Enhancing Training of Spiking Neural Network with Stochastic Latency

Srinivas Anumasa[1,*], Bhaskar Mukhoty[1,*], Velibor Bojković[1], Giulia De Masi[2,3], Huan Xiong[4,1,†], Bin Gu[5, 1,†]

[1] Mohamed bin Zayed University of Artificial Intelligence, UAE
[2] ARRC, Technology Innovation Institute, UAE
[3] BioRobotics Institute, Sant'Anna School of Advanced Studies Pisa, Italy
[4] Harbin Institute of Technology, China
[5] School of Artificial Intelligence, Jilin University, China

Abstract

Spiking neural networks (SNNs) have garnered significant attention for their low power consumption when deployed on neuromorphic hardware that operates in orders of magnitude lower power than general-purpose hardware. Direct training methods for SNNs come with an inherent latency for which the SNNs are optimized, and in general, the higher the latency, the better the predictive powers of the models, but at the same time, the higher the energy consumption during training and inference. Furthermore, an SNN model optimized for one particular latency does not necessarily perform well in lower latencies, which becomes relevant in scenarios where it is necessary to switch to a lower latency because of the depletion of onboard energy or other operational requirements. In this work, we propose Stochastic Latency Training (SLT), a direct training method for SNNs that optimizes the model for the given latency but simultaneously offers a minimum reduction of predictive accuracy when shifted to lower inference latencies. We provide heuristics for our approach with partial theoretical justification and experimental evidence showing the state-of-the-art performance of our models on datasets such as CIFAR-10, DVS-CIFAR-10, CIFAR-100, and DVS-Gesture. Our code is available at https://github.com/srinuvaasu/SLT

Introduction

Spiking neural networks (SNNs) have attracted significant attention as the next-generation neural network, thanks to their bio-inspired, event-driven, and low-power computational capabilities, supported by specialized neuromorphic hardware such as Loihi (Davies et al. 2018), TrueNorth (DeBole et al. 2019), and Spinnaker (Furber et al. 2014). In a typical artificial neural network (ANN), neurons receive input activation values from the previous layer neurons and compute output activation values as a function of the weighted input activation computed as the inner product between input activations and network weights. In contrast, the SNN neurons send the activation spike to the next layer only when their membrane potential exceeds a predetermined membrane threshold, making the spiking activity event-driven and asynchronous. The neuromorphic hardware allows for a simple accumulation of network weights activated by incoming spikes, eliminating the need for computationally intensive floating-point inner products. Overall, this results in significantly lower energy consumption in neuromorphic hardware compared to their standard counterparts, particularly when spiking activity in SNN neurons is sparse.

While the binary spikes produced by the Heaviside activation offer several benefits, they pose a unique challenge for direct training of SNNs. As the gradient of the Heaviside function is almost always zero and non-differentiable at the transition, the standard back-propagation algorithm does not propagate any gradients backwards when used as it is. To circumvent this issue, surrogate training methods (Wu et al. 2018; Zheng et al. 2021) replace the non-differentiable Heaviside activation function with differentiable surrogate functions such as Sigmoid. Recently, (Mukhoty et al. 2023) have shown applying zeroth-order gradients on the Heaviside, on expectation, is equivalent to choosing different surrogates for the Heaviside.

The temporal dimension of the network adds further challenges to the training algorithm. The temporal dimension increases the training time proportionally and creates a longer path for the back-propagation algorithm, introducing vanishing gradients. The direct training method using surrogate functions involves training many parameters in the state-of-the-art SNNs, which is accomplished with the help of GPUs. These SNNs can then be deployed on neuromorphic hardware for energy-efficient inference. In contrast, standard ANNs are trained on GPUs and may require CPU or GPU support during inference, depending on the specific application requirements.

The energy efficiency of SNN models has attracted real-world applications that require predictions from a standalone model based on a finite energy budget. For example, in robotic control, tasks such as localization, motor control, navigation, etc., require frequent decision-making by a neural network deployed onboard with a limited energy source (Gutierrez-Galan et al. 2020; Strohmer, Manoonpong, and

[*]These authors contributed equally.
[†]Corresponding authors: huan.xiong.math@gmail.com, jsgubin@gmail.com

Larsen 2020; Yamazaki et al. 2022). In case the onboard energy source gets depleted, to continue operation, it can become imminent for the model to shift to a lower inference latency to save energy. Additionally, a model might require shifting between different latencies depending on the external throughput requirements. However, the present SNN models are dedicatedly trained for a particular latency and suffer a loss of accuracy when tested at a lower inference latency. Further, maintaining different models for different latencies can be wasteful in terms of memory and become a system bottleneck.

We propose the Stochastic Latency Training (SLT) method, which chooses the latency for a gradient update uniformly from the entire range of latencies. Thus, SLT finds a single SNN model that offers minimum loss of accuracy when operated on a lower latency. We summarise our contribution as below:

- We propose the SLT method that provides flexibility regarding inference latency. The model offers minimal loss of accuracy when evaluated at lower latencies.
- The SLT method offers significantly faster training time compared to the state-of-the-art methods, as the expected latency used in training with SLT is approximately half compared to the maximum training latency of the model.
- We analyse the loss surface of the model trained using SLT, which shows that we achieve a flatter minimum compared to the models without SLT, explaining better generalizability of the models trained using SLT.
- We benchmark the proposed method on standard datasets, demonstrating that SLT is comparable or better than the state-of-the-art methods at maximum training latency and better when evaluated at lower inference latencies.

