Synaptic Plasticity Dynamics for Deep Continuous Local Learning (DECOLLE)

Jacques Kaiser,
FZI Research Center For Information Technology
Karlsruhe, Germany

Hesham Mostafa,
Department of Bioengineering
University of California San Diego
La Jolla, USA

Emre Neftci,
Department of Cognitive Sciences
Department of Computer Science
University of California Irvine, Irvine, USA

May 22, 2020

Abstract

A growing body of work underlines striking similarities between biological neural networks and recurrent, binary neural networks. A relatively smaller body of work, however, addresses the similarities between learning dynamics employed in deep artificial neural networks and synaptic plasticity in spiking neural networks. The challenge preventing this is largely caused by the discrepancy between the dynamical properties of synaptic plasticity and the requirements for gradient backpropagation. Learning algorithms that approximate gradient backpropagation using local error functions can overcome this challenge. Here, we introduce Deep Continuous Local Learning (DECOLLE), a spiking neural network equipped with local error functions for online learning with no memory overhead for computing gradients. DECOLLE is capable of learning deep spatiotemporal representations from spikes relying solely on local information, making it compatible with neurobiology and neuromorphic hardware. Synaptic plasticity rules are derived systematically from user-defined cost functions and neural dynamics by leveraging existing autodifferentiation methods of machine learning frameworks. We benchmark our approach on the event-based neuromorphic dataset N-MNIST and DvsGesture, on which DECOLLE performs comparably to the state-of-the-art. DECOLLE networks provide continuously learning machines that are relevant to biology and supportive of event-based, low-power computer vision architectures matching the accuracies of conventional computers on tasks where temporal precision and speed are essential.

1 Introduction

Understanding how the plasticity dynamics in multilayer biological neural networks are organized for efficient data-driven learning is a long-standing question in computational neurosciences [50, 45, 12]. The generally unmatched success of deep learning algorithms in a wide variety of data-driven tasks prompts the question of whether the ingredients of their success are compatible with their biological counterparts, namely Spiking Neural Networks (SNNs). Biological neural networks distinguish themselves from Artificial Neural Networks (ANNs) by their continuous-time dynamics, the locality of their operations [2], and their spike(event)-based communication. Taking these properties into account in a neural network is challenging, as the spiking nature of the neurons' nonlinearity makes it non-differentiable, the continuous-time dynamics raise a temporal credit assignment problem and the assumption of computations being local to the neuron disqualifies the use of Back-Propagation-Through-Time (BPTT).

