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

Anonymous ECCV submission

Paper ID 6421

1 Training Methodology

1.1 Surrogate-gradient based learning

To train deep SNNs, surrogate-gradient based backpropagation through time (BPTT) [12] 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-

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

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-

$$Loss = -\sum_i y_i log(s_i), \tag{2}$$

$$s_i = \frac{e^{u_i^T}}{\sum_{k=1}^{N} e^{u_k^T}}, \tag{3}$$

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-

$$\frac{\partial Loss}{\partial \boldsymbol{u}_L^T} = \boldsymbol{s} - \boldsymbol{y}, \tag{4}$$

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

$$w_{ij,L} = w_{ij,L} - \eta \Delta w_{ij,L}, \tag{5}$$

$$\begin{aligned}\Delta w_{ij,L} = \sum_t \frac{\partial Loss}{\partial w_{ij,L}^t} &= \sum_t \frac{\partial Loss}{\partial \boldsymbol{u}_L^T} \frac{\partial \boldsymbol{u}_L^T}{\partial w_{ij,L}^t} \\ &= \frac{\partial Loss}{\partial \boldsymbol{u}_L^T} \sum_t \frac{\partial \boldsymbol{u}_L^T}{\partial w_{ij,L}^t},\end{aligned} \tag{6}$$

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-

$$\Delta w_{ij,k} = \sum_t \frac{\partial Loss}{\partial w_{ij,k}^t} = \sum_t \frac{\partial Loss}{\partial o_{i,k}^t} \frac{\partial o_{i,k}^t}{\partial u_{i,k}^t} \frac{\partial u_{i,k}^t}{\partial w_{ij,k}^t}, k = 2, 3, ...L-1 \tag{7}$$

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 [1] as-

$$\frac{\partial o}{\partial u} = \gamma max\{0, 1 - |\frac{u-v}{v}|\}, \tag{8}$$

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

$$v_l = v_l - \eta \Delta v_l, \tag{9}$$

$$\begin{aligned}\Delta v_l = \sum_t \frac{\partial Loss}{\partial v_l} &= \sum_t \frac{\partial Loss}{\partial \boldsymbol{o}_l^t} \frac{\partial \boldsymbol{o}_l^t}{\partial \boldsymbol{z}_l^t} \frac{\partial \boldsymbol{z}_l^t}{\partial v_l} \\ &= \sum_t \frac{\partial Loss}{\partial \boldsymbol{o}_l^t} \frac{\partial \boldsymbol{o}_l^t}{\partial \boldsymbol{z}_l^t} \left(\frac{-v_l \boldsymbol{o}_l^{t-1} - \boldsymbol{u}_l^t}{v_l^2} \right)\end{aligned} \tag{10}$$

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

$$\begin{aligned}\lambda_l = \lambda_l - \eta \Delta \lambda_l, \text{and} \Delta \lambda_l = \sum_t \frac{\partial Loss}{\partial \lambda_l} &= \sum_t \frac{\partial Loss}{\partial \boldsymbol{o}_l^t} \frac{\partial \boldsymbol{o}_l^t}{\partial \boldsymbol{u}_l^t} \frac{\partial \boldsymbol{u}_l^t}{\partial \lambda_l} \\ &= \sum_t \frac{\partial Loss}{\partial \boldsymbol{o}_l^t} \frac{\partial \boldsymbol{o}_l^t}{\partial \boldsymbol{u}_l^t} \boldsymbol{u}_l^{t-1}\end{aligned} \tag{11}$$

Algorithm S1 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

    $\triangle \boldsymbol{W}_l^t = \frac{\partial Loss}{\partial \boldsymbol{O}_l^t} \frac{\partial \boldsymbol{O}_l^t}{\partial \boldsymbol{U}_l^t} \frac{\partial \boldsymbol{U}_l^t}{\partial \boldsymbol{W}_l^t},\quad \triangle V_l^t = \frac{\partial Loss}{\partial \boldsymbol{O}_l^t} \frac{\partial \boldsymbol{O}_l^t}{\partial \boldsymbol{z}_l^t} \frac{\partial \boldsymbol{z}_l^t}{\partial V_l},\quad \triangle \lambda_l^t = \frac{\partial Loss}{\partial \boldsymbol{o}_l^t} \frac{\partial \boldsymbol{o}_l^t}{\partial \boldsymbol{u}_l^t} \frac{\partial \boldsymbol{u}_l^t}{\partial \lambda_l}$

  end for

end for

//update the parameters $\boldsymbol{W}_l = \boldsymbol{W}_l - \eta \sum_t \triangle \boldsymbol{W}_l^t, V_l = V_l - \eta \sum_t \triangle V_l^t, \lambda_l = \lambda_l - \eta \sum_t \triangle \lambda_l^t$

2 Experimental details

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 [9]. 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)

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. 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 [8], we do not have bias terms as BN is used. Additionally, dropout [10] 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 [2], we use average-pooling layers to reduce the feature maps. During ANN training, the weights are initialized using He initialization [3]. 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.

