DALTON - Deep Local Learning in SNNs via local Weights and Surrogate-Derivative Transfer

Ramashish Gaurav[†*], Duy Anh Do[†], Thinh Doan[†], Yang Yi[†]

[†] Department of Electrical and Computer Engineering, Virginia Tech, USA
[*] Corresponding Author: rgaurav@vt.edu

***Abstract*—Direct training of Spiking Neural Networks (SNNs) is a challenging task because of their inherent temporality. Added to it, the vanilla Back-propagation based methods are not applicable either, due to the non-differentiability of the spikes in SNNs. Surrogate-Derivative based methods with Back-propagation Through Time (BPTT) address these direct training challenges quite well; however, such methods are not neuromorphic-hardware friendly for the On-chip training of SNNs. Recently formalized Three-Factor based Rules (TFR) for direct local-training of SNNs are neuromorphic-hardware friendly; however, they do not effectively leverage the depth of the SNN architectures (we show it empirically here), thus, are limited. In this work, we present an *improved version* of a conventional three-factor rule, for local learning in SNNs which effectively leverages depth – in the context of learning features hierarchically. Taking inspiration from the Back-propagation algorithm, we theoretically derive our improved, local, three-factor based learning method, named DALTON (Deep LocAl Learning via local WeighTs and SurrOgate-Derivative TraNsfer), which employs *weights* and *surrogate-derivative* transfer from the local layers. Along the lines of TFR, our proposed method DALTON is also amenable to the neuromorphic-hardware implementation. Through extensive experiments on static (MNIST, FMNIST, & CIFAR10) and event-based (N-MNIST, DVS128-Gesture, & DVS-CIFAR10) datasets, we show that our proposed local-learning method DALTON makes *effective use of the depth* in Convolutional SNNs, compared to the vanilla TFR implementation.**

***Index Terms*—Spiking Neural Networks, local learning, surrogate derivatives, three factor rule, neuromorphic hardware.**

I. Introduction

Spiking Neural Networks (SNNs) are gradually gaining attention from the AI community for a variety of applications [1]–[4], with a significant corpus of work on Spiking-CNNs (or Convolutional-SNNs) in computer vision [5]–[9]. This is primarily because of their promising energy efficiency on the neuromorphic computing hardware [5], [10]–[14], as well as due to their closeness to biological plausibility [15]–[17]. Unlike Artificial Neural Networks (ANNs), where the outputs of the neurons are continuous real values, the neurons in the SNNs transmit spikes in time. Building SNNs is not a trivial task, only a limited number of methods exist. One such popular method is ANN-2-SNN conversion [5], [18]–[20], where a trained Deep Neural Network (DNN) is converted to a deep SNN by replacing the conventional non-spiking neurons (e.g., `ReLU` neurons) with spiking neurons (e.g., `Integrate & Fire (IF)` neurons), along with other necessary network-parameters modifications [19], [20]. Note that deep SNNs have been well capitalized to achieve SoTA results on complex datasets [8], [9], [19], hence, deep SNNs are desired. Although the ANN-2-SNN conversion methods achieve high accuracy results, the SNNs built with them suffer from high firing rates [20], [21] due to the rate approximations of the non-spiking neurons; and subsequently do not offer the best of the energy efficiency on the neuromorphic hardware. Also, the ANN-to-SNN method does not make use of the temporality of spikes in the SNNs, and results in increased latency, thereby increasing the Energy-Delay-Product [5], [19], [21]. Another popular approach is to *directly* train the SNNs (instead of conversion), which requires researchers to address the *non-differentiability* of the discrete spikes (often modeled as the Dirac's delta function) in the SNNs. This non-differentiability of the spikes renders the native application of the workhorse Back-propagation algorithm unfeasible. However, significant strides have been made in recent years to alleviate the non-differentiability problem via the application of the *surrogate-derivatives/gradients* as *approximations* to the ill-defined derivative of the spike function; [17], [22], [23] use the derivative of modified sigmoid functions (more types of surrogate-derivatives in [24]). These growing number of works [14], [17], [22], [23], [25] based on surrogate-derivatives fall into the category of Surrogate Gradient Descent (SurrGD) method, where Back-propagation Through Time (BPTT) in conjunction with surrogate-derivatives is used to directly train the SNNs. By virtue of direct-training, SurrGD makes effective use of temporality in spikes and results in low-latency SNNs.

However, on one hand, where SurrGD addresses the direct training challenges in SNNs, it isn't neuromorphic hardware friendly due to the *global* error back-propagation and non-biological plausibility (where *local* learning is preferred). Note that the ANN-to-SNN method too - by definition, forgoes the direct training of SNNs, and is not applicable for the On-chip implementation. Three-Factor Rule (TFR) based *local* learning to directly train SNNs is gradually gaining traction due to its neuromorphic hardware amenability (e.g., *DECOLLE* [17]). In TFR-based training, the two *pre-* and *post-* synaptic factors are complemented by a third *supervisory/error* factor; DECOLLE uses *layer-wise local* random `Readout` matrices and *global* ground-truths to *locally* generate the third error-signal factor – we present more details on vanilla-TFR/DECOLLE in Sec. II. However, as we demonstrate later, vanilla-TFR/DECOLLE do *not* effectively leverage depth. To this end, we present our novel three-factor based local-learning method: "**DALTON**" – ***D**eep Loc**A**l **L**earning via local Weigh**T**s and Surr**O**gate-*

Derivative TraNsfer in this work, which better leverages depth than DECOLLE. Following are our major contributions:

- We theoretically derive our DALTON method, and it improves the vanilla-TFR/DECOLLE -based learning by:
  - Making use of the *local* layers' weights, instead of the *random* values as the `Readout` layers' weights
  - Computing the *local* truth-values - instead of using the *global* ground-truth for each `Readout` layer
- DALTON method retains the *locality* and *online learning in time* characteristics of the TFR-based learning method
- Through exhaustive experiments on three static-images datasets – MNIST, FMNIST, and CIFAR10, and three event-based images datasets – N-MNIST, DVS128-Gesture, and DVS-CIFAR10, we show that our DALTON method *effectively* uses the depth of Conv-SNNs to improve the performance over vanilla-TFR/DECOLLE

II. Related Work

We start with a brief paragraph on the existing neuromorphic hardware to support the *motivations* behind developing local-learning rules to train SNNs On-chip. Neuromorphic hardware is built on the principles of parallel information processing in brain where the *memory* and *compute* are *local* to each other; this prevents the memory-bottleneck problem that the conventional vonNeumann based systems suffer from. A few popular neuromorphic chips used for deploying SNNs are SpiNNaker2 [11] and Loihi [12], [26]. SpiNNaker2 is composed of multiple Processing Elements (PEs), where each PE comprises an ARM Cortex-M4F *core* and SRAM *memory*, apart from MAC array, buses, etc.; the processor *core* and *memory* are interconnected via buses and crossbars, thus, being local to each other [11]. Similarly, Loihi is composed of neuro-cores, where each neuro-core has $1,024$ spiking neural/computational units and 2Mb SRAM [26] – the authors strongly advocate for *locality* in learning on Loihi. Note that, with respect to SNNs, *locality in learning* can be defined as the local (to synapse/learning engine) availability of all the information needed to update the weights, for which, Loihi offers a variety of support [26]. Thus, the *architectural locality* of *memory* and *compute* in neuromorphic hardware underlies our motivation to develop improved versions of local learning rules to train SNNs. This architectural locality is also the reason why *global* error back-propagation based SurrGD is *not* neuromorphic h/w friendly, and rather, *local* learning is preferred for On-chip training of SNNs. We next discuss the existing local learning works.

Local learning in spiking networks / SNNs makes use of the brain-inspired rules, e.g., Synaptic Time Dependent Plasticity (STDP) [27], Bienenstock-Cooper-Munro (BCM) [28], and Oja [29] rules; these are conventionally unsupervised [30], [31], however, they can be modified and used for supervised learning tasks [16], [32], [33]. Local learning rules make use of the locally available information from the *pre-* and *post-*synaptic neurons to update the synaptic weights. However, such local learning methods are still in nascent stage for applications in training deep SNNs effectively. Of late, a few works [34]–[36] have explored the reward/neuro-modulated STDP to train spiking networks, wherein, the *reward* or the *neuro-modulatory* signal is considered as the *third supervisory* factor (apart from the *pre-* and *post-*synaptic spikes' times as the *two* factors); this not only improves upon the STDP but also falls in line with biologically-plausible local learning. Frémaux et al. [34] present an excellent review of the neuro-modulated STDP and the Three-Factor Rule (TFR) based learning in their work. Along the similar lines, Kaiser et al. [17] recently presented their TFR-based *DECOLLE* method to train SNNs with locally available information and surrogate-derivatives. The authors [17] define a *local loss function* for each *trainable* layer and compute the local errors on the *predicted logits* from the associated *non-trainable random* `Readout` layers, where the *local targets* (for each `Readout` layer) was set equal to the *global ground-truth* values. This local error (for each trainable layer) is the third *crucial* factor in their TFR formulation, and is back-propagated to the respective (one) trainable layer to update the network weights locally. A latest work on the lines of TFR is the Event-based Three-factor Local Plasticity (ETLP) rule by Quintana et al. [37], where the first two factors originate from the pre- and post-synaptic neurons, and the third factor is a teaching signal (which involves no local error calculation). Note that the teaching signal is generated as spikes fired at a certain frequency (e.g., $100Hz$) from the one-hot encoded targets as teaching neurons. Quintana et al. [37] evaluated ETLP on simple `Dense`-SNNs consisting of an input layer, one hidden layer, and an output layer. Since ETLP was not evaluated on the `Conv`olutional-SNNs, their efficacy for more complex deep-SNN architectures is vague.

Note that DECOLLE (based on TFR) was evaluated on the `Conv`olutional-SNNs; however, by design it is limited compared to the SurrGD approach. In the SurrGD method, the *global* error (from the final output layer) is back-propagated to the deeper layers (scaled by the intermediate weights and derivatives) which enables the SNNs to make better/explicit use of the depth to learn rich hierarchical features. DECOLLE on the other hand limits the flow of error-gradients to one trainable layer each, to achieve *locality* in accordance with TFR, and does *not* effectively leverage the depth in SNNs (we demonstrate this drawback with our experiments). DECOLLE is inspired from the Feedback Alignment (FA) approach [38] and the local-errors method [39] to avoid the weight transport problem. Bartunov et al. [40] showed that multi-layer networks trained with FA perform *inferior* to the Back-propagation ones. Mostafa et al. [39] too, found that training DNNs with local errors is *less* effective compared to Back-propagation. Therefore, in light of these limitations of DECOLLE (by the virtue of local TFR) *why even consider such an approach?* This is because TFR-based learning is particularly attractive for *energy-efficient On-chip training* of SNNs due to *their ease of implementation on the neuromorphic hardware*, note that Loihi-2 [12] supports a reward-modulated form of TFR.

We next begin with a short description of SNNs and of the vanilla-TFR/DECOLLE in Sec. III, followed by Sec. IV where we lay down the theoretical and implementation details of our proposed DALTON. We then describe our experiments in Sec. V, followed by a discussion on the results, the DALTON method, and its difference with Back-propagation and vanilla-TFR/DECOLLE in Sec. VI. We then conclude our work in Sec. VII with future directions to improve DALTON.