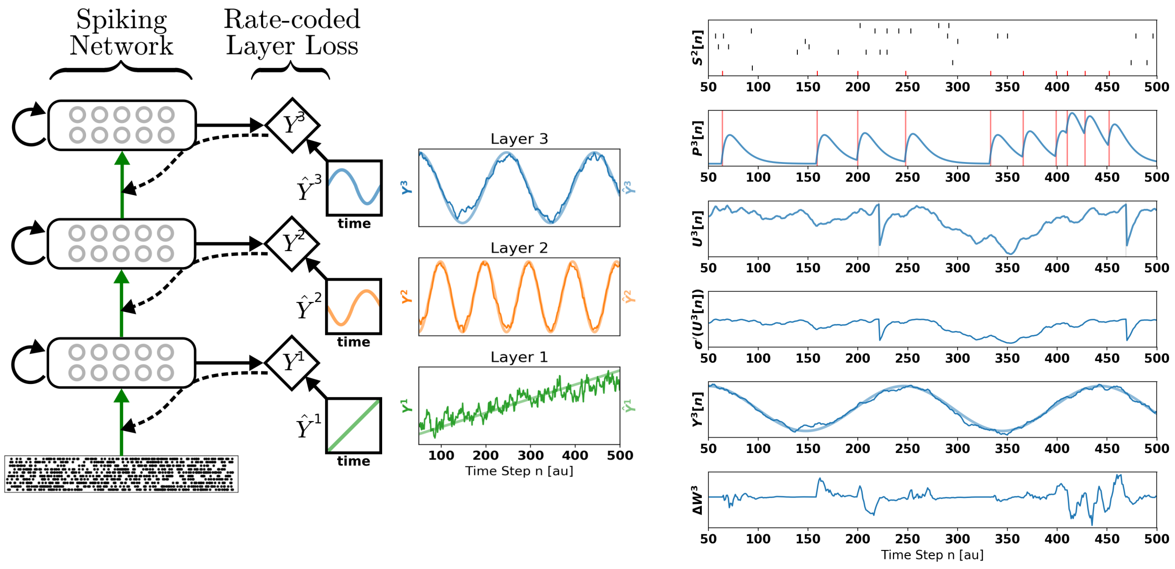

Figure 1: Deep Continuous Local Learning (DECOLLE). (Left) Each layer consists of spiking neurons with continuous dynamics. Each layer feeds into a local readout units through fixed, random connections (diamond-shaped, y). The local layer is trained such that the readout at each layer produce auxiliary targets $\hat{Y}$. Errors are propagated through the random connections to train weights coming into the spiking layer, but no further (curvy, dashed line). To simplify the learning rule and enable linear scaling of the computations, the cost function is a function of the states in the same time step. The state of the spiking neurons (membrane potential, synaptic states, refractory state) is carried forward in time. Consequently, even in the absence of recurrent connections, the neurons are stateful in the sense of recurrent neural networks. (Right) Snapshot of the neural states illustrating the DECOLLE learning rule in the top layer. In this example, the network is trained to produce three time-varying pseudo-targets $\hat{Y}^1$, $\hat{Y}^2$, $\hat{Y}^3$.

In this article, we describe DECOLLE, a SNN model with plasticity dynamics that solves the three problems above, and that performs at proficiencies comparable to that of multilayer neural networks. DECOLLE uses layerwise local readouts [35], which enables gradients to be computed locally (Fig. 1). To tackle the temporal dynamics of the neurons, we use a recently established equivalence between SNNs and recurrent ANNs [37]. This equivalence rests on a computational graph of the SNN, which can be implemented with standard machine learning frameworks as a recurrent neural network. Unlike BPTT and like Real-Time Recurrent Learning (RTRL)[48], DECOLLE is formulated in a way that the information necessary to compute the gradient is propagated forward, making the plasticity rule temporally local. Existing rules of this sort require dedicated state variables for every synapse, thus scaling at least quadratically with the number of neurons [48, 50]. In contrast, DECOLLE scales linearly with the number of neurons. This is achieved using a spatially and temporally local cost function reminiscent of readout mechanisms used in liquid state machines [33], but where the readout is performed over a fixed random combination of the neuron outputs. Our approach can be viewed as a type of synthetic gradient, a technique used to decouple one or more layers from the rest of the network to prevent layerwise locking [23]. Although synthetic gradients usually involve an outer loop that is equivalent to a full Back-Propagation (BP) through the network, DECOLLE instead relies on the random initialization of the local readout and forgoes the outer loop.

Conveniently, DECOLLE can leverage existing autodifferentiation tools of modern machine learning frameworks. Its linear scalability enables the training of hundreds of thousands of spiking neurons on a single GPU, and continual learning on very fine time scales. We demonstrate our approach on the classification of gestures, the IBM DvsGesture dataset [1], recorded using an event-based neuromorphic sensor and report comparable performance to deep neural networks and even networks trained with BPTT.

1.1 Related Work

Previous work demonstrated learning in multiple layers of SNN using feedback alignment [38, 32], performing at about 2% classification error on MNIST. However, those networks operated in the firing rate regime, by using either large populations or slow dynamics. In those works, training was not insensitive to the temporal dynamics of the neurons. The need for temporal dynamics are often obfuscated by the static nature of the benchmarked problems (*e.g.* MNIST), and a long readout interval that allows to ignore initial transients caused by the dynamics. In our previous work [38], ignoring temporal dynamics raised a "loop duration" problem, *i.e.* that the errors are available only after they have propagated through the network. This introduces latency or requires additional buffers for storing intermediate neural states. In traditional deep learning, the loop duration manifests itself as "layerwise locking", during which a layer's weights cannot be updated until a global cost function is evaluated [23]. This causes underutilization of the computing resources and a slowdown in learning. Besides the loop duration problem, multilayer networks trained with feedback alignment cannot reach the performances of gradient BP, especially with deeper networks ($\geq 30\%$ accuracy drop on ImageNet compared to backpropagation [4]).

The complex dynamics of spiking neurons is an important feature that can be exploited for learning spatiotemporal patterns. In a single layer of neurons, this feature can be leveraged using gradient descent, since it is applicable to the subthreshold dynamics of leaky Integrate & Fire (I&F) neurons [19, 8]. Because the I&F neuron output is non-differentiable, however, the application of these approaches to muliple layers is not straightforward. To deal with this problem, SuperSpike uses a surrogate network with differentiable activation functions to compute an approximate gradient [50]. The authors show that this learning rule is equivalent to a forward-propagation of errors using synaptic traces, and is capable of learning in hidden layers of feedforward multilayer networks.

