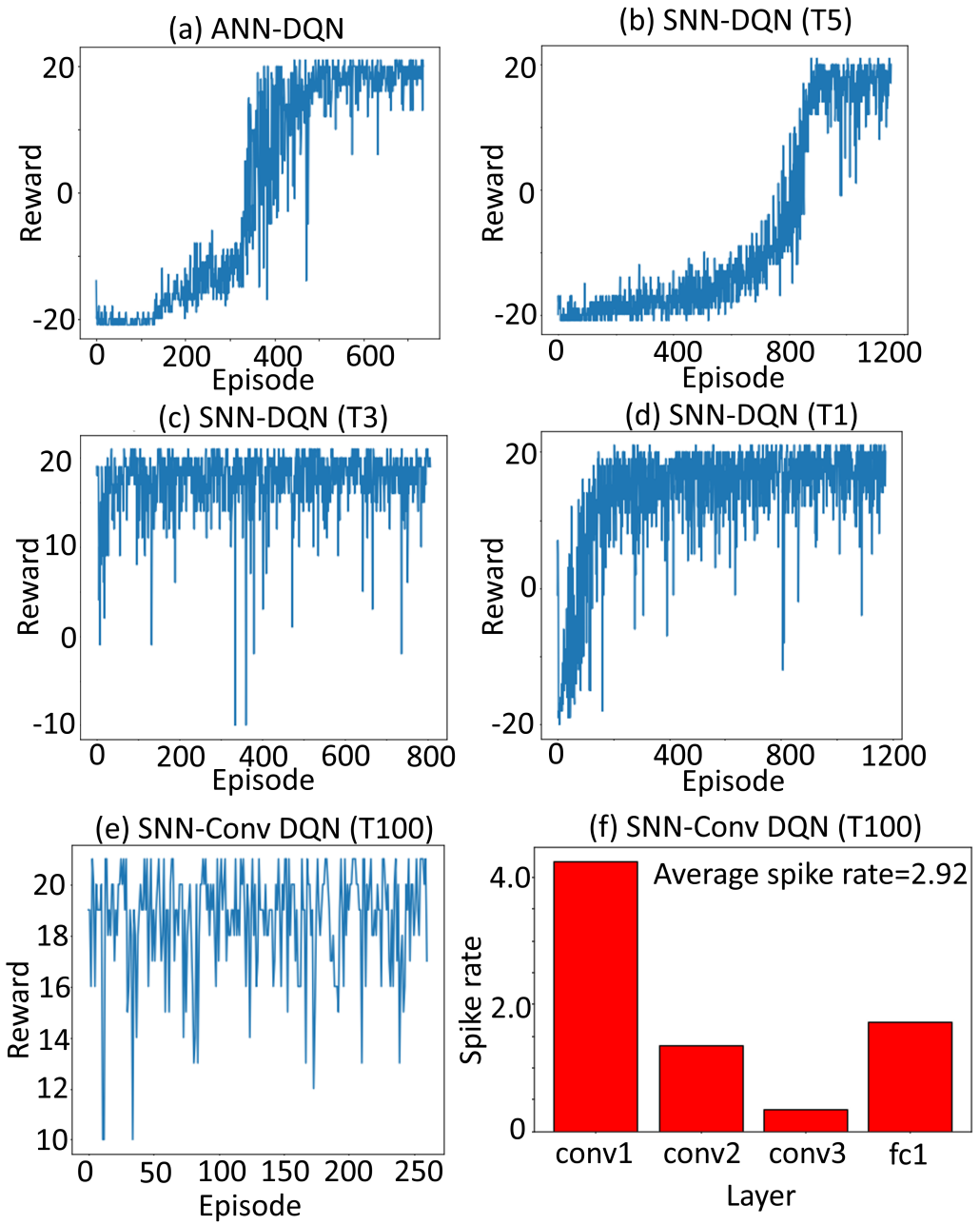


Figure 6: Rewards during training on Atari pong environment with- (a) ANN-DQN and (b) SNN-DQN (T5), (c) SNN-DQN (T3), and (d) SNN-DQN (T1), (e) SNN-Conv DQN (T100), and (f) layerwise spike rate for the network in (e).

as expected, with the average reward for T5 being 52.2±4.3 (mean±standard deviation), whereas for the ANN, this metric is 40.3±4.8. We anticipate this improvement in multi-timestep SNNs is due to inherent memory of previous inputs which results from the residual membrane potential in the spiking neurons. As a result, SNNs offer an advantage compared to feed-forward ANNs for tasks with a sequential component. In RL tasks, since the decision-making is sequential and past frames possess some information about what the next plausible state and corresponding action could be, SNNs might be better suited to leverage such kind of environment. However, cartpole balancing is a very simple task, so to further explore the potential of the proposed method in obtaining low latency solutions for SNN-based RL, we next apply the proposed SNN-DQN framework to Atari pong environment.

A.5.2 RESULTS ON ATARI PONG

Atari pong is a two-dimensional gym environment that simulates table tennis. Here, the agent steps through the environment by observing frames of the game (reshaped to 84X84 pixels), interacts with the environment with 6 possible actions, and receives feedback in the form of the change in the game score. For our experiments, we first train an ANN based deep Q network (DQN), where we use the same DQN proposed in Mnih et al. (2015) with 3 convolution layers and 2 fully connected layers. The first convolution layer has 32 8X8 filters with stride 4. The second convolution layer has 64 4X4 with stride 2. The third convolution layer has 64 3X3 filters with stride 1. The fully connected layer has 512 neurons. The number of neurons in the final layer for any game depends on the number of valid actions for that game, which in this case is 6. Training is performed with Adam optimizer with learning rate 0.00001, batch size 128, discount factor(γ) 0.99 and replay buffer size 100000. Results of our experiments are shown in Fig. 6. Rewards obtained during ANN-DQN training are depicted in Fig. 6 (a). As mentioned in the discussion of RL-SNN in section 5 of the main manuscript, ANN-DQN achieves reward of 19.7 ± 1.1. The training dynamics using T5, T3 and T1 SNN DQN are shown in Fig. 6 (b), (c), (d) respectively. Reward and efficiency analysis of these networks compared to ANN-DQN is given in section 5 of the main manuscript. Additionally, we report the performance of DQNs with converted SNNs in Fig. 6 (e). Note, these SNNs do not undergo any training in SNN domain, rather they are converted from corresponding ANN-DQNs and used for inference. Reward obtained the converted SNN with 100 timesteps is 19.4 ± 1.3, so converted SNN-DQNs (SNN-conv DQNs) perform comparably to ANN-DQNs as also reported in Tan et al. (2020), however the bottleneck is they require $\sim$100 timesteps for high performance. Using the proposed method, we obtain T5 SNN-DQNs with comparable performance to their ANN counterparts, but with considerably lower compute cost (5.22X). We have reported the spike rates for T1 SNN-DQN in Fig. 3 (c). For comparison purposes, we also consider the layerwise spike rate for the converted SNN-DQN as shown in Fig. 6 (f), of the main text. In this case, the average spike rate is 2.92, which leads to 1.75X higher energy efficiency for the SNN-conv DQN (T100) compared to ANN-DQN. As a result, T5 SNN DQN provides 2.98X improvement in energy efficiency over SNN-conv DQNs as proposed in Tan et al. (2020), while achieving comparable performance.