III. Preliminaries

In this section, we present the preliminaries of our DALTON method. We start with the basics of our SNNs, followed by the essentials of the vanilla-TFR/DECOLLE implementation [17] (we explain DECOLLE because it is similar to DALTON).

A. *Spiking Neural Networks*

Spiking Neural Networks (SNNs), as mentioned above, are composed of spiking neurons (instead of the rate-based artificial neurons), with network architectures largely similar to that of the conventional ANNs/DNNs. The spiking neurons upon sufficient excitation, generate and transmit spikes through time; spikes can be binary valued or graded, depending upon the choice of implementation. In our SNNs, we use the binary-valued `Integrate & Fire` (`IF`) spiking neurons and define the spiking function as $S(t) = \Theta(U(t) - U_{\text{thr}})$, where Θ is the Heaviside Step-function, $U(t)$ and U_{thr} are the neuron's voltage and threshold value (U_{thr} is kept the same for all the neurons in our SNNs). The `IF` neuron's continuous-time state equations (current $I(t)$ and voltage $U(t)$) are described below.

$$\frac{dI(t)}{dt} = \frac{-I(t)}{\tau_{\text{cur}}} + \sum_j W_j S_j(t) \tag{1a}$$

$$\frac{dU(t)}{dt} = \frac{\Omega I(t)}{\tau_{\text{vol}}} \qquad \text{when } U(t) < U_{thr} \tag{1b}$$

where W_j is the weight assigned to the corresponding incoming spike $S_j(t)$, Ω is the resistance of the neuron model, and τ_{cur} & τ_{vol} are the neuron's current and voltage time-constants. The voltage $U(t)$ is reset to 0 once the neuron spikes, i.e., $U(t) \leftarrow 0$ when $U(t) \geq U_{\text{thr}}$. We define the discrete-time state equations: $I[t]$ and $U[t]$ later in the Section V-B. Note that we use the term Conv-SNNs to denote the SNNs composed of `Convolutional` and fully-connected `Dense` layers (with/without the `Pooling` layers). Also, the Conv-SNNs' architectures in this work have a spiking layer following each `Convolutional`/`Dense` layer (`Pooling` if present, is applied prior to the spiking layer, i.e., after the `Conv` layer).

B. *Vanilla-TFR / DECOLLE [17]*

Kaiser et al. [17] model their DECOLLE method on TFR, where the weight update $\Delta W_l^{i,j}$ depends on *three locally* available data: a pre-synaptic j^{th} term, a post-synaptic i^{th} term, and a supervisory signal. Here we present the essentials of their equations which are useful in the context of our DALTON method (we stay true to their notations, except for the sub/super-scripts). For readers unfamiliar with DECOLLE, we highly recommend visiting Section 2.3 and Figs. 1 and 4 of [17] (or Fig. 2 of this work). In [17], the authors attach a random `Readout` layer $G_l \in \mathbb{R}^{a \times b}$ to each spiking layer (indexed by l), such that the predicted logits $Y_l^i = \sum_j G_l^{ij} S_l^j$, where S_l^j is the spike output. The MSE *local* loss $\mathcal{L}_l = \frac{1}{2}\sum_i (Y_l^i - \hat{Y}_l^i)^2$ is computed for each spiking layer l ($\hat{Y}_l^i$ is the one-hot encoded *local target*). Note that in conventional Back-propagation, a *gobal* loss $\mathcal{L}$ is computed, which is a function of *all* the weight matrices W_l; thus, the weight update $\frac{\partial \mathcal{L}}{\partial W_l}$ is defined for *every* layer l. Since in DECOLLE, a *local* loss $\mathcal{L}_l$ is computed for each layer l, they [17] ensure that $\mathcal{L}_l$ is a function of W_l *only*, and *no* other W_m $\forall$ $m \neq l$, this translates that the gradient of $\mathcal{L}_l$ exists *only* with respect to W_l. Therefore, $\forall$ the layers $m \neq l$, the authors [17] set $\frac{\partial \mathcal{L}_l}{\partial W_m^{ij}} = 0$ to enforce *locality of weight updates*. Thus, the weight updates of layer l, i.e., $\Delta W_l^{ij} = -\eta \frac{\partial \mathcal{L}_l}{\partial W_l^{ij}} = -\eta \frac{\partial \mathcal{L}_l}{\partial S_l^i} \frac{\partial S_l^i}{\partial W_l^{ij}}$ (by application of chain rule, η = learning rate). By again applying chain rule on $\frac{\partial S_l^i}{\partial W_l^{ij}}$, we get $\frac{\partial S_l^i}{\partial W_l^{ij}} = \frac{\partial \Theta(U_l^i)}{\partial U_l^i} \frac{\partial U_l^i}{\partial W_l^{ij}}$ (since $S_l^i = \Theta(U_l^i)$, where U_l^i is the neuron's voltage). Due to the non-differentiability of the spikes S_l^i, the term $\frac{\partial \Theta(U_l^i)}{\partial U_l^i}$ is approximated by $\Theta(U_l^i)$'s surrogate derivative, i.e., $\sigma'(U_l^i)$, where σ is a chosen surrogate function. Note that U_l^i is a function of a term P_l^j in [17], thus, $\frac{\partial U_l^i}{\partial W_l^{ij}} = P_l^j$ (P_l^j in turn is a function of spikes S_{l-1}^j). Therefore, $\frac{\partial S_l^i}{\partial W_l^{ij}} = \sigma'(U_l^i) P_l^j$, which implies $\Delta W_l^{ij} = -\eta \frac{\partial \mathcal{L}_l}{\partial S_l^i} \sigma'(U_l^i) P_l^j$ (Eq. (7) in [17]); where P_l^j and $\sigma'(U_l^i)$ are the two *pre*- and *post*-synaptic factors, and $\frac{\partial \mathcal{L}_l}{\partial S_l^i} = \sum_k G_l^{ki}(Y_l^k - \hat{Y}_l^k)$ is the third *error/supervisory* factor; thus, DECOLLE effectively culminates to a TFR – it depends upon *three* factors *local* to the synapse W_l^{ij}, i.e., $\frac{\partial \mathcal{L}_l}{\partial S_l^i}$, $\sigma'(U_l^i)$, and P_l^j. Note that, for their experiments with DVS128-Gesture & N-MNIST datasets, Kaiser et al. [17] set $G_l \in \mathbb{R}^{n_l \times c}$ where n_l is the number of neurons in the layer l, and c is the number of classes.

IV. Methods

In this section, we first revisit the Back-propagation theory in DNNs/CNNs which lays down the theoretical foundations of the derivation of our proposed DALTON method. We then describe the implementation details of DALTON to train SNNs in BPTT and Real Time Recurrent Learning (RTRL) settings.

A. *Revisiting Back-propagation in DNNs/CNNs*

Our primary motivation from the Back-propagation theory is to uncover the *relationship between the weight gradients* $\frac{\partial \mathcal{L}}{\partial W_l}$ *of the consecutive blocks/layers* l ($\mathcal{L}$ is the *global* loss function), and use that to improve the vanilla-TFR method. Fig. 1 shows the architecture of a typical deep CNN (non-spiking); consider it to have N blocks, of which, there are N_C `Conv` layers and $N - N_C = N_D$ `Dense` layers. The operations in a block l composed of a `Conv`/`Dense` layer followed by an activation layer (θ function) is given by:

$$X_l = W_l \boxtimes O_{l-1} \qquad \text{and} \qquad O_l = \theta(X_l), \tag{2}$$

where O_{l-1} is the input to the block l from the previous layer $l-1$, $\boxtimes$ is a generic operator applying the connection weights W_l to the O_{l-1} such that, $X_l = D_l^w O_{l-1}$ for the `Dense` layers (i.e., $W_l = D_l^w$), and $X_l = C_l^w * O_{l-1}$ for the `Conv` layers (i.e., $W_l = C_l^w$, and $*$ is the convolution operator). Here, θ is the activation function (e.g., `ReLU`), and X_l and O_l are its inputs and outputs. Note that for $l = 1$, O_{l-1} = `Input` to the network. The `Output` of the network is $Y^p = X_N$, and given the ground truth Y^t, the *global* loss $\mathcal{L}$ can be computed as $\mathcal{L} = \frac{1}{2}(Y^p - Y^t)^2$ for the MSE loss function. After computing $\frac{\partial \mathcal{L}}{\partial W_l}$ $\forall$ $l \in [1, N-1]$, we derive the weight-gradients *relation* for the consecutive layers (i.e., $\frac{\partial \mathcal{L}}{\partial W_l}$ and $\frac{\partial \mathcal{L}}{\partial W_{l+1}}$); below, we show these relations for the `Dense` and `Conv` layers (no `Pooling` after) in the Eqs. (3) and (4) respectively.

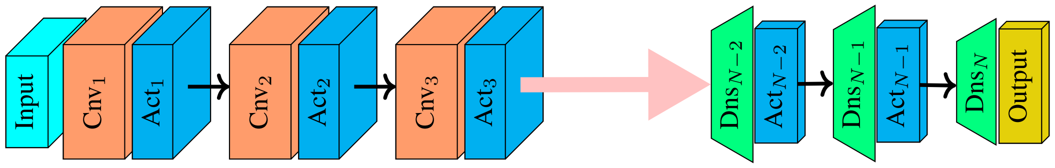


Fig. 1: *A Typical CNN's Architecture.* "`Input`" is the real-valued image input, "Cnv_l" is the l^{th} `Conv`olutional layer weights, "Dns_l" is the l^{th} layer of `Dense` weights, "Act_l" is the l^{th} layer of activation units (e.g., `ReLU` neurons) following the l^{th} `Convolutional`/`Dense` layer, and "`Output`" is the layer of no activation output units. Arrows indicate the forward propagation / flow of the activation outputs. At the `Output` layer, *global* loss $\mathcal{L}$ is computed, and propagated backwards via weight-gradients.

- For `Dense` W_l layers:

$$\frac{\partial \mathcal{L}}{\partial W_l} = \left[\left\{W_{l+1}^T\left(\frac{\partial \mathcal{L}}{\partial W_{l+1}}(O_l^T)^{-1}\right)\right\}\odot \nabla\theta(X_l)\right] O_{l-1}^T \quad (3)$$

- For `Conv` W_l layers:

$$\frac{\partial \mathcal{L}}{\partial W_l} = \left[\left\{\mathbb{W}_{l+1}\otimes\left(\frac{\partial \mathcal{L}}{\partial W_{l+1}}\otimes \mathbb{O}_l^{-1}\right)\right\}\odot \nabla\theta(X_l)\right]\otimes \mathbb{O}_{l-1} \quad (4)$$

where, $\odot$, $\otimes$, and ∇ are the Hadamard product, Tensordot, and Gradient operations respectively. For easy access in the later sections, we also mention all these symbols with their meanings in the Table I. Note that $O_.$ and $\mathbb{O}_.$ in the Eqs. (3) and (4) are *matrices* and *tensors* respectively. We provide a concise proof of the Eq. (3) below; a more complete proof for both the Eqs. (3) and (4) is in the Suppl. doc - Appx. B.