Because the traces need to be computed for every trainable parameter, Superspike scales temporally and spatially as $O(N^2)$, where N is the number of neurons. While the complex biochemical processes at the synapse could account for the quadratic scaling, it prevents an efficient implementation in available hardware. Like SuperSpike, DECOLLE uses surrogate gradients to perform weight updates, but as discussed later, the cost function is local in time and space, such that only one trace per input neuron is required. This enables the algorithm to scale linearly in space. Furthermore, in DECOLLE the computation of the gradients can reuse the variables computed for the forward dynamics, such that learning has no additional memory overhead.

DECOLLE has some resemblance with reservoir networks, which are neural networks with fixed internal connectivity and trainable readout functions [24, 15, 33, 45]. The local readout in DECOLLE acts like a decoder layer in the flavor of the linear readouts in reservoir networks. In contrary to reservoir networks, DECOLLE learns the internal weights, but the readout weights are random and fixed. The training of the internal weights allows the network to learn representations that are easier to classify inputs for subsequent layers [35].

Spiking neural networks can be viewed as a subclass of binary, recurrent ANNs [37]. In the ANN sense, they are recurrent even when all the connections are feed-forward because the neurons maintain a state that is propagated forward at every time step. Binary neural networks, where both activations and/or weights are binary were studied in deep learning as a way to decrease model complexity during inference [13, 42]. BPTT for training SNNs was investigated in [8, 29, 43, 6, 21]. BPTT-based approaches provide an unbiased estimation of the gradients but at a cost in memory, because the entire sequence and resulting activity states are stored to compute gradients. Although the truncation of the sequences (as in truncated BPTT) can mitigate this problem, it is not adequate when discretizing continuous-time networks, such as the SNN [37] because the sequences can consists of hundreds of steps. This is because the time constants and simulation timestep in SNNs are such that the truncation window must be much larger. For SNN simulations with biological time constants, it is common to use simulation time steps $\Delta t \leq 1\mathrm{ms}$. Smaller time steps capture non-linear dynamics more accurately and determine the temporal precision of all produced spike times. Assuming $\Delta t = 1\mathrm{ms}$ (as used in this work), and if relevant interactions occur at one second, this implies that the truncation window must be at about 1000 timesteps. This significantly increases the complexity of BPTT in SNNs. In practice, the size of SNN trainable by BPTT is severely limited by the available GPU memory [43]. As we explain later in this article, DECOLLE requires an order T less memory resources compared to BPTT, where T is the sequence length. Hence, DECOLLE networks are generally not memory-limited. Furthermore, DECOLLE can be formulated as a local, three-factor synaptic plasticity rule, and is thus amenable to implementation in dedicated, event-based (neuromorphic) hardware [14] and compatible with neurobiology.

Decoupled Neural Interfaces (DNI) were proposed to mitigate layerwise locking in training deep neural networks [23]. In DNIs, this decoupling is achieved using a synthetic gradient, a neural network that estimates the gradients for a portion of the network. In an inner loop, the network parameters are trained using the synthetic gradients, and in an outer loop, the synthetic gradient network parameters are trained using a full BP step. The gradient computed using local errors in DECOLLE described below can be viewed as a type of synthetic gradient, which ignores the outer loop to avoid a full BP step. Although ignoring the outer loop limits DECOLLE's adaptation of the features using errors from other layers, we find that the network performs at or above state-of-the-art accuracy on N-MNIST and DVS Gesture benchmark tasks.

A related method called E-prop was developed in parallel to DECOLLE [7]. The resulting learning rule in E-prop is of the same form as Superspike and DECOLLE. E-prop uses adaptive spiking Long Short Term Memory (LSTM) neurons to maintain a longer term memory. This generalization allows to solve tasks with long-term dependencies (similar to LSTMs) but requires maintaining one trace per synapse. These memory requirements quickly exceed the capabilities of modern GPUs, especially when applied to convolutional neural networks. Even in neuromorphic hardware, maintaining a synapse-specific trace can incur a prohibitive cost in area and power [20, 14]. In DECOLLE, we focus on networks which do not incur any memory overhead for training, allowing to tractably train large networks.

This work builds on a combination of how gradients are dynamically computed in SuperSpike and local errors. We show in the methods section that this combination considerably reduces the computational requirements compared to a computing a global loss.