Related Work

In recent years, the performance of SNN models has improved significantly on both static and neuromorphic datasets, benefiting majorly from two approaches: ANN-to-SNN conversion and direct training of SNN models.

ANN-to-SNN Conversion: In ANN-to-SNN conversion, an already trained ANN model with ReLU activation (Diehl et al. 2015; Cao, Chen, and Khosla 2015; Rueckauer et al. 2017) or a tailored ANN model (Bu et al. 2021; Deng and Gu 2021) with a new activation function, is converted to SNN model by copying the weights of ANN to SNN and replacing the non-linear activation function with IF neuron (see section) The performance of the converted SNN model is generally less than the parent ANN model due to the conversion loss. The conversion loss generally reduces with higher latency values to reach the corresponding ANN model accuracy.

Direct Training: It is the most popular method for training SNNs from scratch based on back-propagation through time (BPTT) (Werbos 1990) that treats SNN as a recurrent neural network. However, the SNNs provide two unique challenges for the BPTT algorithm. The first is due to the Heaviside function responsible for generating spikes, which has zero gradient almost everywhere, except at transition, where it is non-differentiable. Secondly, as the chain length for back-propagation increases proportionally with the training latency, the vanishing gradient problem sets in. The STBP (Wu et al. 2018) method, based on BPTT, solves the first challenge with the help of surrogate functions. The surrogate method introduces a differentiable function (e.g., sigmoid) as an approximation of the Heaviside, which replaces the Dirac function during back-propagation. The batch normalization (BN) technique (Krizhevsky, Hinton et al. 2009) partially solves the vanishing gradient problem and enables training of deep networks through smoothing of the loss landscape (Santurkar et al. 2018). The tdBN method (Zheng et al. 2021) introduces a threshold-dependent batch normalization that harnesses this technique by including the temporal dimensions under normalization. In contrast, the BNTT (Kim and Panda 2021) method performs BN separately for the entire time dimension. The TEBN method (Duan et al. 2022) introduces a trade-off between the two by reducing the number of trainable parameters.

Though direct training can perform well for specific latencies, they are not supposed to generalize well for a latency range. Due to their fine-tuning to a particular latency, the generalization performance suffers. Moreover, the training time of the method increasing proportionally with latency poses a significant challenge.

Preliminaries

The LIF (Leaky Integrate and Fire) neurons used in SNN are governed by differential equations which, after discretization, can be represented by a recurrent relation as given in Eqn. 1. The membrane potential of neuron i at the layer l at the time step t is represented by $u_i^{(l)}[t]$, while $x_i^{(l)}[t]$ denotes the corresponding output spike which is generated by the Heaviside function h whenever the membrane potential exceeds a fixed membrane threshold u_{th}. The membrane potential recursively depends upon its value at a previous time-step scaled by the constant $\beta < 1$ and the weighted input spikes accumulated from the previous layer neurons. Following a spike, the membrane potential is either reduced by the magnitude of u_{th} called soft rest or reset to zero, known as hard reset. In this work, we follow the latter:

$$u_i^{(l)}[t] = \beta u_i^{(l)}[t-1] + \sum_j w_{ij} x_j^{(l-1)}[t] \tag{1}$$

$$x_i^{(l)}[t] = h(u_i^{(l)}[t] - u_{th}) = \begin{cases} 1 & \text{if } u_i^{(l)}[t] > u_{th} \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

$$u_i^{(l)}[t] = u_i^{(l)}[t](1 - x_i^{(l)}[t]) \tag{3}$$

With slight abuse of notation, denote input to the SNN model as $x_i^{(0)}[t]$ that represents any real value, while for layers $l > 0$, we have $x_i^{(l)}[t] \in \{0, 1\}$. This work considers LIF neurons for which the constants β and u_{th} are set to 0.5 and 1.0, respectively, following the convention in (Deng et al. 2022) for comparison. In contrast, an IF neuron used in ANN-to-SNN conversion retains the full membrane potential from the last time-step by having $\beta = 1$.

Proposed Method

The main question that we consider in this paper is how to train an SNN for some fixed latency, say T, but in such a way that it generalizes well for all the lower latencies $t \in \{1, \ldots, T\}$? First, let us recall that a standard way to make predictions in SNNs is through averaging input membrane potentials in the output layer, i.e., the output neurons do not spike but instead record the input membrane potentials over time. More precisely, at each time step, we record the pre-synaptic input in this layer as $O[t] \in \mathbb{R}^{\mathcal{Y}}$, and the predicted class corresponds to $\arg\max_{y \in \mathcal{Y}} \frac{1}{T} \sum_{t=1}^{T} O[t]$, with $\mathcal{Y}$ being the set of output labels. We adopt this expression as it is prevailing in the recent SNN literature and also referred to as standard direct training (SDT) (Zheng et al. 2021; Deng et al. 2022),

$$\mathcal{L}_{\text{SDT}}(T) = \mathcal{L}_{\text{CE}}(\frac{1}{T} \sum_{t=1}^{T} O(t), y), \tag{4}$$

where $\mathcal{L}_{\text{CE}}$ is the standard cross-entropy loss applied to the input arguments after normalizing via the soft-max function. Another popular method to compute the loss that has shown superior performance comes from (Deng et al. 2022), which first computes the cross-entropy loss with respect to $O(t)$ at each time step and then takes the average. We refer to it as TET:

$$\mathcal{L}_{\text{TET}}(T) = \frac{1}{T} \sum_{t=1}^{T} \mathcal{L}_{\text{CE}}(O(t), y) \tag{5}$$