1) Proof of Eq. (3): We begin by noting the equation below that concisely represents the expression for the gradient $\frac{\partial \mathcal{L}}{\partial W_l}$:

$$\frac{\partial \mathcal{L}}{\partial W_l} = \left[\prod_{i=l+1}^{N} W_i^T\cdot(Y^p - Y^t)\cdot \bigodot_{j=N-1}^{l}\nabla\theta(X_j)\cdot\right] O_{l-1}^T \quad (5)$$

where $\prod$ and $\bigodot$ represent chained matrix multiplication and Hadamard product respectively; more on Eq. (5) in Suppl. doc - Appx. B.1. Next, Eq. (5) implies that the gradient $\frac{\partial \mathcal{L}}{\partial W_{l+1}}$ is:

$$\frac{\partial \mathcal{L}}{\partial W_{l+1}} = \left[\prod_{i=l+2}^{N} W_i^T\cdot(Y^p - Y^t)\cdot \bigodot_{j=N-1}^{l+1}\nabla\theta(X_j)\right] O_l^T \quad (6)$$

Multiplying Eq. (6) on both sides by the right inverse of O_l^T, i.e., $(O_l^T)^{-1}$ (s.t., $O_l^T(O_l^T)^{-1} = I$), we have the following:

$$\frac{\partial \mathcal{L}}{\partial W_{l+1}}(O_l^T)^{-1} = \left[\prod_{i=l+2}^{N} W_i^T\cdot(Y^p - Y^t)\cdot \bigodot_{j=N-1}^{l+1}\nabla\theta(X_j)\right] \quad (7)$$

Now that we have separated the expression in square brackets (i.e., [.]), we come back to the Eq. (5) and expand it as follows:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W_l} &= \left[\prod_{i=l+1}^{N} W_i^T\cdot(Y^p - Y^t)\cdot \bigodot_{j=N-1}^{l}\nabla\theta(X_j)\right] O_{l-1}^T\\ &= [\{W_{l+1}^T\\ &\quad \underline{\left(\prod_{i=l+2}^{N} W_i^T\cdot(Y^p - Y^t)\cdot \bigodot_{j=N-1}^{l+1}\nabla\theta(X_j)\right)}\\ &\quad \}\odot\nabla\theta(X_l)]O_{l-1}^T \quad \text{(by expanding brackets in order)}\end{aligned} \quad (8)$$

TABLE I: Mathematical symbols used and their meanings

Symbol	Meaning	Symbol	Meaning
$\boxtimes$	Generic Operator	∇	Gradient Operator
$*$	Convolution Operator	$\odot$	Hadamard Product
Θ	Spike Function	$\otimes$	Tensordot (Einsum)
Π	Matrix Product	$\equiv$	Equivalence

Note: The generic operator $\boxtimes$ can either mean matrix multiplication or $*$

Thus, by substituting the underlined expression in Eq. (8) with $\frac{\partial \mathcal{L}}{\partial W_{l+1}}(O_l^T)^{-1}$ (see Eq. (7)), we arrive the following Eq. (9):

$$\frac{\partial \mathcal{L}}{\partial W_l} = \left[\left\{W_{l+1}^T\left(\frac{\partial \mathcal{L}}{\partial W_{l+1}}(O_l^T)^{-1}\right)\right\}\odot \nabla\theta(X_l)\right] O_{l-1}^T \quad (9)$$

thereby proving the Eq. (3). The consecutive layers' weight-gradients *relation* for the `Conv` layers followed by `Pooling` layer, and its *derivation* are in the Suppl. doc - Appx. C.

B. Derivation of the DALTON method to train SNNs

The two main aspects of vanilla-TFR / DECOLLE (we use both the terms interchangeably) that we want to improve upon, is (1) the choice of the *random* `Readout` matrices G_l, and (2) the usage of the *global* ground-truths as the *local* truth-value $\hat{Y}_l$. Henceforth, we use the notations Y_l^t to denote the *local* truth-values (i.e., $Y_l^t = \hat{Y}_l$ in DECOLLE), and Y_l^p as the local predicted values (i.e., $Y_l^p = Y_l$ in DECOLLE). Similar to the DECOLLE's setup, in each block l, our Conv-SNNs have a non-trainable `Readout` layer associated with each spiking layer following the trainable `Conv`/`Dense` layer (see Fig. 2). Also note that, the series of contiguous `Conv` layers/blocks is followed by a series of contiguous `Dense` layers/blocks.

We begin by defining the neuronal dynamics of our Conv-SNNs (*isomorphic* to the considered deep-CNN operations in the Sec. IV-A), considering *one* time-step's operation:

$$U_l = W_l \boxtimes S_{l-1} \qquad \text{and} \qquad S_l = \Theta(U_l) \quad (10)$$

where U_l is the voltage of the spiking layer l's neurons, $\boxtimes$ is the operator applying the weights W_l to the incoming spikes S_{l-1}, and Θ is the Heaviside Step-function. Similar to the Section IV-A, $W_l = D_l^w$ weights for the `Dense` layers, and $W_l = C_l^w$ weights for the `Conv` layers. Note that, for a current time-step, since U_l is a function of I_l, and I_l is a function of the current time-step's S_{l-1} , we collapse $U_l = W_l \boxtimes S_{l-1}$ (Eq. (1)). Moreover, in our case, the output of each `Readout` layer (in Fig. 2) is $Y_l^p = G_l \boxtimes S_l$ (where G_l is the l^{th} `Readout` weights). We choose MSE as the *local* loss function, i.e., $\mathcal{L}_l = \frac{1}{2}(Y_l^p - Y_l^t)^2$. Note that similar to DECOLLE, we set $\frac{\partial \mathcal{L}_l}{\partial W_m^{ij}} = 0, \forall\ m \neq l$. We then calculate $\frac{\partial \mathcal{L}_l}{\partial W_l}$ for each trainable layer l:

$$\begin{aligned}\frac{\partial \mathcal{L}_l}{\partial W_l} &= \frac{1}{2}\frac{\partial (Y_l^p - Y_l^t)^2}{\partial W_l} \\ &= (Y_l^p - Y_l^t)\frac{\partial Y_l^p}{\partial W_l} \qquad \text{note: } Y_l^p = G_l \boxtimes S_l, S_l = \Theta(U_l) \\ &= (Y_l^p - Y_l^t)\left(\frac{\partial Y_l^p}{\partial S_l}\frac{\partial S_l}{\partial U_l}\frac{\partial U_l}{\partial W_l}\right) \\ &= \left[\left\{(Y_l^p - Y_l^t)\frac{\partial Y_l^p}{\partial S_l}\right\}\frac{\partial S_l}{\partial U_l}\right]\frac{\partial U_l}{\partial W_l} \quad \text{note: } U_l = W_l \boxtimes S_{l-1} \\ &= \left[\left\{\frac{\partial G_l \boxtimes S_l}{\partial S_l}(Y_l^p - Y_l^t)\right\} \odot \nabla\Theta(U_l)\right]\frac{\partial W_l \boxtimes S_{l-1}}{\partial W_l}\end{aligned} \tag{11}$$

such that, following two cases hold depending on W_l's type:

- Case 1 - `Dense` W_l, which implies $W_l \boxtimes S_{l-1} = W_l S_{l-1}$:

That is, from Eq. (11):

$$\begin{aligned}\frac{\partial \mathcal{L}_l}{\partial W_l} &= \left[\left\{\frac{\partial G_l \boxtimes S_l}{\partial S_l}(Y_l^p - Y_l^t)\right\} \odot \nabla\Theta(U_l)\right]\frac{\partial W_l S_{l-1}}{\partial W_l} \\ &= \left[\left\{\frac{\partial G_l \boxtimes S_l}{\partial S_l}(Y_l^p - Y_l^t)\right\} \odot \nabla\Theta(U_l)\right] S_{l-1}^T\end{aligned} \tag{12}$$

- Case 2 - `Conv` W_l, which implies $W_l \boxtimes S_{l-1} = W_l * S_{l-1}$:

That is, from Eq. (11):

$$\begin{aligned}\frac{\partial \mathcal{L}_l}{\partial W_l} &= \left[\left\{\frac{\partial G_l \boxtimes S_l}{\partial S_l}(Y_l^p - Y_l^t)\right\} \odot \nabla\Theta(U_l)\right]\frac{\partial W_l * S_{l-1}}{\partial W_l} \\ &= \left[\left\{\frac{\partial G_l \boxtimes S_l}{\partial S_l}(Y_l^p - Y_l^t)\right\} \odot \nabla\Theta(U_l)\right] \otimes \mathbb{S}_{l-1}\end{aligned} \tag{13}$$

Note that in Eq. (12), S_{l-1}^T $(= \frac{\partial W_l S_{l-1}}{\partial W_l})$ is a *matrix*, whereas in Eq. (13), $\mathbb{S}_{l-1}$ $(= \frac{\partial W_l * S_{l-1}}{\partial W_l})$ is a *tensor*. Also note that, we have not specified the nature of the $\boxtimes$ op. for the `Readout` layer in $G_l \boxtimes S_l$ in the Eqs. (12) and (13); we obtain that later. Since the Eqs. (3) and (4), i.e., the *related global* loss $\mathcal{L}$'s weight-gradients of the consecutive layers in a deep-CNN (obtained via Back-propagation) enable effective learning via leveraging depth, we approximate the weight-gradients of our Conv-SNN's *local* loss $\mathcal{L}_l$, i.e., the Eqs. (12) and (13) equal to the Eqs. (3) and (4) respectively, depending upon W_l; thus:

$$\frac{\partial \mathcal{L}_l}{\partial W_l} \approx \frac{\partial \mathcal{L}}{\partial W_l} \tag{14}$$

Note that, there are a few works [38], [39], [41] that align well with our approach of setting the gradients approximately equal in Eq. (14). In FA [38], the angle between the FA-based gradient and Back-propagation-based gradient *converge* below 45° in fully-connected networks during training; although [38] did not use *local* errors, this *convergence* implies that the loss gradients with respect to the "symmetric" and (*locally associated*) "random" weights eventually become similar. Next, the authors in [39] explicitly mention that training with *local* errors "retain the hierarchical composition of features" – this conceptually conveys that the *local* gradients $\frac{\partial \mathcal{L}_l}{\partial W_l}$ bear *similar* characteristics as of the *global* gradients $\frac{\partial \mathcal{L}}{\partial W_l}$ (since the weights are trained via error-gradients). Finally, the closest basis to our work is [41], where the authors introduce the concept of *Decoupled Neural Interfaces* (DNI) to train the network weights in a layer-wise *local* manner; they do so by generating *synthetic gradients* $\frac{\partial \mathcal{L}_i}{\partial h_i}$ *local* to each layer i. We note that the key equation in [41]: $\frac{\partial \mathcal{L}}{\partial \theta_i} \simeq \hat{f}_{\text{Bprop}}(h_i)\frac{\partial h_i}{\partial \theta_i}$ is quite similar to our Eq. (14), where $\hat{f}_{\text{Bprop}}$ is $\frac{\partial \mathcal{L}_i}{\partial h_i}$, and h_i and θ_i are activations and weights respectively. Thus, from Eq. (14):

- For `Dense` W_l layers, setting Eq. (12) $\approx$ Eq. (3):

$$\begin{aligned}&\left[\left\{\frac{\partial G_l \boxtimes S_l}{\partial S_l}(Y_l^p - Y_l^t)\right\} \odot \nabla\Theta(U_l)\right] S_{l-1}^T \qquad \approx \\ &\qquad \left[\left\{W_{l+1}^T\left(\frac{\partial \mathcal{L}}{\partial W_{l+1}}(O_l^T)^{-1}\right)\right\} \odot \nabla\theta(X_l)\right] O_{l-1}^T\end{aligned} \tag{15}$$