2 Methods

2.1 Neuron and Synapse Model

The neuron and synapse models used in this work follow leaky, current-based I&F dynamics with a relative refractory mechanism. The dynamics of the membrane potential U_i of a neuron i is governed by the following differential equations:

$$\begin{aligned} U_i(t) =& V_i(t) - \rho R_i(t) + b_i, \\ \tau_{mem}\frac{\mathrm{d}}{\mathrm{d}t}V_i(t) =& - V_i(t) + I_i(t), \\ \tau_{ref}\frac{\mathrm{d}}{\mathrm{d}t}R_i(t) =& - R_i(t) + S_i(t), \end{aligned} \tag{1}$$

with $S_i(t)$ the binary value (0 or 1) representing whether neuron i spiked at time t. The separation of the membrane potential into two variables U and V is done here for implementations reasons only. Biologically, the two states can be interpreted as a special case of a two-compartment model, consisting of one dendritic (V) and one somatic (U) compartment [18, Chap. 6.4]. The absence of dynamics for U can be interpreted as the special case when somatic capacitance is much smaller than the distal capacitance. A spike is emitted when the membrane potential reaches a threshold $S_i(t) = \Theta(U_i(t))$, where $\Theta(x) = 0$ if $x < 0$, otherwise 1 is the unit step function. The constant b_i represents the intrinsic excitability of the neuron. The refractory mechanism is captured with the dynamics of R_i: the neuron inhibits itself after firing, by a constant weight ρ. In contrast to standard I&F refractory mechanisms, a strong enough input can still induce the neuron to fire immediately after a spike. The factors τ_{ref} and τ_{mem} are time constants of the membrane and refractory dynamics, respectively. I_i denotes the total synaptic current of neuron i, expressed as:

$$\tau_{syn}\frac{\mathrm{d}}{\mathrm{d}t}I_i(t) = - I_i(t) + \sum_{j\in\text{pre}} W_{ij}S_j(t), \tag{2}$$

where W_{ij} is the synaptic weights between pre-synaptic neuron j and post-synaptic neuron i. Because V_i and I_i are linear with respect to the weights W_{ij}, The dynamics of V_i can be rewritten as:

$$\begin{aligned} V_i(t) =& \sum_{j\in\text{pre}} W_{ij}P_{ij}(t), \\ \tau_{mem}\frac{\mathrm{d}}{\mathrm{d}t}P_{ij}(t) =& - P_{ij}(t) + Q_{ij}(t), \\ \tau_{syn}\frac{\mathrm{d}}{\mathrm{d}t}Q_{ij}(t) =& - Q_{ij}(t) + S_j(t). \end{aligned} \tag{3}$$

The states P and Q describe the traces of the membrane and the current-based synapse, respectively. For each incoming spike, the trace Q undergoes a jump of height 1 and otherwise decays exponentially with a time constant τ_{syn}. Weighting the trace Q_{ij} with the synaptic weight W_{ij} results in the Post–Synaptic Potentials (PSPs) of neuron i caused by input neuron j.

All efferent synapses with identical time constants have identical dynamics. By linearity of P and Q, the state of the synapse can be described by a single synaptic variable per pre-synaptic neuron [10]. In the equation above, this is evident by the fact that P_{ij} and Q_{ij} are only driven by S_j, and so the index i can be dropped. This results in as many P and Q variables as there are pre-synaptic neurons, independently of the number of synapses. This strategy is commonly used in synapse circuits in neuromorphic hardware to reduce circuit area [3], and in software simulations of spiking neurons to improve memory consumption and computation time [10].

Discrete Spike Response Model of the Neuron and Synapse Dynamics

Because a computer will be used to simulate the dynamics, the dynamics are simulated in discrete time. We denote the simulation time step with Δt. We also make the layerwise organization of the network apparent with the superscript l denoting the layer to which the neuron belongs. The dynamical equations in Eq. (1) and Eq. (3) are expressed in discrete time as:

$$\begin{aligned} U_i^l[t] &= \sum_j W_{ij}^l P_j^l[t] - \rho R_i^l[t] + b_i^l, & P_j^l[t+\Delta t] &= \alpha P_j^l[t] + (1-\alpha) Q_j^l[t], \\ & & Q_j^l[t+\Delta t] &= \beta Q_j^l[t] + (1-\beta) S_j^{l-1}[t], \\ S_i^l[t] &= \Theta(U_i^l[t]), & R_i^l[t+\Delta t] &= \gamma R_i^l[t] + (1-\gamma) S_i^l[t], \end{aligned} \tag{4}$$

where the constants $\alpha = \exp(-\frac{\Delta t}{\tau_{\mathrm{mem}}})$, $\gamma = \exp(-\frac{\Delta t}{\tau_{\mathrm{ref}}})$ and $\beta = \exp(-\frac{\Delta t}{\tau_{\mathrm{syn}}})$ reflect the decay dynamics of the membrane potential U, the refractory state R and the synaptic state Q during a Δt timestep. Note that Eq. (4) is equivalent to a discrete-time version of the Spike Response Model (SRM$_0$) with linear filters [17].