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. 1. The figure demonstrates the results of "T2FSNN" encoding [7], where the timing of spikes carries information and other rate and temporal coding schemes including "Rate" [8], "Phase" [4], and "Burst" coding [6]. The left plot in Fig. 1 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. 1 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"

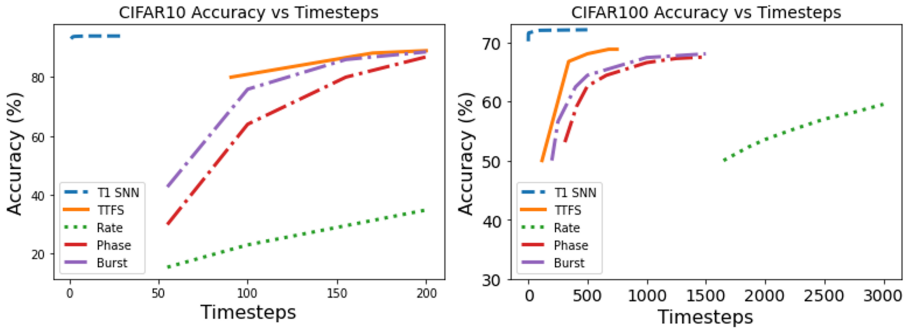

Fig. 1. Accuracy versus latency curve for various SNN coding methods on VGG16, the values for TTFS [7], phase [4], burst [6] and rate [8] have been adopted from [7].

.

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" [7], but our inference latency is significantly lower and no extra memory access operations are involved for fetching membrane potentials of previous timesteps.

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.

5 Additional Results on Static Image Classification

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 of main manuscript. In this part, we revisit this

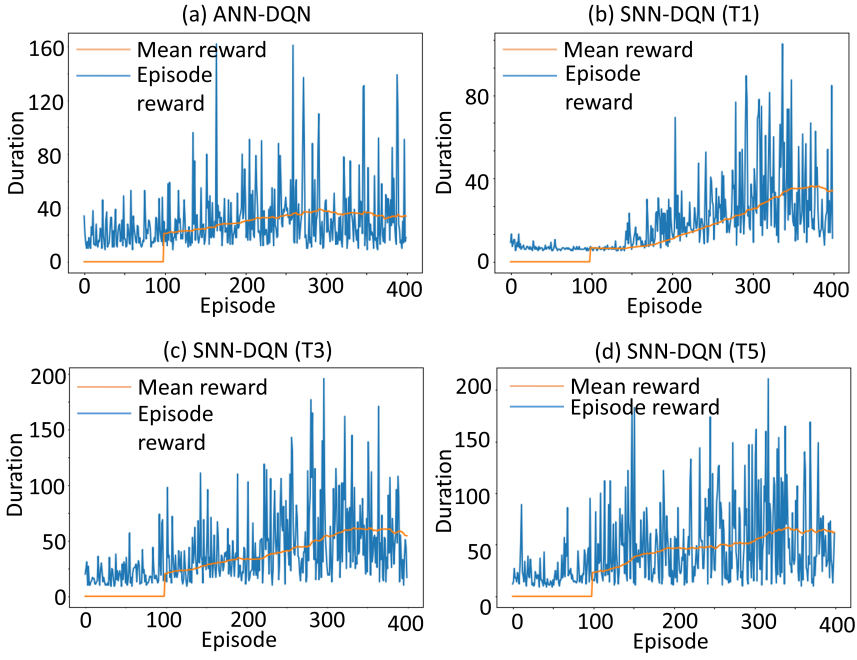

Fig. 2. Rewards during training on Cartpole environment with- (a) ANN-DQN and (b) SNN-DQN (T1), (c) SNN-DQN (T3), and (d) SNN-DQN (T5).

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.

6 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.

6.1 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 2 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 as expected, with the average reward for T5 being 52.2$\pm$4.3 (mean$\pm$standard deviation), whereas for the ANN, this metric is 40.3$\pm$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.