Observe membrane potential of the LIF neuron requires a few time steps to stabilize after initialization, as can be seen in Figure 1 where the distribution of membrane potential of a neuron is found to evolve with time (Mukhoty et al. 2023). As the loss in SDT is computed with respect to the average membrane potential obtained after the last time step, the weights trained to work with a stable membrane potential do not work well with membrane potential just after initialization. Thus, the weights trained for fixed T in SDT do not work well with a lower $t < T$ used during inference.

TET intends to address this problem by first computing the loss at each time step and then taking the average of all time steps. However, the TET training is still biased towards the fixed latency T, as the average loss has components from the entire latency range. To see it from another point of view, consider the gradients of the loss with respect to the weights of the penultimate layer, l.

$$\frac{\partial \mathcal{L}_{\text{TET}}(T)}{\partial W} = \frac{1}{T} \sum_{t=1}^{T} \frac{\partial \mathcal{L}_{\text{CE}}(O(t), y)}{\partial O(t)} \cdot \frac{\partial O(t)}{\partial u^l(t)} \cdot \frac{\partial u^l(t)}{\partial W} \tag{6}$$

The gradient $\frac{\partial \mathcal{L}_{\text{TET}}(T)}{\partial w_{ij}}$ of the parameter w_{ij} with respect to loss $\mathcal{L}_{\text{TET}}(T)$ can be given as,

$$\frac{\partial \mathcal{L}_{\text{TET}}(T)}{\partial w_{ij}} = \frac{1}{T} \sum_{t=1}^{T} \frac{\partial \mathcal{L}_{\text{CE}}(O(t), y)}{\partial O(t)} \cdot \frac{\partial O(t)}{\partial u_i^l(t)} \cdot \frac{\partial u_i^l(t)}{\partial w_{ij}} \tag{7}$$

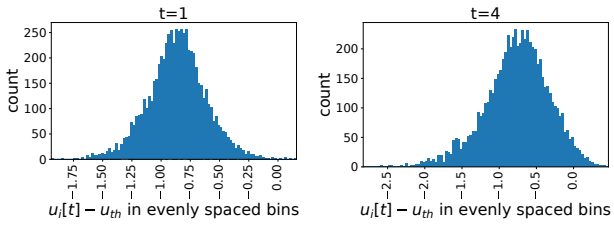

Figure 1: Fixing a neuron, we plot the distribution of membrane potential for different inputs at time steps t=1 and t=4.

Using the recurrence in Equation 1, the term $\frac{\partial u_i^l(t)}{\partial w_{ij}}$, can be expanded as :

$$\frac{\partial u_i^l(t)}{\partial w_{ij}} = \sum_{k=0}^{t} \beta^{t-k} x_j^{(l-1)}[k] \tag{8}$$

If we analyse the Equation 7 and 8, the gradient $\frac{\partial \mathcal{L}_{\text{TET}}(T)}{\partial w_{ij}}$ can be seen as a weighted sum of the gradients $\frac{\partial \mathcal{L}_{\text{TET}}(T)}{\partial O(t)} \frac{\partial O(t)}{\partial u_i^l(t)}$ computed at different latencies, where, the term $\frac{\partial u_i^l(t)}{\partial w_{ij}}$ is treated as weight. Heuristically, the weight for a higher latency is probably greater than the weight for a lower latency, as during a longer time, more spikes will be collected in (8). The resulting implicitly weighted gradients cause the model parameter to be biased towards losses at higher latencies.

The SLT Method

To improve the performance of the model at a lower latency, the gradient update needs to be unbiased towards losses computed at lower latency. An unbiased gradient update can be achieved by suppressing or nullifying the gradient from the loss at higher latencies. We propose a simple but efficient algorithm (see Algorithm 1) to find a balance between nullifying the gradient from the loss at higher latency without affecting the performance. At every batch, we sample the latency t from the set $\{1, 2, \cdots, T\}$, compute the loss, and then update the model parameters. It ensures when a lower latency value is sampled, the influence of loss from a higher latency is entirely void.

Let us denote a particular loss function $\mathcal{L}(t)$ computed over a batch with a maximum latency t. Let us denote the vector of losses as: $\hat{\mathcal{L}} = [\mathcal{L}(1), \mathcal{L}(2) \cdots, \mathcal{L}(T)] \in \mathbb{R}^T$. Given a batch, the SLT method randomly chooses a latency and evaluates the loss:

$$\mathcal{L}_{\text{SLT}} = \langle \hat{\mathcal{L}}, \mathbf{e}_t \rangle \quad t \sim [1, 2, .., T] \tag{9}$$

where, $\mathbf{e}_t \in \{0, 1\}^T$ denotes the standard basis vectors of $\mathbb{R}^T$, which determines the loss corresponding to the chosen latency. If the sampled latency for a gradient update is t, the loss $\mathcal{L}_{\text{SLT}}$ simplifies to $\mathcal{L}(t)$. Thus, on expectation over the randomness in choosing $\mathbf{e}_t$, SLT minimizes the following loss:

$$\mathbb{E}_t[\mathcal{L}_{\text{SLT}}] = \mathbb{E}_t[\langle \hat{\mathcal{L}}, \mathbf{e}_t \rangle] = \frac{1}{T} \sum_{t=1}^{T} \mathcal{L}(t) \tag{10}$$