2.2 Deep Learning with Local Losses

Loss functions are almost always defined using the network output at the top layer. Assuming a global cost function $\mathcal{L}(S^N)$ defined on the spikes S^N of the top layer and targets $\hat{Y}$, the gradients with respect to the weights in layer l are:

$$\frac{\partial \mathcal{L}(S^N)}{\partial W_{ij}^l} = \frac{\partial \mathcal{L}(S^N)}{\partial S_i^l} \frac{\partial S_i^l}{\partial U_i^l} \frac{\partial U_i^l}{\partial W_{ij}^l}. \tag{5}$$

The factor $\frac{\partial \mathcal{L}(S^N)}{\partial S_i^l}$ captures the backpropagated errors, *i.e.* how changing the output of neuron i in layer l modifies the global loss. This problem is known as the credit assignment problem. It generally involves non-local terms, including the activity of other neurons, their errors, and their temporal history. Thus, using local information only, a neuron in a deep layer cannot infer how a change in its activity will affect the top-layer cost. An increasing body of work is showing that approximations to the backpropagated errors in SNNs can allow local learning, for example in feedback alignment [31]. However, maintaining the history of the dynamics efficiently remains a challenging and open problem. While it is possible to use BPTT methods to compute these errors, this comes at a significant cost in memory and computation [48], and is not consistent with the constraint of local information.

We address this conundrum using deep local learning [35]. We focus on a form of deep local learning that attaches random readouts to deep layers and defines auxiliary cost functions over the readout. These auxiliary cost functions provide a task-relevant source of error for neurons in deep layers. The random readout is obtained by multiplying the neural activations with a random and fixed matrix. Training deep layers using auxiliary local errors that minimize the cost locally still allows the network as a whole to reach a small top-layer cost. As explained in [35], minimizing a local readout's classification loss puts pressure on deep layers to learn useful task-relevant features, which allow the random local classifiers to solve the task. Moreover, each layer builds on the features of the previous layer to learn features that are further disentangled with respect to the categories for its local random classifier compared to the previous layer. Thus, even though no error information propagates downwards through the layer stack, the layers indirectly learn useful hierarchical features that end up minimizing the cost at the top layer. Although the reasons for the effectiveness of local errors in deep network is intriguing and merits further work, it is orthogonal to the scope of this article. In this article, we focus on the fact that, provided local loss functions, surrogate learning in deep spiking neural networks becomes particularly efficient.

2.3 Deep Continuous Local Learning (DECOLLE)

As discussed above, in DECOLLE, we attach a random readout to each of the N layers of spiking neurons:

$$Y_i^l = \sum_j G_{ij}^l S_j^l,$$

where G_{ij}^l are fixed, random matrices (one for each layer l) and Θ is an activation function. The global loss function is then defined as the sum of the layerwise loss functions defined on the random readouts, *i.e.* $\mathcal{L} = \sum_{l=1}^N L^l(Y^l)$. To enforce locality, DECOLLE sets to zero all non-local gradients, *i.e.* $\frac{\partial L^l}{\partial W_{ij}^m} = 0$ if $m \neq l$. With this assumption, the weight updates at each layer become:

$$\Delta W_{ij}^l = -\eta \frac{\partial L^l}{\partial W_{ij}^l} = -\eta \frac{\partial L^l}{\partial S_i^l} \frac{\partial S_i^l}{\partial W_{ij}^l}, \tag{6}$$

where η is the learning rate. Assuming the loss function depends only on variables in same time step, the first gradient term on the right hand side, $\frac{\partial L^l}{\partial S_i^l}$, can be trivially computed using the chain rule of derivatives. Applying the chain of derivatives to the second gradient term yields:

$$\frac{\partial S_i^l}{\partial W_{ij}^l} = \frac{\partial \Theta(U_i^l)}{\partial U_i^l} \frac{\partial U_i^l}{\partial W_{ij}^l}.$$

Due to the sparse, binary activation of spiking neurons, this expression vanishes everywhere except at 0, where it is infinite [37]. To solve this problem, parameter updates in DECOLLE are based on a differentiable but slightly different version of the task-performing network. This approach was previously described as surrogate gradient-based learning [37, 50]:

$$\frac{\partial S_i^l}{\partial W_{ij}^l} = \sigma'(U_i^l) \frac{\partial U_i^l}{\partial W_{ij}^l},$$

where $\sigma'(U_i^l)$ is the surrogate gradient of the non-differentiable step function $\Theta(U_i^l)$. The rightmost term is computed as:

$$\frac{\partial U_i^l}{\partial W_{ij}^l} = P_j^l - \rho \frac{\partial R_i^l}{\partial W_{ij}^l}.$$