- For `Conv` W_l layers, setting Eq. (13) $\approx$ Eq. (4):

$$\begin{aligned}&\left[\left\{\frac{\partial G_l \boxtimes S_l}{\partial S_l}(Y_l^p - Y_l^t)\right\} \odot \nabla\Theta(U_l)\right] \otimes \mathbb{S}_{l-1} \qquad \approx \\ &\left[\left\{\mathbb{W}_{l+1} \otimes \left(\frac{\partial \mathcal{L}}{\partial W_{l+1}} \otimes \mathbb{O}_l^{-1}\right)\right\} \odot \nabla\theta(X_l)\right] \otimes \mathbb{O}_{l-1}\end{aligned} \tag{16}$$

Note the color matching in the Eqs. (15) and (16), which holds because the Eq. (10) and Eq. (2) are similar (i.e., $S_l \equiv O_l$ and $U_l \equiv X_l$, where $\equiv$ implies equivalence). Next, Eq. (15) implies $\frac{\partial G_l \boxtimes S_l}{\partial S_l} = W_{l+1}^T$, where $W_{l+1}^T \leftarrow \frac{\partial W_{l+1} O_l}{\partial O_l}$; and Eq. (16) implies $\frac{\partial G_l \boxtimes S_l}{\partial S_l} = \mathbb{W}_{l+1}$, where $\mathbb{W}_{l+1} \leftarrow \frac{\partial W_{l+1} * O_l}{\partial O_l}$. Thus, we deduce *two* important observations in DALTON method:

- The value of the `Readout` layers G_l:

$$G_l = W_{l+1} \tag{17}$$

- Depending upon the type of G_l weights, $\boxtimes$ in $Y_l^p = G_l \boxtimes S_l$ is either a `Dense` operation or a `Conv` operation.

Also, from the Eqs. (15) and (16), we get $(Y_l^p - Y_l^t)$ equal to either $\frac{\partial \mathcal{L}}{\partial W_{l+1}}(O_l^T)^{-1}$ or $\frac{\partial \mathcal{L}}{\partial W_{l+1}} \otimes \mathbb{O}_l^{-1}$ respectively. We therefore, next obtain the *local* truth values Y_l^t (for `Dense` and `Conv` cases) in our DALTON method. We start by noting that, since we had set $\frac{\partial \mathcal{L}_l}{\partial W_l} \approx \frac{\partial \mathcal{L}}{\partial W_l}$ (i.e., Eq. (14)), it implies $\frac{\partial \mathcal{L}}{\partial W_{l+1}} \approx \frac{\partial \mathcal{L}_{l+1}}{\partial W_{l+1}}$ in DALTON; Therefore,

- Case 1 - `Dense` W_l layers (from Eq. (15)):

$$\begin{aligned}Y_l^p - Y_l^t &= \frac{\partial \mathcal{L}_{l+1}}{\partial W_{l+1}}(O_l^T)^{-1} \\ &\equiv \frac{\partial \mathcal{L}_{l+1}}{\partial W_{l+1}}(S_l^T)^{-1} \qquad \because S_l \equiv O_l \text{ from Eq. (2) \& (10)} \\ &= \left[\left\{G_{l+1}^T(Y_{l+1}^p - Y_{l+1}^t)\right\} \odot \nabla\Theta(U_{l+1})\right] S_l^T(S_l^T)^{-1}\end{aligned} \tag{18}$$

Eq. (18) is obtained by substituting $\frac{\partial \mathcal{L}_{l+1}}{\partial W_{l+1}}$ with the expression in Eq. (12) for $l+1$, where $\boxtimes$ in $G_{l+1} \boxtimes S_{l+1}$ is a `Dense` operation, i.e., $\frac{\partial G_{l+1} S_{l+1}}{\partial S_{l+1}} = G_{l+1}^T$ Thus, from Eq. (18):

$$Y_l^t = Y_l^p - \left[\left\{G_{l+1}^T(Y_{l+1}^p - Y_{l+1}^t)\right\} \odot \nabla\Theta(U_{l+1})\right] \tag{19}$$

- Case 2 - `Conv` W_l layers (from Eq. (16)):

$$\begin{aligned}Y_l^p - Y_l^t &= \frac{\partial \mathcal{L}_{l+1}}{\partial W_{l+1}} \otimes \mathbb{O}_l^{-1} \\ &\equiv \frac{\partial \mathcal{L}_{l+1}}{\partial W_{l+1}} \otimes \mathbb{S}_l^{-1} \qquad \because S_l \equiv O_l \text{ from Eq. (2) \& (10)} \\ &= \left[\left\{\mathbb{G}_{l+1} \otimes (Y_{l+1}^p - Y_{l+1}^t)\right\} \odot \nabla\Theta(U_{l+1})\right] \otimes \mathbb{S}_l \otimes \mathbb{S}_l^{-1}\end{aligned} \tag{20}$$

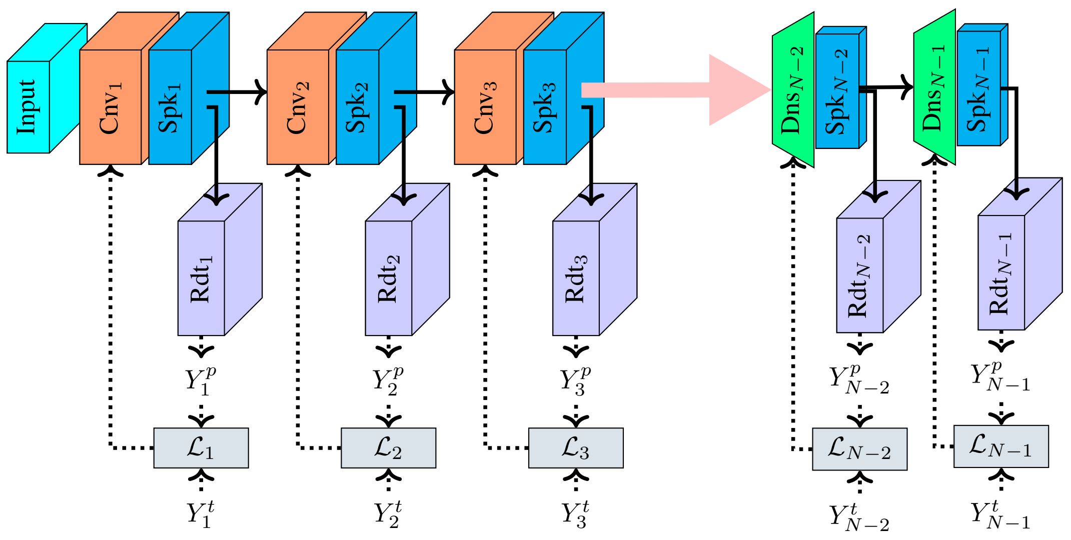


Fig. 2: *Spiking CNN's Architecture for Local Learning.* This architecture is synonymous with that of DECOLLE [17] and DALTON. "`Input`" is the binary-valued spike input, "Cnv_l" and "Dns_l" are the l^{th} trainable `Conv`olutional and `Dense` layers, "Spk_l" is the l^{th} layer of spiking units (e.g., `IF` neurons), and "Rdt_l" is the l^{th} non-trainable `Readout` layer. Solid **black** arrows indicate the flow of spike outputs and dashed arrows indicate the flow of computations to update the trainable layers. Y_l^p is the output from "Rdt_l", and Y_l^t is the truth value. $\mathcal{L}_l$ is the local loss computed from Y_l^p and Y_l^t, and used to update the "Cnv_l" / "Dns_l" layers' weights. Two differences in this architecture which set DALTON apart from DECOLLE is that in DALTON, the "Rdt_l" is not random and the Y_l^t is computed; more details later in the Discussion Section VI.

Eq. (20) is obtained by substituting $\frac{\partial \mathcal{L}_{l+1}}{\partial W_{l+1}}$ with the expression in Eq. (13) for $l+1$, where $\boxtimes$ in $G_{l+1} \boxtimes S_{l+1}$ is a `Conv` operation, i.e., $\frac{\partial G_{l+1} * S_{l+1}}{\partial S_{l+1}} = \mathbb{G}_{l+1}$. Thus, from Eq. (20):

$$Y_l^t = Y_l^p - \left[\left\{\mathbb{G}_{l+1} \otimes (Y_{l+1}^p - Y_{l+1}^t)\right\} \odot \nabla\Theta(U_{l+1})\right] \quad (21)$$

Both the Eqs. (19) and (21) are obtained under the assumption that the right inverse of S_l^T and $\mathbb{S}_l$ exist, and the inverse operation holds the Identity relation; more details about this assumption can be found in our Suppl. doc - Appx. B. An important *catch* to consider while computing the local ground-truths Y_l^t for the `Conv` W_l layers using the Eq. (21) is for the last (N_C^{th}) and the penultimate ($(N_C-1)^{\text{th}}$) `Conv` layers. Note that in the Eq. (21), one needs to use the $(l+1)^{\text{th}}$ `Readout` layer weights, i.e., G_{l+1}; thus for the N_C^{th} and $(N_C-1)^{\text{th}}$ layers, one has to consider G_{N_C+1} and G_{N_C} `Readout` layer weights respectively. Recollect that $G_l = W_{l+1}$ (Eq. (17)), thus, $G_{N_C+1} = W_{N_C+2}$ and $G_{N_C} = W_{N_C+1}$, where the weights W_{N_C+2} and W_{N_C+1} are `Dense` layer weights. Thus, to compute the Y_l^t values for the last two `Conv` layers, one should use the Eq. (19) instead of Eq. (21). Table II provides the summary on setting G_l and the choice of the equations to compute Y_l^t for different layers. Note that for the last $(N-1)^{\text{th}}$ `Readout` layer, $Y_{N-1}^t = Y^t$, where Y^t is the *global* ground-truth. A *key observation* from the above derivations is that the Eq. (17) uses the *local* $(l+1)^{\text{th}}$ weights, and Eqs. (19) and (21) use the *local* $(l+1)^{\text{th}}$ `Readout` layer weights and surrogate-derivatives (as well as the *local* predicted & truth values); thus DALTON is *local* in space, as well as *online-in-time*.

1) Relation of DALTON with vanilla-TFR based learning : As stated above, we wanted to find alternatives to using *random* `Readout` matrices G_l and the *global* ground-truths Y^t as the *local* truth-values in DECOLLE. In our derivations, we have determined the values of G_l and Y_l^t for each layer. We next juxtapose the vanilla-TFR/DECOLLE [17] equation and ours, i.e., Eqs. (12) and (13) of DALTON. In relation to the Eq. (7) in [17], i.e., $\Delta W_l^{ij} = -\eta \frac{\partial L_l}{\partial S_l^i} \sigma'(U_l^i) P_l^j$, which is in the form of TFR, the pre-synaptic factor P_l^j in our case is S_{l-1} – Eqs. (12) and (13) (note that P_l^j is a function of S_{l-1}^j in Eq. (4) of [17]), and the post-synaptic factor $\sigma'(U_l^i)$ remains unchanged (we too use surrogate-derivatives in place of $\nabla\Theta(U_l)$ – Eqs. (12) and (13)). However, in relation to the third factor $\frac{\partial L_l}{\partial S_l^i} = \sum_k G_l^{ki}(Y_l^k - \hat{Y}_l^k)$ -Eq. (8) in [17], although the form in our case is similar, we use different values of G_l and $\hat{Y}_l$ (-Y_l^t in our case). With the usage of the Eq. (17) and Eqs. (19)/(21) for G_l and Y_l^t respectively, we transfer the *information* in depth via the non-trainable `Readout` layers to the trainable `Conv` & `Dense` layers in our DALTON method. Note: In addition to the relation of DALTON with vanilla-TFR, we also present DALTON's contrast with Back-propagation and DECOLLE in the Sections VI-B and VI-C respectively.