Now, let us use the TET loss, $\mathcal{L}_{\text{TET}}$, as the particular loss function used in the SLT method to perform the gradient updates. So that,

$$\mathbb{E}_t[\mathcal{L}_{\text{SLT}}^{\text{TET}}] = \frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_{\text{TET}}(t) = \frac{1}{T}\sum_{t=1}^{T}\frac{1}{t}\sum_{s=1}^{t}\mathcal{L}_{\text{CE}}(O(s), y) = \sum_{t=1}^{T} c(t)\mathcal{L}_{\text{CE}}(O(t), y) \tag{11}$$

where, after collecting the terms, we have,

$$c(t) = \frac{1}{T}\sum_{i=t}^{T}\frac{1}{i}. \tag{12}$$

The last expression is the weight assigned on expectation over the loss at each time step. A closer look at the expression $c(t)$ will reveal that weights are much higher for lower latencies than uniform weights assigned by the TET loss. Next, we may consider applying SLT method on the SDT loss:

$$\mathbb{E}_t[\mathcal{L}_{\text{SLT}}^{\text{SDT}}] = \frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_{\text{SDT}}(t) = \frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_{\text{CE}}(\frac{1}{t}\sum_{s=1}^{t}O(s), y) \tag{13}$$

Lemma 1. *SLT-TET loss upper bounds the SLT-SDT loss on expectation.*

Proof. From Lemma 4.1 (Deng et al. 2022), we have, for all t, $\mathcal{L}_{\text{TET}}(t) \geq \mathcal{L}_{\text{SDT}}(t)$, hence,

$$\mathbb{E}[\mathcal{L}_{\text{SLT}}^{\text{TET}}] = \frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_{\text{TET}}(t) \geq \frac{1}{T}\sum_{t=1}^{T}\mathcal{L}_{\text{SDT}}(t) = \mathbb{E}_t[\mathcal{L}_{\text{SLT}}^{\text{SDT}}] \tag{14}$$

□

We implement the $\mathcal{L}_{\text{SLT}}$ loss using the procedure described in Algorithm 1, where for each batch, we sample a latency value, t, from a predefined range $\{1, 2, \cdots, T\}$. We process the batch using the SNN model with latency t, compute the cross-entropy loss using the predicted values and the true class labels, and utilize the surrogate gradients method to back-propagate the loss and update the model parameters.

Algorithm 1: SLT: Stochastic Latency Training

Require: maximum latency T, training dataset (X, Y)

$m \leftarrow$ number of training examples
for $i = 1$ to num_epochs **do**
  for $j = 1$ to $\frac{m}{\text{batch\_size}}$ **do**
    $X_{batch}, Y_{batch} \leftarrow$ get_batch$(X, Y, \text{batch_size}, j)$
    $t \sim \{1, 2, \cdots, T\}$ {sampling latency t uniformly}
    $y_{pred} = \text{SNN}(X_{batch}, t)$ {predictions are made using SNN model with latency t}
    $L = loss(y_{pred}, Y_{batch})$
    $\theta \leftarrow step(\theta, L)$
  end for
end for
return θ

	CIFAR-10 (100)	DVS-CIFAR-10
Input Neurons	32×32	48×48
No. of classes	10(100)	10
No. of epochs	300	300
Mini batch size	256	64
LIF: β	0.5	0.5
LIF: u_0	0	0
LIF: u_{th}	1	1
λ_{TET}	0.05	0.0001
Optimiser	Adam	Adam
Learning Rate	0.01	0.001
Adam: Betas	(0.9; 0.999)	(0.9; 0.999)
Rate Scheduler	CosineAnn.	CosineAnn.

Table 1: Hyper-parameter settings for comparison

Experiments

The generalization performance of our method is presented in two parts. First, we demonstrate the generalization performance of the SLT method by combining it with two standard techniques, namely, tdBN (Zheng et al. 2021) and TET (Deng et al. 2022), which we refer to as SLT-tdBN and SLT-TET, respectively. For comparison, following the recent trend (Deng et al. 2022), we chose Resnet19 (Zheng et al. 2021) architecture for the datasets CIFAR-10 and CIFAR-100, and VGGSNN (Deng et al. 2022) architecture for DVS-CIFAR10 and DVS-Gesture dataset. For the first part, instead of training the baselines, we take the accuracies as reported in the respective publications. Secondly, we evaluate the accuracy of the methods on a range of latencies, where both SLT and the baselines are trained on a fixed latency.

Datasets

CIFAR CIFAR-10 (Krizhevsky, Hinton et al. 2009) is a standard dataset with static images of 10 classes, each represented by 5000 train images and 1000 test images. CIFAR-100 is a more challenging dataset with 100 classes with 500 train images and 100 test images per class (Krizhevsky, Hinton et al. 2009). They are supplied to the SNN network using constant encoding with cutout augmentation for the fair comparison with TET (Deng et al. 2022). For training using SLT, the maximum latency was set to T=8, while the model is evaluated at T=6, T=4, and T=2. For SLT-TET method, we use the regularization parameter $\lambda = 0.05$, as prescribed in TET. The learning rate and batch size are set to 0.01 and 256, respectively, for both datasets. Table 1 provides the detailed hyper-parameter settings for training different datasets.