The terms involving R_i^l are difficult to calculate because they depend on the spiking history of the neuron. As in Superspike, we ignore these dependencies and use regularization to favor low firing rates, a regime in which the R_i^l has a negligible effect on the membrane dynamics. Putting all three terms together, we obtain the DECOLLE rule governing the synaptic weight update:

$$\Delta W_{ij}^l = -\eta \frac{\partial L^l}{\partial S_i^l} \sigma'(U_i^l) P_j^l. \tag{7}$$

In the special case of the Mean Square Error (MSE) loss for layer l, described as

$$L^l = \frac{1}{2} \sum_i \left(Y_i^l - \hat{Y}_i^l \right)^2,$$

the DECOLLE rule becomes

$$\begin{aligned} \Delta W_{ij}^l &= -\eta \, \mathrm{error}_i^l \, \sigma'(U_i^l) P_j^l, \\ \mathrm{error}_i^l &= \sum_j G_{ki}^l (Y_k^l - \hat{Y}_k^l), \end{aligned} \tag{8}$$

where $\hat{Y}^l$ is the pseudo-target for layer l.

Memory complexity of DECOLLE: The variables P and U required for learning are local and readily available from the forward dynamics. Because the errors are computed locally to each layer, DECOLLE does not need to store any additional intermediate variables, *i.e.* there is no space requirement for the parameter update computation. The same neural traces P and Q maintained during the forward pass are sufficient (see Sec. 2.4). The computational cost of the weight update is the same as the Widrow-Hoff rule (one addition and two products per connection, see Eq. (8)). This makes DECOLLE significantly cheaper to implement compared to BPTT for training SNN, *e.g.* SLAYER [43] which scales spatially as $O(NT)$, where T is the number of timesteps (see Appendix Sec. 5.2 for details on scaling).

Sign-Concordant Feedback Alignment in the Local Layers The gradients of the local losses L_i^l involve backpropagation through the local random projection Y^l. This is a non-local operation as it requires the symmetric transpose of the matrix G. This raises a weight transport problem, whereby the synaptic weight must be "transported" from one neuron to another. In a von Neumann computer, this is not a problem since memory is shared across processes. However, if memory is local, then a dedicated process must transmit this information. Feedback alignment in non-spiking networks was demonstrated to overcome this problem at a cost in accuracy [35]. In our experiments, we use sign-concordant feedback weights to compute the gradients of the local losses: the backward weights have the same sign as the forward ones, but subject to fixed multiplicative Gaussian noise. The noise here reflects the fact that weights do not need to be exactly symmetric. This assumption is the most plausible scenario in mixed-signal neuromorphic devices, where connections can be programmed with the same sign bidirectionally, but the effective weights are subject to fabrication mismatch [36]. Since the weights in the local readouts are fixed, there is no weight transport problem during learning. Thus, the computation of the errors can be carried out using another random matrix H^l [32] whose elements are equal to $H_{ij}^l = G_{ij}^{l,T} \omega_{ij}^l$ with a Gaussian distributed $\omega_{ij}^l \sim N(1, \frac{1}{2})$. To enforce sign-concordance, all values ω_{ij}^l below zero were set to zero.

Biological Plausibility of DECOLLE and Suitability for Neuromorphic Hardware: Eq. (8) consists of three factors, one modulatory ($error_i$), one post-synaptic ($\sigma'(U_i^l)$) and one pre-synaptic (P_j^l). These types of rules are often termed three-factor rules, which have been shown to be consistent with biology [41], while being compatible with a wide number of unsupervised, supervised and reinforcement learning paradigms [47]. The terms P and Q represent neural and synaptic states that are readily available at the neuron. In our previous work and general experience, the shape of the surrogate function σ does not play a major role in DECOLLE[1]. The surrogate function σ can be a piecewise linear function [38], such that σ' becomes a boxcar function. This corresponds to a learning update that is gated by the post-synaptic membrane potential, and is reminiscent of membrane voltage-based rules, where spike-driven plasticity is induced only when membrane voltage is inside an eligibility window [9, 11].

In the derivation of the DECOLLE rule, we used an instantaneous readout function Y^l in the sense that it did not depend on states of the previous time step. In biology, this readout would be carried out by spiking neurons. This introduces a temporal dependency. As in SuperSpike, this temporal dependency significantly increases the complexity of the learning, and is costly to implement in neuromorphic hardware. One solution is to compute the errors using spiking neurons with dynamics faster than those of the hidden neurons. In mixed signal hardware, this can be achieved through fast membrane and synaptic time constants. In digital hardware this could be achieved using a dedicated logic block.