TABLE II: *Rules to obtain G_l and Y_l^t in DALTON method.*

Case	G_l	Y_l^t values	Case	G_l	Y_l^t values
$l > N_C$	D_{l+1}^w	Eq. (19)	$l = N_C - 1$	C_{l+1}^w	Eq. (19)
$l = N_C$	D_{l+1}^w	Eq. (19)	$l \leq N_C - 2$	C_{l+1}^w	Eq. (21)

C. Implementation of DALTON method to train SNNs

We now lay down the implementation details of training our Conv-SNNs with the DALTON method. We have considered two common training settings in our experiments – Back-propagation Through Time (BPTT) and Real Time Recurrent Learning (RTRL); both BPTT and RTRL to train SNNs make use of the surrogate derivatives to overcome the non-differentiability of spikes. Note that, we use $\sigma'(x) = (1 +$

TABLE III: *Our considered Conv-SNN Architectures.* Strings separated by "-" are called blocks here. Block – "XcY" ⇒ X Conv kernels each of dimension (Y×Y) with stride (1 × 1), "XcYsZ" ⇒ X Conv kernels each of dimension (Y×Y) with stride (Z×Z), "XcYpZ" ⇒ X Conv kernels each of dimension (Y×Y) with AveragePool size (Z×Z), and "fX" ⇒ fully connected Dense layer with X output neurons. Each Conv layer in the Average-Pooling based architectures has a stride size of (1 × 1). At the end of each blue, red, and teal colored blocks, a dropout with probability 0.1, 0.2, and 0.3 is applied. The spiking non-linearity is applied at the end of each block, prior to the application of dropout (if present). Note that there are no biases in all the architectures, and the *A4* and *A5* archs have the same depth, but different width. Best viewed in color.

Archs.	**Strided-Convolution**	**Average-Pooling**
A1	16c3s2-f128-f64	16c3p2-f128-f64
A2	32c7s2-32c5s2-f512-f128	32c7p2-32c5p2-f512-f128
A3	48c5s2-64c3s2-128c3s2-f512-f256	48c5p2-64c3p2-128c3p2-f512-f256
A4	24c3-24c3s2-48c3s2-48c3s2-f256-f128	24c3-24c3p2-48c3p2-48c3p2-f256-f128
A5	64c3-64c3s2-128c3s2-128c3s2-f1024-f1024	64c3-64c3p2-128c3p2-128c3p2-f1024-f1024

$|x|)^{-2}$ as the surrogate derivative [22] in our experiments, where $x = U(t) - U_{\text{thr}}$. In BPTT, the network weights are updated at the end of *all* the simulation time steps; thus, a computational graph of all the time-steps has to be maintained which is memory-intensive. A simple way to do memory-efficient training is via RTRL, i.e., updating the network weights *each* time-step, thereby, forgoing the need to maintain the computational graph of all the simulation time-steps. Since the DALTON method is derived by considering *one* time-step operation, this makes it inherently online-in-time and suitable for RTRL-based training, and by extension, applicable for BPTT-based training too. Note that, for both BPTT and RTRL, we compute Y_l^t and update G_l at the end of every batch.

An important observation in Eq. (21) for Y_l^t is that the Jacobian of the Conv operation i.e., $\frac{\partial G_{l+1} * S_{l+1}}{\partial S_{l+1}} = \mathbb{G}_{l+1}$ is composed of G_{l+1} filter weights alone. Therefore, for a batch of input samples (during one iteration), we calculate the Jacobian of the Conv operation with respect to one sample, and copy it for the entire batch; such *optimization* greatly improves the computational efficiency of training Conv-SNNs with DALTON. For the Conv-SNNs with a Pooling layer immediately following the Conv layer, one needs to take the Jacobian of the Pooling operation too (derivations are in the Suppl. doc - Appx. C). In the case of AveragePooling operation, a similar optimization (as for the Jacobian of Conv with respect to spikes) can be applied; this holds because the AveragePooling weights remain the same for all the samples in a batch. However, such an optimization does not hold in case of the MaxPooling operation, because the Jacobian of the MaxPooling output varies for each input.

V. Experiments

To evaluate the efficacy of DALTON, we compare our results with that of the vanilla-TFR and SurrGD (for SurrGD, Conv-SNNs do not have local Readout layers). Note that vanilla-TFR, *by design* is limited compared to SurrGD, and we expect our results with DALTON to be an *improvement* over the vanilla-TFR. Also, note that our goal is *not* to beat the State-of-The-Art (SoTA) results on the experimented datasets, rather to show the effectiveness of DALTON in leveraging depth over vanilla-TFR. Although recent works in the SNN domain [8], [23] have established new SoTA results on our experimented datasets, our experimental-settings differ widely from theirs; also, [8], [23] are not in local-learning domain. Therefore, we conclude that it is fair to compare our local-learning method DALTON in a controlled experimental-setting with our own developed architectures and implementations of vanilla-TFR and SurrGD. Note that we acknowledge works – [16], [17], [37] which fall in the local-learning domain.

A. On the choice of Datasets and Architectures

We experiment with 5 architectures of varying depths under two training-settings – BPTT and RTRL. For BPTT and RTRL each, we employ two variations of each of our architectures – one with Strided-Convolutions and another with Average-Pooling; all the architectures have been well described in the Table III. We refer to the individual architectures as $AxSC$ and $AxAP$ ($\forall\ x \in [1, 5]$), respectively denoting the Architecture x under the Strided-Convolution and Average-Pooling columns in the Table III. We experiment on three static-images datasets – MNIST, FMNIST, and CIFAR10, and on three event-based images datasets – N-MNIST, DVS128-Gesture, and DVS-CIFAR10 (datasets' description in Suppl. doc - Appx. E.1).

B. Model details

The encoding neurons in the Input layer (in Fig. 2) follow the discrete-time dynamics: $I_l^i[t] = \alpha \times \mathbf{x}_l^i[t] + \beta$ and $U_l^i[t] = U_l^i[t-1] + I_l^i[t]$ to encode the input values $\mathbf{x}_l^i[t]$ to the binary spikes, where α and β are the neuron's gain and bias values respectively; and $\mathbf{x}[t]$ are either scaled pixel values or event-based input spikes. The IF neurons in the spiking layers, i.e., Spk$_l$ corresponding to the Conv/Dense layers in Fig. 2 follow the neuron dynamics: $I_l^i[t] = \gamma I_l^i[t-1] + \sum_j W_l^{ij} S_{l-1}^j[t]$ and $U_l^i[t] = U_l^i[t-1] + I_l^i[t]$, where γ is the neuron's current decay constant ($\gamma = \exp(-\Delta t/\tau_{\text{cur}})$, $\Delta t = 0.001$). Values of the hyper-parameters α, β, and τ_{cur} are in the Table IVa.

C. Experiment Procedure

Table IVb presents the comprehensive set of all our experiments for the BPTT and RTRL training-settings, with the two variations of *A1*–*A5* architectures, i.e., Strided-Convolution and Average-Pooling. Each cell in the Table IVb denotes one experiment-*setting*, where for each *setting*, all the *combinations* of τ_{cur}, α, and β are considered for experiments with the applicable datasets; each *combination* is repeated for three

TABLE IV: *All Experiment Settings.*

(a) *Hyper-Parameters (HPs)* $\forall$ *Archs.* $x \in [1, \cdots, 4]$ in 2^{nd} column, $\tau_{cur} = 0.001$ and $\alpha = 1$ for DVS-CIFAR10.

HPs	***AxSC* & *AxAP***	***A5SC* & *A5AP***
τ_{cur}	$\begin{Bmatrix}0.001,\\0.005\end{Bmatrix}$	$\begin{Bmatrix}0.001,\\0.005\end{Bmatrix}$
α	[1, 2]	[1]
β	[0]	[0]

(b) *Set of our experiments.* For each cell (or experiment-setting), the ✓⇒ experiments conducted with at least one dataset, and ✗⇒ no experiments conducted. Also, for each cell, all the combinations of τ_{cur}, α, and β in Table IVa are considered -each combination repeated three times.

		Strided-Convolution					**Average-Pooling**				
		A1	A2	A3	A4	A5	A1	A2	A3	A4	A5
BPTT	SurrGD	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	vanilla-TFR	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	DALTON	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RTRL	vanilla-TFR	✓	✓	✗	✗	✓	✓	✓	✓	✗	✓
	DALTON	✓	✓	✗	✗	✓	✓	✓	✓	✗	✓

TABLE V: *Accuracy Results from the BPTT experiments with Strided-Conv (SC) Architectures.*

SC Arch	**MNIST**			**FMNIST**			**CIFAR10**		
	SurrGD	**vanilla-TFR**	**DALTON**	**SurrGD**	**vanilla-TFR**	**DALTON**	**SurrGD**	**vanilla-TFR**	**DALTON**
A1	98.78, 0.07	98.00, 0.15	**98.23, 0.12**	90.73, 0.06	89.17, 0.13	**89.57, 0.45**	58.57, 0.62	48.25, 0.88	**57.37, 0.20**
A2	99.39, 0.04	98.99, 0.01	**99.14, 0.05**	90.69, 0.27	89.31, 0.13	**90.54, 0.07**	66.85, 0.21	54.46, 0.6	**65.48, 0.14**
A3	99.43, 0.01	**99.00, 0.04**	98.37, 0.05	90.50, 0.04	**89.47, 0.11**	88.78, 0.19	66.58, 0.31	53.77, 0.71	**63.40, 0.18**
A4	99.40, 0.04	98.32, 0.10	**98.68, 0.12**	90.35, 0.04	86.71, 0.29	**88.57, 0.31**	65.60, 0.60	42.98, 1.16	**59.43, 0.21**
A5	-	-	-	-	-	-	72.99, 0.05	49.31, 0.2	**62.83, 1.06**

TABLE VI: *Accuracy Results from the BPTT experiments with Average-Pooling (AP) Architectures.*

AP Arch	**MNIST**			**FMNIST**			**CIFAR10**		
	SurrGD	**vanilla-TFR**	**DALTON**	**SurrGD**	**vanilla-TFR**	**DALTON**	**SurrGD**	**vanilla-TFR**	**DALTON**
A1	98.84, 0.07	98.17, 0.06	**98.29, 0.15**	91.24, 0.11	89.75, 0.08	**90.08, 0.24**	59.28, 0.42	49.31, 1.98	**57.84, 0.09**
A2	99.42, 0.03	**98.83, 0.05**	98.48, 0.03	89.99, 0.03	**88.58, 0.10**	88.17, 0.28	63.92, 0.41	51.74, 0.46	**62.99, 0.31**
A3	99.28, 0.04	**98.95, 0.04**	98.01, 0.20	89.20, 0.16	**87.82 , 0.11**	87.09, 0.08	72.82, 0.04	58.69, 0.32	**67.21, 0.24**
A4	99.33, 0.06	98.44, 0.07	**98.72, 0.06**	89.19, 0.15	85.75, 0.28	**87.35, 0.21**	71.99, 0.11	48.18, 1.12	**62.82, 0.53**
A5	-	-	-	-	-	-	78.25, 0.05	55.15, 0.30	**66.46, 0.46**

`SEED`s $\in \{0, 3, 100\}$. The learning rate η is set to 0.001 for all the experiments, and is multiplied by 0.5 every 30 epochs. The MSE loss function is chosen for all the experiments; note that for the SurrGD experiments, *global* MSE loss is calculated only at the final `Readout` layer, which is set to trainable; and for the vanilla-TFR and DALTON experiments, *local* MSE loss is calculated for each `Readout` layer l (set untrainable).