DVS-CIFAR-10 (Li et al. 2017) is a neuromorphic dataset consisting of events that record the change in pixel intensity, with time-stamp and location. One thousand images from each class of CIFAR-10 are converted to event-based temporal data that records pixel locations for intensity change with

Dataset	Methods	Architecture	Simulation Length	Accuracy
CIFAR10	A2S-RMP (Han, Srinivasan, and Roy 2020)	ResNet20	2048	91.36
	A2S-Opt (Deng and Gu 2021)	ResNet20	128	93.56
	A2S-QCFS (Bu et al. 2021)	ResNet20	64	92.35
	Hybrid training (Rathi et al. 2020)	ResNet-20	250	92.22
	Diet-SNN (Rathi and Roy 2020)	ResNet-20	10	92.54
	STBP (Wu et al. 2018)	CIFARNet	12	89.83
	STBP NeuNorm (Wu et al. 2019)	CIFARNet	12	90.53
	TSSL-BP (Zhang and Li 2020)	CIFARNet	5	91.41
	GLIF (Yao et al. 2022)	ResNet19	4	94.85
	DSpike (Li et al. 2021)	ResNet18	4	93.66
	tdBN (Zheng et al. 2021)	ResNet-19	6	93.16
			4	92.92
			2	92.34
	SLT-tdBN	ResNet-19	6	94.66
			4	94.44
			2	94.31
	TET (Deng et al. 2022)	ResNet-19	6	94.50
			4	94.44
			2	94.16
	SLT-TET	ResNet-19	6	**95.26**
			4	**95.18**
			2	**94.96**
CIFAR100	A2S-RMP (Han, Srinivasan, and Roy 2020)	ResNet20	2048	67.82
	A2S-Opt (Deng and Gu 2021)	ResNet20	512	72.34
	A2S-QCFS (Bu et al. 2021)	ResNet20	128	70.55
	Diet-SNN (Rathi and Roy 2020)	ResNet-20	5	64.07
	DSpike (Li et al. 2021)	ResNet18	4	73.35
	tdBN (Zheng et al. 2021)	ResNet-19	6	71.12
			4	70.86
			2	69.41
	SLT-tdBN	ResNet-19	6	71.34
			4	70.98
			2	68.87
	TET (Deng et al. 2022)	ResNet-19	6	74.72
			4	74.47
			2	72.87
	SLT-TET	ResNet-19	6	**74.87**
			4	**75.01**
			2	**73.77**
DVS-CIFAR10	tdBN (Zheng et al. 2021)	ResNet-19	10	67.8
	Streaming Rollout (Kugele et al. 2020)	DenseNet	10	66.8
	Conv3D (Wu et al. 2021)	LIAF-Net	10	71.70
	LIAF (Wu et al. 2021)	LIAF-Net	10	70.40
	GLIF (Yao et al. 2022)	7B-WideNet	16	78.10
	DSpike (Li et al. 2021)	ResNet18	10	75.90
	TET (Deng et al. 2022)	VGGSNN	10	**81.56**
	SLT-tdBN	VGGSNN	10	**79.27**
	SLT-TET	VGGSNN	10	**81.46**

Table 2: Comparison of existing results with the proposed SLT method shows improvement in generalization performance. A2S denotes ANN2SNN conversion methods.

the help of a dynamic vision sensor. These events are collated into several frames to feed the data to a neural network, where the maximum training latency determines the number of frames. For the direct training methods considered here, the number of frames for DVS datasets is set to 10, and each frame is resized from (128×128) to (48×48). Under data augmentation, the frames are further processed with transforms such as Random Horizontal Flip and Random Crop. For DVS-CIFAR-10, the exact pre-processing step used in the TET method is obscure, so we re-compute the results with data augmentation as per (Duan et al. 2022).

DVS-Gesture (Amir et al. 2017) is another neuromorphic dataset for gesture recognition with eleven classes with a pre-processing step similar to DVS-CIFAR-10.

Standard Generalization Performance

Table 2 summarises the results on CIFAR-10, CIFAR-100, and DVS-CIFAR-10 datasets. For the CIFAR-10 dataset at $T = 2$, our approach, when combined with the tdBN method, gives 1.97% improvement in the test accuracy. For both $T = 4, 6$ SLT-tdBN shows a similar improvement in the test accuracy compared to tdBN. Similarly, when our approach is combined with TET, the SLT-TET method consistently improves the test accuracy across all the latencies. For the CIFAR-100 dataset, the SLT-TET method shows a considerable improvement compared to the TET method. For the DVS-CIFAR-10 dataset, SLT-tdBN showed a 8% jump in performance compared to tdBN. This demonstrates the effectiveness of the proposed approach and its impact on neuromorphic datasets. For DVS-CIFAR-10 dataset, Table 2 reports all the accuracies from the respective publications, except for the TET method, where we report results obtained by ourselves due to difficulty in reproduction.