Regularization and Implementation Details From a technological point of view, SNNs are interesting when the spike rate is low as dedicated neuromorphic hardware can directly exploit this sparsity to reduce computations by the same factor [34, 14]. To ensure reasonable firing rates and prevent sustained firing, we use two regularizers. One keeps U below to the firing threshold on average, and one activity regularizer enforces a minimum firing rate in each layer. The final loss function is:

$$\mathcal{L}_g = \sum_l L^l + \lambda_1 \langle [U_i^l + 0.01]^+ \rangle_i + \lambda_2 [0.1 - \langle U_i^l \rangle_i]^+ \tag{9}$$

where $\langle \cdot \rangle_i$ denotes averaging over index i, $[\cdot]^+$ is a linear rectification, and λ_1, λ_2 are hyperparameters. The minimum firing rate regularization is included to prevent the layers becoming completely silent during the training. Our experiments used a piecewise linear surrogate activation function, such that its derivative becomes the boxcar function $\sigma'(x) = 1$ if $x \in [-.5, .5]$ and 0 otherwise.

[1] Conversely, the particular surrogate function is reported to play an important role in BPTT [6]. This is likely due the product of the gradient approximations carried across multiple layers. This in turn can cause vanishing or exploding gradients.

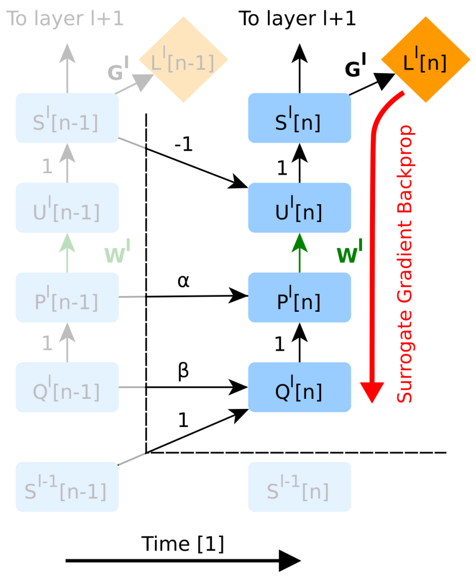

Figure 2: The unfolded computational graph of a feedforward SNN. Time flows to the right. Only temporal dependencies between timestep $n-1$ and n are shown here. Green edges indicate variables trained in the presented version of DECOLLE. Red edges indicate the flow of the gradients. Note that this graph is similar to that of a simple recurrent neural network. The forward RTRL approach combined with local errors means that errors do not propagate through neurons and across layers, as all the information required for learning is available at the layer and the current time step n. For implementation purposes however, autodifferentiation can be used to compute gradients within the neuron and time step (see Appendix Sec. 2.4 for details). To avoid clutter, the the node for R has been omitted.

In all our experiments, weight updates are made for each time step of the simulation. We use the AdaMax optimizer [26] with parameters $\beta_1 = 0$, $\beta_2 = 95$ and learning rate 10^{-9}, and a smooth L1 loss. Biases were used for all layers and trained in all DECOLLE layers. The weights G^l used for the local readouts were initialized uniformly. PyTorch code and a tutorial are publicly available on Github[2]. DECOLLE is simulated using mini-batches to leverage the GPU's parallelism.

2.4 Computational Graph and Implementation using Automatic Differentiation:

Perhaps one of the strongest advantages of DECOLLE is its out-of-the-box compatibility with Automatic Differentiation (AD) tools for implementing gradient BP. AD is a technology recently incorporated in machine learning frameworks to automatically compute gradients in a computational graph[3]. AD operates on the principle that all numerical computations are compositions of a finite set of elementary operations for which derivatives are known. By combining the derivatives of the operations through the chain rule of derivatives, the derivative of the overall composition can be computed in a single pass [5].

In practice, machine learning frameworks augment each elementary computation with its corresponding derivative function. As the desired operation is constructed, the dependencies with other variables are recorded as a computational graph. To perform gradient BP, after a forward pass, a backward pass computes all the derivatives of the operations in the graph. The root node of the reverse graph is typically a scalar loss function, and the leaf nodes are generally inputs and parameters of the network. After the backward pass, the gradients of all leaf nodes are applied to the trained parameters or inputs according to the optimization routine (*e.g.* Adam or similar).