1) BPTT training-setting: Under the BPTT training-setting, we compare DALTON with vanilla-TFR and SurrGD (results in Tables V and VI). All the BPTT experiments are conducted with MNIST, FMNIST, and CIFAR10 datasets, except for the *A5SC* and *A5AP* architectures where only CIFAR10 dataset is considered; the number of training epochs for MNIST, FMNIST, and CIFAR10 are 50, 100, and 150 respectively, with batch size = 250 for all. For the SurrGD (and vanilla-TFR) experiments, the `Output` layer's (and each `Readout` layer's) loss is calculated over the mean of the predicted logits across all the simulation time-steps. However, for the DALTON experiments, while the output layer's (i.e., the final `Readout` layer's) loss is calculated over the mean of the predicted logits, the intermediate `Readout` layers' loss is calculated over all the simulation time-steps.

2) RTRL training-setting: Under the RTRL training-setting, we compare DALTON with vanilla-TFR alone (conventionally, SurrGD was developed for the BPTT setting, results in Table VII). All the RTRL experiments are conducted with N-MNIST, DVS128-Gesture, & DVS-CIFAR10 datasets, except for *A5SC* and *A5AP* archs where only CIFAR10 is considered. The number of training epochs for N-MNIST, DVS128-Gesture, DVS-CIFAR10, and CIFAR10 are $50, 150, 200$, and 100 resp., with batch-size: $250, 25, 500$, and 5000 resp. For both, DALTON & vanilla-TFR experiments, all the `Readout` layers' loss is calculated over the predicted logits each time-step.

VI. Result Analysis & Discussion

We now discuss the results, and summarize the differences between DALTON and Back-propagation & vanilla-TFR. For all the experiments, we calculate test accuracy over the entire test set every epoch. We then take the mean (and std.) of the maximum test accuracy (over all the epochs) for a single combination of τ_{cur}, α, and β, over three `SEED`s. We then finally select the maximum mean accuracy (and related std.) over all the combinations of τ_{cur}, α, and β for the experiments conducted with one dataset, one architecture, and with one training method, and report the same in each cell of the Tables V, VI, VIIa, and VIIb; (x, y) in a cell implies (mean accuracy $\pm$ std.). Note that, for determining the efficacy of DALTON, we compare its results here with that of vanilla-TFR only, and not with SurrGD (a comparison with recent STDP-based methods is in the Suppl. doc - Appx. D); this is because vanilla-TFR and DALTON are *local* learning methods; whereas, SurrGD is not, and is expected to perform better than vanilla-TFR and DALTON both, by the virtue of *global* back-propagation. Also, although DALTON's results do not establish a new SoTA on the datasets, they show the performance improvement over vanilla-TFR/DECOLLE, which is exactly what we aimed for.

TABLE VII: *Accuracy Results from the RTRL experiments with Strided-Conv (SC) and Average-Pool (AP) Architectures.*

(a) *Architectures 1 and 2 Results on event-based image datasets.*

	Strided-Conv				**Average-Pool**					
	N-MNIST		**DVS128-Gesture**		**N-MNIST**		**DVS128-Gesture**		**DVS-CIFAR10**	
	v-TFR	**DALTON**	**v-TFR**	**DALTON**	**v-TFR**	**DALTON**	**v-TFR**	**DALTON**	**v-TFR**	**DALTON**
A1	94.77, 0.31	**95.96, 0.04**	84.67, 0.38	**87.73, 0.75**	95.51, 0.42	**96.78, 0.13**	86.27, 0.82	**88.4, 0.98**	42.00, 0.00	**43.20, 0.02**
A2	**98.12, 0.07**	97.68, 0.12	88.93, 0.50	**90.13, 0.82**	**98.21, 0.10**	97.96, 0.17	88.13, 2.32	**91.07, 0.75**	45.93, 0.02	**52.17, 0.01**

(b) *Architecture 5 Results.*

A5		**CIFAR10**
SC	**v-TFR**	42.25, 0.46
	DALTON	**62.30, 0.19**
AP	**v-TFR**	48.98, 0.42
	DALTON	**66.60, 0.11**

With *A3* on DVS-CIFAR10 and Average-Pool variation, we obtained 50.13, 0.01 and $\mathbf{56.47}, \mathbf{0.02}$ with vanilla-TFR (v-TFR) and DALTON respectively.

A. Analysis of the accuracy results in Tables V, VI, and VII

As can be seen in the Tables V and VI, for the experiments (under BPTT setting) with simple datasets – MNIST and FMNIST, DALTON performs slightly better than vanilla-TFR for the most part. On the other hand, DALTON clearly performs *much better* than vanilla-TFR on the more complex CIFAR10. As expected, in case of the experiments with CIFAR10, SurrGD leverages the depth and width of the *individual* networks (*A1*–*A5*) well, while vanilla-TFR fails to do so and performs poorly. We note that in some cases of *A1*–*A3*, the performance of DALTON is *very close* to that of the SurrGD, and much better than vanilla-TFR for *A4* and *A5*. Also note that, our experiments' emphasis is on showing that given a network of any specific *depth* and *width*, DALTON leverages *them* better than vanilla-TFR (more details in the Suppl. doc - Appx. E.2). An important observation (in the Tables V and VI) with respect to MNIST and FMNIST is that, SurrGD performs nearly same for *A1*–*A4*, which implies that *depth and width have little role to play in case of simple datasets* – MNIST and FMNIST; this explains why vanilla-TFR and DALTON performed similar.

Coming to the Table VIIa, for the RTRL-setting experiments with N-MNIST, DVS128-Gesture, and DVS-CIFAR10, we see that for N-MNIST, a clear winner between vanilla-TFR and DALTON does not emerge (for both Strided-Conv and Average-Pool). However, for the complex DVS128-Gesture and DVS-CIFAR10 datasets, we see that DALTON performs better than vanilla-TFR for both *A1* and *A2*. The Table VIIb with *A5*'s accuracies on CIFAR10 under the RTRL setting, further gives the compelling empirical proof of *DALTON's efficacy to leverage depth better than vanilla-TFR* for local learning in SNNs. Note that Kaiser et al. [17] also experimented with the N-MNIST and DVS128-Gesture datasets under the RTRL setting, and their DECOLLE method achieved approximately 96% and 95.5% accuracies respectively (they used a small subset of the N-MNIST dataset). In our experiments with N-MNIST and DVS128-Gesture, we have tried to stay true to their (i.e., [17]'s) experimental settings, except that we have used the entirety of the N-MNIST dataset for training and evaluation (Kaiser et al. [17] used only a part of it). We also note that Kaiser et al. [17] found that increasing the depth of their network did *not* help improve the DECOLLE's performance, which echoes our experimental findings with the vanilla-TFR. Also, in a related work on local learning in SNNs [16], we note that the authors achieved approximately 97% and 86% accuracy on MNIST and FMNIST respectively, ours with DALTON (and vanilla-TFR) is unarguably better, as seen in the Tables V and VI. With respect to the latest work on TFR, i.e., the ETLP rule [37], the authors achieved 94.30% on the N-MNIST dataset (although, they used a simple Dense SNN).

B. Contrast – DALTON vs. Back-propagation

Despite DALTON's theoretical foundations in the Back-propagation theory, there are certain obvious differences between them; we highlight those below. First off, in Back-propagation, a *global* loss $\mathcal{L}$ at the final `Output` layer is computed, such that $\frac{\partial \mathcal{L}}{\partial W_l}$ is valid (and computed) $\forall\, l \in [1, N]$. However, in case of DALTON, a *local* loss $\mathcal{L}_l$ is computed for every $l \in [1, N-1]$, and $\frac{\partial \mathcal{L}_l}{\partial W_l}$ is *not* valid for every l, rather $\frac{\partial \mathcal{L}_l}{\partial W_m} = 0,\ \forall\ m \neq l$; i.e., only $\frac{\partial \mathcal{L}_1}{\partial W_1}, \frac{\partial \mathcal{L}_2}{\partial W_2} \cdots, \frac{\partial \mathcal{L}_{N-1}}{\partial W_{N-1}}$ are valid. Next, while computing $\frac{\partial \mathcal{L}}{\partial W_l}$ in Back-propagation, all the terms (weights, derivatives) of the next deeper layers $[l+1, N]$ are considered (see Fig. 3), whereas, while computing $\frac{\partial \mathcal{L}_l}{\partial W_l}$ in DALTON, only local $(l-1)^{\text{th}}, l^{\text{th}}, (l+1)^{\text{th}}$ terms are required – due to the approximation $\frac{\partial \mathcal{L}_l}{\partial W_l} \approx \frac{\partial \mathcal{L}}{\partial W_l}$ in Eq. (14). Note that in DALTON, the third "error" term comprises of the local targets Y_l^t, which is also computed with locally available terms (as can be seen in Eqs. (19) and (21)). Although, careful readers would note the "chained locality" in the computation of Y_l^t (as it is also dependent upon Y_{l+1}^t), where the local targets have to be computed layer-after-layer from the global ground truth (see Fig. 3). On one hand, while this successive dependency contributes to *effectively leveraging depth* in SNNs, on the other, it implies that one has to wait for the computation of all Y_l^ts *before* updating the network weights in parallel; i.e., *after* the Y_l^ts are available to each l^{th} layer, subsequent weight updates can be done *locally in parallel*. Note that all the Y_l^ts can be computed in *just one* time-step by consecutively passing the *already* computed Y_{l+1}^t and Y_{l+1}^p to the l^{th} layer. This is true because the other terms in the computation of Y_l^t are not only local, but also *readily* available in the current time-step – as demonstrated by the success of our RTRL experiments. Note that in vanilla-TFR/DECOLLE's case too (see Fig. 3), one has to propagate the global ground truth to all the intermediate trainable layers in *one* time-step (which also requires dedicated hardware support). Overall, DALTON attempts to bring the best of both – the local-learning three-factor based rule and the global Back-propagation based SurrGD.