Generalization Across Latency Range

To compare test accuracies on a range of latencies, we train the TET and tdBN methods on fixed T and measure their performance on a latency range $[1 : 16]$. Table 3 shows such a comparison on a shorter latency range $[1 : 10]$. The SLT models used in Table3 are taken directly from Table 2, so that CIFAR datasets have the maximum training latency $T = 8$, while for DVS datasets it is $T = 10$.

Comparing tdBN and SLT-tdBN at $T = 1$, we find 1.85%, 8.24%, 28.12% and 14.07% increase in accuracy for CIFAR-10, CIFAR-100, DVS-CIFAR-10 and DVS-Gesture respectively. Comparison of other latencies also shows a similar improvement in accuracy for SLT-tdBN over tdBN for all the datasets. Comparing SLT-TET and TET, we also find improvement across most latencies and datasets. For CIFAR-10, SLT-TET provides better accuracies irrespective of the latency chosen for inference. For DVS-Gesture, we can find that the performance of SLT-TET is much better than TET for all latency except at $T = 1$.

Figure 2 shows a visual comparison of accuracy with the baselines for the complete latency range for CIFAR-10, CIFAR-100 and DVS-CIFAR-10 datasets. It is evident that, despite being trained on fixed maximum latency, SLT-TET generalizes well across the latency range, and the result mostly holds across the datasets.

	CIFAR-10					
	1	2	4	6	**8**	10
TET	93.99	94.58	95	94.96	95.02	94.98
SLT-TET	**94.53**	**94.96**	**95.18**	**95.26**	**95.3**	**95.29**
tdBN	91.5	93.49	94.33	94.52	94.49	94.6
SLT-tdBN	**93.35**	**94.31**	**94.44**	**94.66**	**94.66**	**94.75**
	CIFAR-100					
	1	2	4	6	**8**	10
TET	70.08	72.79	74.41	74.78	75	75.03
SLT-TET	**71**	**73.77**	**75.01**	**74.87**	74.99	75.01
tdBN	56.65	64.79	67.79	68.69	69.09	69.4
SLT-tdBN	**64.89**	**68.87**	**70.98**	**71.34**	**71.72**	**71.81**
	DVS-CIFAR10					
	1	2	4	6	8	**10**
TET	69.17	74.58	**79.38**	79.90	**80.94**	**81.56**
SLT-TET	**70.73**	**76.98**	78.96	**80.31**	80.73	81.46
tdBN	36.56	58.75	72.81	76.04	76.35	77.60
SLT-tdBN	**64.69**	**72.71**	**77.19**	**78.54**	**79.69**	**79.27**
	DVS-Gesture					
	1	2	4	6	8	**10**
TET	**88.60**	93.35	95.70	96.87	97.26	98.43
SLT-TET	87.50	**94.14**	**96.87**	**97.65**	**98.04**	98.43
tdBN	64.06	77.34	90.63	91.01	92.18	93.75
SLT-tdBN	**78.13**	**88.67**	**94.92**	**94.53**	**94.92**	**96.09**

Table 3: Evaluating model accuracy at different inference latencies while trained on a fixed maximum latency(CIFAR: T=8, DVS: T=10) demonstrates that SLT outperforms its competitors for most inference latencies.

Efficiency in Training Time

Figure 3 compares the time taken by each epoch of the SLT method with their baseline. After 100 epochs, the average time per epoch for tdBN is 139.7 sec., while 82.9 sec for SLT-tdBN. Thus, SLT-tdBN provides a 1.69 times speed up by choosing a shorter inference latency on expectation. Similarly, the respective averages for TET and SLT-TET are at 137.4 sec. and 82.3 sec., which amounts to 1.67 times speedup in training time.

Analysis of Loss Landscape

We compare the 1D loss landscape (Li et al. 2018) of SLT models with their counterparts. The plots are generated using the loss function of the respective model at the respective approximate saddle points as found at the end of the training process. Our objective here is to provide an explanation for the superior generalization capability of the SLT models. We consider the models from Table 2 for the CIFAR-10 dataset, trained with maximum latency $T = 8$.

Figure 4(a) illustrates the loss landscapes of the models trained using the tdBN and the corresponding SLT-tdBN method. The loss function employed to generate the loss surfaces in Figure 4(a) is given earlier in Equation 4. We plotted two 1D loss surfaces for each method: one with the original latency value of $T = 8$ and another with a reduced latency of $T = 1$. Our observations indicate that the loss curves of the proposed method exhibit a flatter profile compared to the direct loss method, thereby indicating the superior generalization capability of our proposed approach in contrast to