6.2 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 [5] 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. 3. Rewards obtained during ANN-DQN training are depicted in Fig. 3 (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. 3 (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. 3 (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 [11], however the bottleneck is they require ∼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), of the main manuscript. For comparison purposes, we also consider the layerwise spike rate for the converted SNN-DQN as shown in Fig. 3 (f). 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 [11], while achieving comparable performance.

References

1. Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., Maass, W.: Long short-term memory and learning-to-learn in networks of spiking neurons. In: Advances in Neural Information Processing Systems. pp. 787–797 (2018)
2. 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)
3. He, K., Zhang, X., Ren, S., Sun, J.: 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)
4. Kim, J., Kim, H., Huh, S., Lee, J., Choi, K.: Deep neural networks with weighted spikes. Neurocomputing **311**, 373–386 (2018)
5. 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)
6. 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)
7. 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)
8. 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)
9. 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)

10. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research **15**(1), 1929–1958 (2014)
11. 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)
12. Werbos, P.J.: Backpropagation through time: what it does and how to do it. Proceedings of the IEEE **78**(10), 1550–1560 (1990)

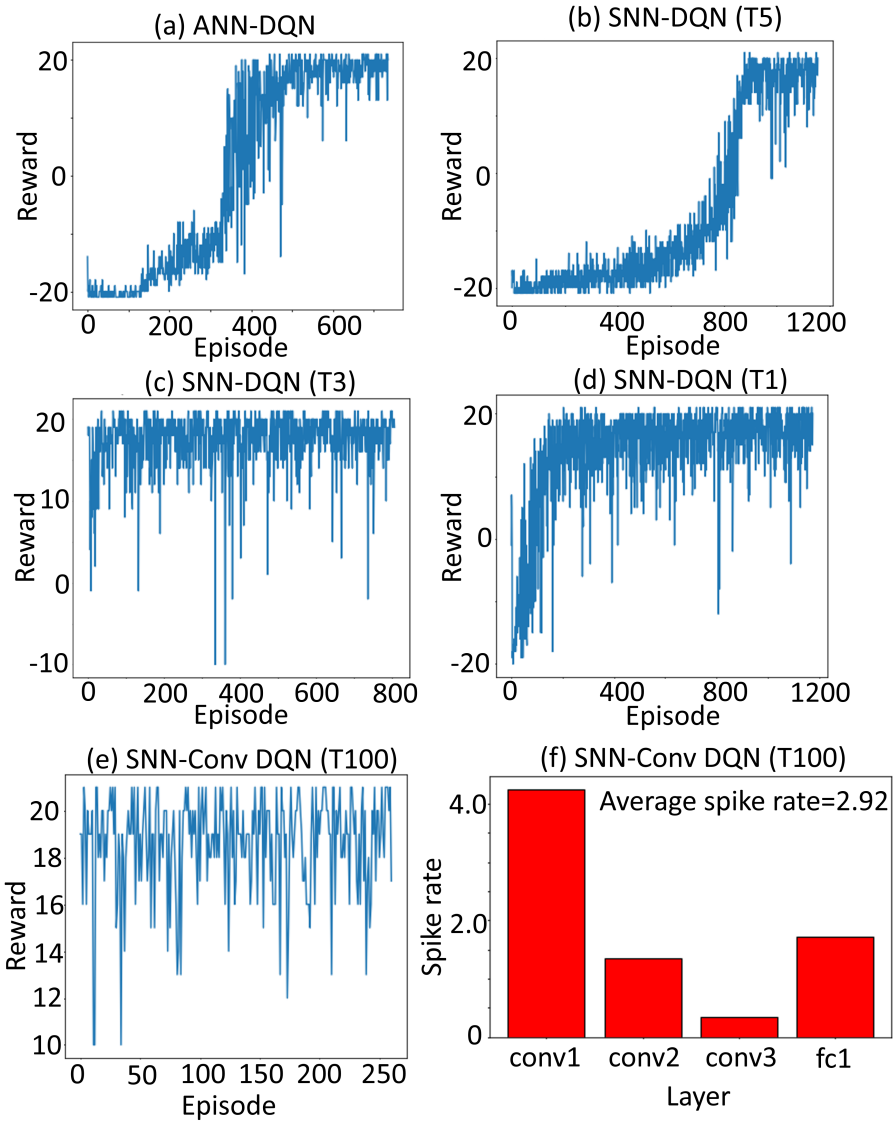

Fig. 3. Rewards during training on Atari pong environment with- (a) ANN-DQN and (b) SNN-DQN (T5), (c) SNN-DQN (T3), and (d) SNN-DQN (T1), (e) SNN-Conv DQN (T100), and (f) layerwise spike rate for the network in (e).