C. Contrast – DALTON vs. vanilla-TFR/DECOLLE

As described in the Section IV-B1, DALTON conforms to the vanilla-TFR formulation with three local factors. However, the major difference between DALTON and vanilla-TFR is the way the third 'error-signal' factor is computed. In vanilla-TFR, the third error factor is $\frac{\partial \mathcal{L}_l}{\partial S_l} = \sum G_l(Y_l^p - Y_l^t)$, where the local target Y_l^t and the readout matrix G_l remain *static* throughout the training of the SNN; Y_l^t is set equal to the global ground truth and G_l is randomly (or uniformly) initialized, for every layer l. However, in case of DALTON, although the third error factor "expression" remains the same, Y_l^t and G_l are *computed*

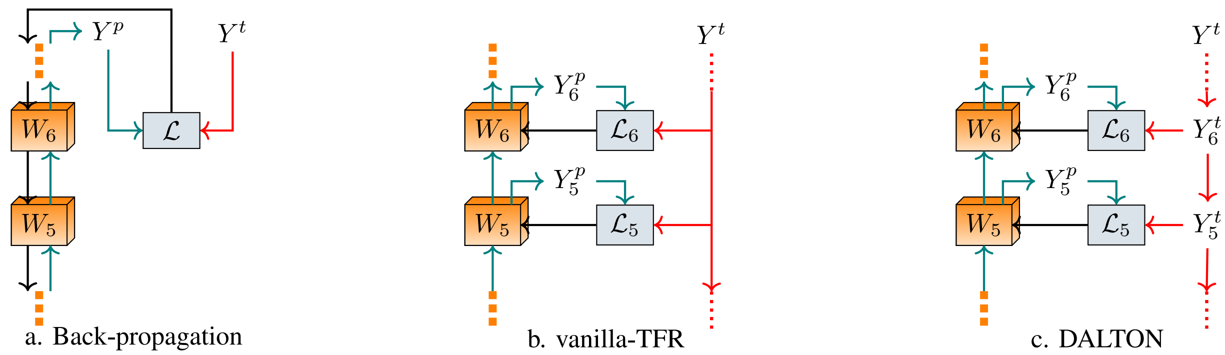


Fig. 3: *Abstract models of loss computation and gradient flow* – (a). "Back-propagation", (b): "vanilla-TFR", & (c): "DALTON". W_l are the trainable weights; $\mathcal{L}$ in (a). "Back-propagation" is the *global* loss computed from the predicted output Y^p and the *global* ground truth Y^t; and $\mathcal{L}_l$ in both, (b). "vanilla-TFR" and (c). "DALTON" are the *local* losses. In (b). "vanilla-TFR", $\mathcal{L}_l$ are computed via the *locally* predicted outputs Y_l^p and the *global* ground truth Y^t; whereas, in (c). "DALTON", $\mathcal{L}_l$ are computed via the *locally* predicted outputs Y_l^p and the *locally* computed targets Y_l^t (unlike vanilla-TFR). Red arrows show the propagation/computation of targets, teal arrows show the forward propagation, and black arrows show the gradient flow. Note that, for the sake of clarity and contrast, these models don't show all the relations to compute the variables, e.g., Y_l^p, Y_l^t, etc.

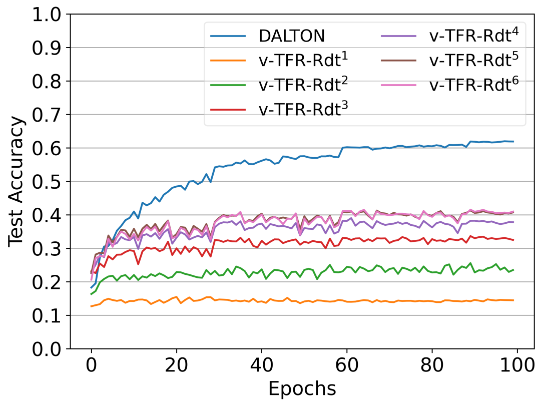
for every layer l after processing each batch, as per the Eqs. (19)/(21) and Eq. (17) respectively. On one hand where the Eq. (17) is simply an assignment operation, on the other, Eqs. (19) and (21) imply – one has to compute the local truth values Y_l^t for each trainable layer, which requires matrix/tensor multiplications, as well as the Jacobian computation of the `Conv` operation with respect to the input spikes. Furthermore, in case of `Pooling` layers in the architectures, one would also be required to compute the Jacobian of the `Pooling` operation. Although one can leverage the *optimizations* suggested in the Section IV-C to improve the run-time efficiency of computing Y_l^t in batch-wise training, the proposed optimizations do not hold in the case of online-in-sample training. Coming to the algorithmic-complexity comparison between vanilla-TFR and DALTON, consider an SNN with C and D number of `Conv` and `Dense` layers respectively, where the cost of each `Conv` and `Dense` layer *forward-pass* operation is upper-bounded by $O(c_f)$ and $O(d_f)$ respectively, and *backward-pass* operation is upper-bounded by $O(c_b)$ and $O(d_b)$ respectively (note that layer-wise local loss $\mathcal{L}_l$ is computed). For such an SNN, in one forward and backward pass, vanilla-TFR would compute $O((c_f + c_b)C + (d_f + d_b)D)$ operations. However, since DALTON computes Y_l^t for each layer (differing by layer-type), consider it to be upper bounded by $O(y_c)$ and $O(y_d)$ for each `Conv` and `Dense` layers respectively. Thus, for the same SNN, in one forward and backward pass, DALTON computes $O((c_f + c_b + y_c)C + (d_f + d_b + y_d)D)$ operations (see Fig. 3b & 3c, DALTON algorithms in Suppl. doc - Appx. E.3). With respect to DALTON's memory footprint, in the present form, it is heavily memory inefficient due to the *explicit* computations of Jacobians and matrix/tensor multiplications to compute the local truths Y_l^t. Also, the computation of Y_l^t needs a stored copy of the spikes S_{l+1} and of the surrogate-derivatives of the layer $l+1$. Thus, compared to the vanilla-TFR method, a naive implementation of DALTON is not computationally efficient.

D. Possible applications of DALTON

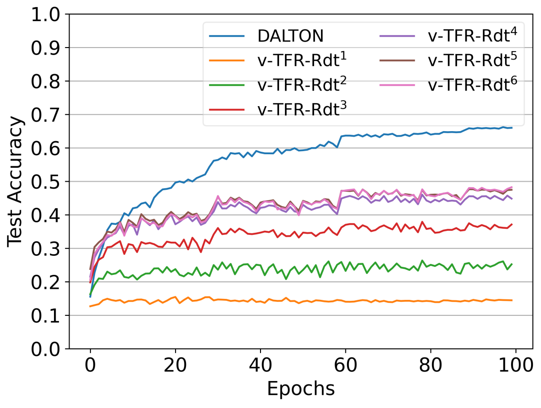
With the above computational limitations of DALTON in the present form, there is still a silver lining to our proposed method. For online cases (both in time and sample), where *part* of the whole data/signal arrives each time-step *and* is widely spaced in time (i.e., the system waits for a few time-steps before another sample arrives), DALTON can be useful over the vanilla-TFR. Examples of such systems are Remote Sensing and Satellite Communication systems, Edge devices, IoT applications, etc. [42]–[45]; basically, systems where the on-site data processing is limited due to the design constraints, and the data has to be transmitted over long time-duration. This would enable DALTON to execute the compute intensive operations in time. We particularly emphasize on the Eq. (19) to obtain the local truths Y_l^t for the `Dense` layers, which does *not* involve the computation of Jacobian and Tensordot operations, rather a matrix multiplication and the Hadamard product. This makes the computation of Y_l^t for `Dense` layers relatively cheaper than the computation of Y_l^t for the `Conv` layers; we observed the same in our experiments. Thus, for Time Series Classification with SNNs composed of `Dense` layers alone [3], [14], DALTON can be easily (and cheaply) applied with better performance over the vanilla-TFR. This is because DALTON offers faster learning than vanilla-TFR *per network-update* (see Fig. 4, more such results in the Suppl. doc - Appx. F). We note that the Jacobians are computed for conventional Back-propagation too, and with proper hardware & library/CUDA support, the Jacobian computations for DALTON can be made highly efficient. Also, neuromorphic h/w is a constantly innovating area, thus, the chained computation of Y_l^t can be achieved with dedicated h/w design and support.

VII. Conclusion and Future Work

In this work, we presented our novel local-learning three-factor based DALTON method to train the SNNs (`Conv` and `Dense` variants). DALTON method preserves the locality of the weight-updates, while also simultaneously leveraging the depth of the network-architecture; thereby, bringing a balance between the vanilla-TFR and SurrGD methods – however, at the cost of high compute and memory resource requirements. Although the DALTON method has its computational limitations, it is useful in cases where the network's weight-updates



(a) *A5SC* – CIFAR10



(b) *A5AP* – CIFAR10

Fig. 4: *Accuracy Plots* – RTRL Experiments with *A5SC* and *A5AP* architectures on CIFAR10. The accuracies plotted here are obtained as explained in the Section VI's start. v-TFR-Rdt^l implies the output from the l^{th} `Readout` layer of the network trained with vanilla-TFR. Note that with DALTON, we are able to achieve better results than vanilla-TFR quickly over epochs. The (almost) smooth increase in DALTON's accuracy and eventual saturation hints towards its stable and convergent learning. More details on DALTON's training efficiency (stability and convergence) can be found in the Suppl. doc - Appx. H.

can be afforded to be slow. Note that, as explained in the Section II, three-factor rules based methods to locally train the SNNs are limited in literature; where, either the performance does not scale with the network depth (e.g., DECOLLE), or the trained architectures are too simple (e.g., ETLP). This work being one of the very few neuromorphic-friendly three-factor based local-learning methods to train deep SNNs, contributes a new direction to locally train them, and has lots of room to grow and improve. In this work, we not only *empirically* showed the efficacy of DALTON to effectively leverage depth of complex SNN architectures (with experiments on 6 datasets of varying complexity), but also *theoretically* substantiated it. Though we do not explicitly compare DALTON with the results obtained by the authors of DECOLLE [17] and ETLP [37] in our tables, we do provide a comparison with them at the end of the Section VI-A, where DALTON demonstrates better results over the SoTA local-learning ones. With respect to the current computational limitations of the DALTON method mentioned in the immediately previous section, one can look into leveraging the computational optimizations in the deep learning libraries, and reducing the computation and memory requirements by also leveraging the already stored spikes and surrogate derivatives (as they are part of the current time-step's computational graph) – instead of computing and maintaining a separate copy. Moreover, the usage of the current time-step's $(l+1)^{th}$ layer's weights as the l^{th} `Readout` layer's weights implies that one can completely forgo the actual manifestation of the `Readout` layers in the SNNs for local-learning, and reuse the existing stored weights of the main trainable network. Also, for the SNNs composed of fully-connected `Dense` layers alone, direct hardware implementation of DALTON would be easier and offer faster computations than for Conv-SNNs. Nonetheless, *DALTON offers quality weight-updates per time-step (compared to the vanilla-TFR), which enables the SNNs to learn faster and better*, and that too in a neuromorphic hardware-friendly manner (by the virtue of the three-factor formulation) – this should be the key takeaway of our work.

Funding

This work was supported in part by the U.S. National Science Foundation (NSF) under Grant CCF-1750450, Grant ECCS-1731928, Grant ECCS-2128594, Grant ECCS-2314813, and Grant CCF-1937487.

References

[1] J. Wu, E. Yılmaz, M. Zhang, H. Li, and K. C. Tan, "Deep spiking neural networks for large vocabulary automatic speech recognition," *Frontiers in neuroscience*, vol. 14, p. 199, 2020.