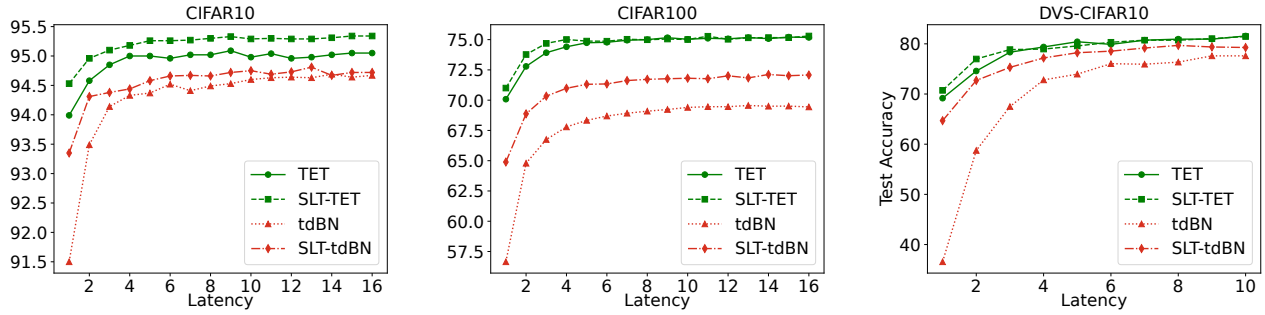

Figure 2: Comparison of the test accuracies across the latency range with models trained on maximum latency $T = 8$ for CIFAR and $T = 10$ for DVS-CIFAR-10. SLT-TET shown to outperform TET for CIFAR-10, while it is comparable or better for CIFAR-100 and DVS-CIFAR-10. SLT-tdBN outperforms tdBN at all the latencies in all three datasets.

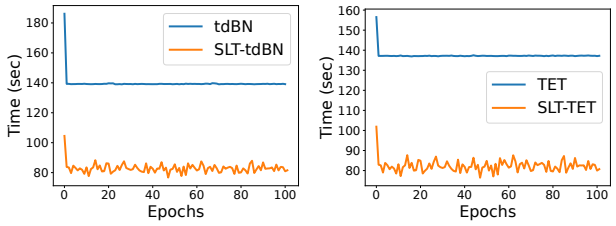

Figure 3: We plot the time taken per epoch of training by tdBN and SLT-tdBN for 100 epochs on the CIFAR-10 dataset. The average for tdBN is 139.7 sec, while for SLT-tdBN, it is 82.9 sec, equivalent to 1.69 times speed up in training time. Similarly, we compare TET with SLT-TET, where the respective averages are 137.4 and 82.3 sec, giving us a 1.67 times speedup.

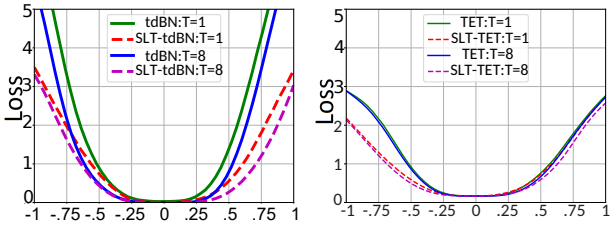

Figure 4: For the CIFAR-10 dataset, models trained using the SLT method exhibit flatter 1D loss landscapes compared to their respective baselines.

the model trained without the SLT method. Further, Figure 4(b) depicts the loss landscapes of the models trained using the TET and SLT-TET methods. We employed the TET loss function for both models. Notably, our approach SLT-TET yielded flatter minima compared to the model trained with the TET method, resulting in improved generalization capability of the proposed approach.

Estimation of Energy Requirements

To estimate the energy efficiency of our models, we followed the methodology discussed in (Rathi and Roy 2020) to compute synaptic operations (SOP) and use the energy value 77fJ/SOP reported in (Qiao et al. 2015). For the neuromorphic dataset DVS-Gesture, during inference at lower latencies, $T = 1, 2, 3, 4$, the SLT-TET model requires 0.03mJ, 0.07mJ, 0.12mJ, 0.17mJ energy respectively. Interestingly, the energy required for TET is the same as SLT-TET. Thus, the proposed method not only reduces the training time but also maintains a similar energy requirement during inference with similar or better test accuracy. To compare with tdBN, which has test accuracy of 93.75 at $T = 10$ and requires 0.34mJ per inference, SLT-tDBN achieves a similar accuracy of 94.92 at latency $T = 4$, which only requires 0.16mJ of energy.

Conclusion

We propose an efficient training method for direct training of SNNs. The proposed algorithm considers gradient updates that are not tailored to any specific inference latency. Though the state-of-the-art methods achieve good generalization performance at the target latency, they show a drop in accuracy when the latency is switched to lower values. Through experiments, we demonstrate that the proposed method is flexible in inference latency and offers the least drop in accuracy. We also demonstrate the efficiency of our algorithm in terms of training time, offering a significant speed-up, along with the fact that the stochasticity in latency brings the model to flatter minima.

Acknowledgements

This work is part of the research project "ENERGY-BASED PROBING FOR SPIKING NEURAL NETWORKS" performed at Mohamed bin Zayed University of Artificial Intelligence (MBZUAI), in collaboration with Technology Innovation Institute (TII) (Contract No. TII/ARRC/2073/2021).

References

Amir, A.; Taba, B.; Berg, D.; Melano, T.; McKinstry, J.; Di Nolfo, C.; Nayak, T.; Andreopoulos, A.; Garreau, G.; Mendoza, M.; et al. 2017. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7243–7252.

Bu, T.; Fang, W.; Ding, J.; Dai, P.; Yu, Z.; and Huang, T. 2021. Optimal ANN-SNN conversion for high-accuracy and ultra-low-latency spiking neural networks. In *International Conference on Learning Representations*.

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

Davies, M.; Srinivasa, N.; Lin, T.-H.; Chinya, G.; Cao, Y.; Choday, S. H.; Dimou, G.; Joshi, P.; Imam, N.; Jain, S.; et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1): 82–99.

DeBole, M. V.; Taba, B.; Amir, A.; Akopyan, F.; Andreopoulos, A.; Risk, W. P.; Kusnitz, J.; Otero, C. O.; Nayak, T. K.; Appuswamy, R.; et al. 2019. TrueNorth: Accelerating from zero to 64 million neurons in 10 years. *Computer*, 52(5): 20–29.