[2] A. Qammaz and A. A. Argyros, "Mocapnet: Ensemble of snn encoders for 3d human pose estimation in rgb images." in *BMVC*, 2019, p. 46.

[3] H. Fang, A. Shrestha, and Q. Qiu, "Multivariate time series classification using spiking neural networks," in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–7.

[4] R. Gaurav, T. C. Stewart, and Y. C. Yi, "Spiking reservoir computing for temporal edge intelligence on loihi," in *2022 IEEE/ACM 7th Symposium on Edge Computing (SEC)*. IEEE, 2022, pp. 526–530.

[5] B. Rueckauer, C. Bybee, R. Goettsche, Y. Singh, J. Mishra, and A. Wild, "Nxtf: An api and compiler for deep spiking neural networks on intel loihi," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 18, no. 3, pp. 1–22, 2022.

[6] H. Zheng, Y. Wu, L. Deng, Y. Hu, and G. Li, "Going deeper with directly-trained larger spiking neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, 2021.

[7] Z. Yan, J. Zhou, and W.-F. Wong, "Near lossless transfer learning for spiking neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 12, 2021, pp. 10 577–10 584.

[8] Q. Meng, M. Xiao, S. Yan, Y. Wang, Z. Lin, and Z.-Q. Luo, "Training high-performance low-latency spiking neural networks by differentiation on spike representation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 12 444–12 453.

[9] W. Fang, Z. Yu, Y. Chen, T. Huang, T. Masquelier, and Y. Tian, "Deep residual learning in spiking neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 21 056–21 069, 2021.

[10] Y. Yan, T. C. Stewart, X. Choo, B. Vogginger, J. Partzsch, S. Höppner, F. Kelber, C. Eliasmith, S. Furber, and C. Mayr, "Comparing loihi with a spinnaker 2 prototype on low-latency keyword spotting and adaptive robotic control," *Neuromorphic Computing and Engineering*, vol. 1, no. 1, p. 014002, 2021.

[11] S. Höppner, Y. Yan, A. Dixius, S. Scholze, J. Partzsch, M. Stolba, F. Kelber, B. Vogginger, F. Neumärker, G. Ellguth *et al.*, "The spinnaker 2 processing element architecture for hybrid digital neuromorphic computing," *arXiv preprint arXiv:2103.08392*, 2021.

[12] G. Orchard, E. P. Frady, D. B. D. Rubin, S. Sanborn, S. B. Shrestha, F. T. Sommer, and M. Davies, "Efficient neuromorphic signal processing with loihi 2," in *2021 IEEE Workshop on Signal Processing Systems (SiPS)*. IEEE, 2021, pp. 254–259.

[13] M. Davies, A. Wild, G. Orchard, Y. Sandamirskaya, G. A. F. Guerra, P. Joshi, P. Plank, and S. R. Risbud, "Advancing neuromorphic computing with loihi: A survey of results and outlook," *Proceedings of the IEEE*, vol. 109, no. 5, pp. 911–934, 2021.

[14] R. Gaurav, T. C. Stewart, and Y. Yi, "Reservoir based spiking models for univariate time series classification," *Frontiers in Computational Neuroscience*, vol. 17, p. 1148284, 2023.

[15] A. Taherkhani, A. Belatreche, Y. Li, G. Cosma, L. P. Maguire, and T. M. McGinnity, "A review of learning in biologically plausible spiking neural networks," *Neural Networks*, vol. 122, pp. 253–272, 2020.

[16] A. Shrestha, H. Fang, Q. Wu, and Q. Qiu, "Approximating backpropagation for a biologically plausible local learning rule in spiking neural networks," in *Proceedings of the International Conference on Neuromorphic Systems*, 2019, pp. 1–8.

[17] J. Kaiser, H. Mostafa, and E. Neftci, "Synaptic plasticity dynamics for deep continuous local learning (decolle)," *Frontiers in Neuroscience*, vol. 14, p. 424, 2020.

[18] H. Gao, J. He, H. Wang, T. Wang, Z. Zhong, J. Yu, Y. Wang, M. Tian, and C. Shi, "High-accuracy deep ann-to-snn conversion using quantization-aware training framework and calcium-gated bipolar leaky integrate and fire neuron," *Frontiers in Neuroscience*, vol. 17, 2023.

[19] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," *Frontiers in neuroscience*, vol. 13, p. 95, 2019.

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

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

[22] F. Zenke and S. Ganguli, "Superspike: Supervised learning in multilayer spiking neural networks," *Neural computation*, vol. 30, no. 6, 2018.

[23] M. Xiao, Q. Meng, Z. Zhang, D. He, and Z. Lin, "Online training through time for spiking neural networks," *Advances in Neural Information Processing Systems*, vol. 35, pp. 20 717–20 730, 2022.

[24] E. O. Neftci, H. Mostafa, and F. Zenke, "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks," *IEEE Signal Processing Magazine*, vol. 36, no. 6, pp. 51–63, 2019.

[25] F. Zenke and T. P. Vogels, "The remarkable robustness of surrogate gradient learning for instilling complex function in spiking neural networks," *Neural computation*, vol. 33, no. 4, pp. 899–925, 2021.

[26] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *Ieee Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[27] H. Markram, W. Gerstner, and P. J. Sjöström, "Spike-timing-dependent plasticity: a comprehensive overview," *Frontiers in synaptic neuroscience*, vol. 4, p. 2, 2012.

[28] E. L. Bienenstock, L. N. Cooper, and P. W. Munro, "Theory for the development of neuron selectivity: orientation specificity and binocular interaction in visual cortex," *Journal of Neuroscience*, vol. 2, no. 1, 1982.

[29] E. Oja, "Simplified neuron model as a principal component analyzer," *Journal of mathematical biology*, vol. 15, pp. 267–273, 1982.

[30] E. M. Izhikevich and N. S. Desai, "Relating stdp to bcm," *Neural computation*, vol. 15, no. 7, pp. 1511–1523, 2003.

[31] E. Oja, "Unsupervised learning in neural computation," *Theoretical computer science*, vol. 287, no. 1, pp. 187–207, 2002.

[32] J. J. Wade, L. J. McDaid, J. A. Santos, and H. M. Sayers, "Swat: A spiking neural network training algorithm for classification problems," *IEEE Transactions on neural networks*, vol. 21, no. 11, 2010.

[33] S. R. Kheradpisheh, M. Ganjtabesh, S. J. Thorpe, and T. Masquelier, "Stdp-based spiking deep convolutional neural networks for object recognition," *Neural Networks*, vol. 99, pp. 56–67, 2018.

[34] N. Frémaux and W. Gerstner, "Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules," *Frontiers in neural circuits*, vol. 9, p. 85, 2016.

[35] M. Mozafari, S. R. Kheradpisheh, T. Masquelier, A. Nowzari-Dalini, and M. Ganjtabesh, "First-spike-based visual categorization using reward-modulated stdp," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 12, pp. 6178–6190, 2018.

[36] H. Fang, Y. Zeng, and F. Zhao, "Brain inspired sequences production by spiking neural networks with reward-modulated stdp," *Frontiers in Computational Neuroscience*, vol. 15, p. 612041, 2021.

[37] F. M. Quintana, F. Perez-Peña, P. L. Galindo, E. O. Netfci, E. Chicca, and L. Khacef, "Etlp: Event-based three-factor local plasticity for online learning with neuromorphic hardware," *arXiv:2301.08281*, 2023.

[38] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature communications*, vol. 7, no. 1, p. 13276, 2016.

[39] H. Mostafa, V. Ramesh, and G. Cauwenberghs, "Deep supervised learning using local errors," *Frontiers in neuroscience*, vol. 12, p. 608, 2018.

[40] S. Bartunov, A. Santoro, B. Richards, L. Marris, G. E. Hinton, and T. Lillicrap, "Assessing the scalability of biologically-motivated deep learning algorithms and architectures," *Advances in neural information processing systems*, vol. 31, 2018.

[41] M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu, "Decoupled neural interfaces using synthetic gradients," in *International conference on machine learning*. PMLR, 2017, pp. 1627–1635.

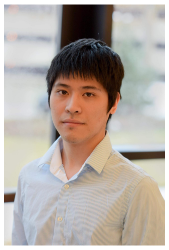
[42] B. Paillassa, B. Escrig, R. Dhaou, M.-L. Boucheret, and C. Bes, "Improving satellite services with cooperative communications," *International Journal of Satellite Communications and Networking*, 2011.

[43] W. Han and M. Jochum, "Latency analysis of large volume satellite data transmissions," in *2017 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, 2017, pp. 384–387.

[44] J. Chen, X. Di, R. Xu, H. Qi, L. Cong, K. Zhang, Z. Xing, X. He, W. Lei, and S. Zhang, "A remote sensing data transmission strategy based on the combination of satellite-ground link and geo relay under dynamic topology," *Future Generation Computer Systems*, vol. 145, 2023.

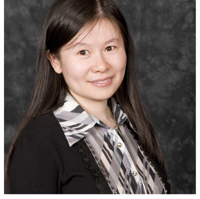
[45] J. Chen, X. Di, R. Xu, H. Luo, H. Qi, P. Zhan, and Y. Jiang, "An efficient scheme for in-orbit remote sensing image data retrieval," *Future Generation Computer Systems*, 2023.

RAMASHISH GAURAV is currently a Ph.D. candidate in the BRain Inspired Computing & Communications (BRICC) lab at the Bradley Department of Electrical and Computer Engineering in Virginia Tech. He is also a member of the Multifunctional Integrated Circuits and Systems (MICS) group at the Virginia Tech. He obtained his Integrated Dual Degree from IIT-BHU (Varanasi), India, and Masters of Applied Science from the University of Waterloo, Canada. His research interests are in Spiking Neural Networks, Neuromorphic Computing, AI & ML.



DUY ANH DO is currently pursuing a Doctor of Philosophy degree in the Bradley Department of Electrical and Computer Engineering at Virginia Tech. He completed his B.S. degree and his M.S. degree in 2020 and 2021 respectively, both from the Purdue University Fort Wayne. His research interests are in neural networks with focus on developing algorithms for multi-task reinforcement learning with theoretical guarantees.

THINH DOAN is an Assistant Professor in the Electrical and Computer Engineering Department at Virginia Tech. He obtained his Ph.D. degree in 2018 at the University of Illinois, Urbana-Champaign, his M.S. in 2013 at the University of Oklahoma, and his B.S. in 2008 at Hanoi University of Science and Technology, Vietnam, all in Electrical Engineering. His research interests span the intersection of control theory, optimization, machine learning, reinforcement learning, game theory, and applied probability theory.



YANG YI (Senior Member, IEEE) is an Associate Professor in the Bradley Department of Electrical and Computer Engineering at the Virginia Tech. Her research interests include Very Large Scale Integrated (VLSI) circuits and systems, Computer-Aided Design (CAD), Neuromorphic Computing and Architectures for brain-inspired computing systems, and low-power circuits design with advanced nano-technologies for high-speed wireless systems. She leads the BRain Inspired Computing & Circuits (BRICC) lab, and is the Director of the Multifunctional Integrated Circuits and Systems (MICS) group at the Virginia Tech; she is also a member of the Wireless@VT group.