Deng, S.; and Gu, S. 2021. Optimal conversion of conventional artificial neural networks to spiking neural networks. *International Conference on Learning Representations*.

Deng, S.; Li, Y.; Zhang, S.; and Gu, S. 2022. Temporal efficient training of spiking neural network via gradient reweighting. *arXiv preprint arXiv:2202.11946*.

Diehl, P. U.; Neil, D.; Binas, J.; Cook, M.; Liu, S.-C.; and Pfeiffer, M. 2015. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, 1–8. ieee.

Duan, C.; Ding, J.; Chen, S.; Yu, Z.; and Huang, T. 2022. Temporal effective batch normalization in spiking neural networks. *Advances in Neural Information Processing Systems*, 35: 34377–34390.

Furber, S. B.; Galluppi, F.; Temple, S.; and Plana, L. A. 2014. The spinnaker project. *Proceedings of the IEEE*, 102(5): 652–665.

Gutierrez-Galan, D.; Dominguez-Morales, J. P.; Perez-Peña, F.; Jimenez-Fernandez, A.; and Linares-Barranco, A. 2020. NeuroPod: a real-time neuromorphic spiking CPG applied to robotics. *Neurocomputing*, 381: 10–19.

Han, B.; Srinivasan, G.; and Roy, K. 2020. 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*, 13558–13567.

Kim, Y.; and Panda, P. 2021. Revisiting batch normalization for training low-latency deep spiking neural networks from scratch. *Frontiers in neuroscience*, 1638.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images.

Kugele, A.; Pfeil, T.; Pfeiffer, M.; and Chicca, E. 2020. Efficient processing of spatio-temporal data streams with spiking neural networks. *Frontiers in Neuroscience*, 14: 439.

Li, H.; Liu, H.; Ji, X.; Li, G.; and Shi, L. 2017. Cifar10-dvs: an event-stream dataset for object classification. *Frontiers in neuroscience*, 11: 309.

Li, H.; Xu, Z.; Taylor, G.; Studer, C.; and Goldstein, T. 2018. Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31.

Li, Y.; Guo, Y.; Zhang, S.; Deng, S.; Hai, Y.; and Gu, S. 2021. Differentiable spike: Rethinking gradient-descent for training spiking neural networks. *Advances in Neural Information Processing Systems*, 34: 23426–23439.

Mukhoty, B.; Bojkovic, V.; de Vazelhes, W.; Zhao, X.; Masi, G. D.; Xiong, H.; and Gu, B. 2023. Direct Training of SNN using Local Zeroth Order Method. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Qiao, N.; Mostafa, H.; Corradi, F.; Osswald, M.; Stefanini, F.; Sumislawska, D.; and Indiveri, G. 2015. A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses. *Frontiers in neuroscience*, 9: 141.

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

Rathi, N.; Srinivasan, G.; Panda, P.; and Roy, K. 2020. Enabling deep spiking neural networks with hybrid conversion and spike timing dependent backpropagation. *arXiv preprint arXiv:2005.01807*.

Rueckauer, B.; Lungu, I.-A.; Hu, Y.; Pfeiffer, M.; and Liu, S.-C. 2017. Conversion of continuous-valued deep networks to efficient event-driven networks for image classification. *Frontiers in neuroscience*, 11: 682.

Santurkar, S.; Tsipras, D.; Ilyas, A.; and Madry, A. 2018. How does batch normalization help optimization? *Advances in neural information processing systems*, 31.

Strohmer, B.; Manoonpong, P.; and Larsen, L. B. 2020. Flexible spiking cpgs for online manipulation during hexapod walking. *Frontiers in neurorobotics*, 14: 41.

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

Wu, Y.; Deng, L.; Li, G.; Zhu, J.; and Shi, L. 2018. Spatio-temporal backpropagation for training high-performance spiking neural networks. *Frontiers in neuroscience*, 12: 331.

Wu, Y.; Deng, L.; Li, G.; Zhu, J.; Xie, Y.; and Shi, L. 2019. Direct training for spiking neural networks: Faster, larger, better. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, 1311–1318.

Wu, Z.; Zhang, H.; Lin, Y.; Li, G.; Wang, M.; and Tang, Y. 2021. Liaf-net: Leaky integrate and analog fire network for lightweight and efficient spatiotemporal information processing. *IEEE Transactions on Neural Networks and Learning Systems*, 33(11): 6249–6262.

Yamazaki, K.; Vo-Ho, V.-K.; Bulsara, D.; and Le, N. 2022. Spiking neural networks and their applications: A Review. *Brain Sciences*, 12(7): 863.

Yao, X.; Li, F.; Mo, Z.; and Cheng, J. 2022. Glif: A unified gated leaky integrate-and-fire neuron for spiking neural networks. *Advances in Neural Information Processing Systems*, 35: 32160–32171.

Zhang, W.; and Li, P. 2020. Temporal spike sequence learning via backpropagation for deep spiking neural networks. *Advances in Neural Information Processing Systems*, 33: 12022–12033.

Zheng, H.; Wu, Y.; Deng, L.; Hu, Y.; and Li, G. 2021. Going deeper with directly-trained larger spiking neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 11062–11070.