SNNs being a special case of recurrent neural networks, it is possible to apply AD to the full graph [43]. On the other hand, DECOLLE only requires backpropagating through a subgraph corresponding to one layer and within the same time step (Fig. 2). This is because the information necessary for computing the gradients (P, Q, R, and U) is carried forward in time, and because local loss functions provide gradients for each layer.

AD in DECOLLE thus computes the gradients, locally, for each layer within each timestep. Because some operations in the subgraph can be non-differentiable (such as the spiking nonlinearity), we call this the "surrogate gradient backprop" (Fig. 2). This integration allows leveraging the layers, operations, optimizers and cost functions provided by the software. All experiments under the Experiments section use AD to compute derivatives. To prevent AD from unnecessarily backpropagating in time, we rely on special "stop-gradient" operations. In the appendix, we provide pseudocode and discussion of how this can be achieved.

[2]https://github.com/nmi-lab/decolle-public
[3]Gradient BP is a special case of reverse mode AD, see [5] for complete review

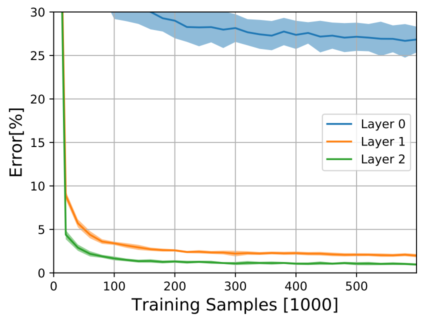

Figure 3: Classification results on the N-MNIST dataset for the three DECOLLE layers. Classification Error for the N-MNIST task during learning for all local errors associated with the convolutional layers. Shadings indicate standard deviation across the 10 runs.

3 Experiments

3.1 Regression with Poisson Spike Trains

To illustrate the inner workings of DECOLLE, we first demonstrate DECOLLE in a regression task. A three-layer fully connected network consisting of 512 neurons each is stimulated with a fixed 500ms Poisson spike train. Each layer in the network is trained with a different pseudo-target: $\hat{Y}^1$, a ramp function; $\hat{Y}^2$, a high-frequency sinusoidal function and $\hat{Y}^3$, a low-frequency sinusoidal function. (Fig. 1) illustrates the states of the neurons. For illustration purposes, the recording of the neural states was made in the absence of parameter updates (*i.e.* the learning rate is 0). The refractory mechanism decreases the membrane potential after the neuron spikes ($U[t]$). As discussed in the methods we use regularization on the membrane potential to keep the neurons from sustaining high firing rates and an activity regularizer to maintain a minimum firing rate. Updates to the weight are made at each time step and can be non-zero when the derivative of the activation function $\sigma'(U)$ and P are non-zero. The magnitude and direction of the update are determined by the error. Note that, in effect, the error is randomized as a consequence of the random local readout. The network learned to use the input spike times to reliably produce the targets.

3.2 N-MNIST

The N-MNIST dataset was recorded with a Dynamic Vision Sensor (DVS) [30] mounted on a pan-tilt unit performing microsaccadic motions in front of a screen displaying samples from the MNIST dataset [39]. Unlike standard imagers, the DVS records streams of events that signal the temporal intensity changes at each of its 128×128 pixels. For each of the 10 digits, we used 2000 samples for training and 100 samples for testing. The samples are cropped spatially from 34×34 to 32×32 and temporally to 300ms for both the train and test set. The network is simulated with a 1ms resolution – in other words, we sum up events in 1ms time bins. No further pre-processing is applied to the events.

Events are separated in two channels with respect to their polarity. A N-MNIST sample is therefore represented as a tensor of shape $300 \times 2 \times 32 \times 32$, stacked into mini-batch of 500 samples. The DECOLLE network is fed with 1ms slices of the input at a time. We relied on the same three-layer convolutional architecture used in the DvsGesture task described below. After a “burn-in” period of 50ms during which no update is made, gradient updates are performed at every simulation step. Hence, there are 250 weight updates per mini-batch. While the relevance of the time domain in N-MNIST is debatable [22], this experiment shows that the neural dynamics of our network leads to successful classifications in under 300ms.

The results on the N-MNIST dataset are shown in Figure 3. The experiment was performed 10 times with different random seeds. DECOLLE’s final error is $0.96\% \pm 0.12\%$ for the third layer with 600 000 training iterations. We note that, due to the large memory requirements, it is not practical to train the DECOLLE convolutional network using BPTT. Hence we cannot provide BPTT baselines.

Model	**Error**	**Training**	**Iterations**	**Ref.**
DECOLLE	$\mathbf{4.46 \pm 0.16\%}$	**Online**	**.16M**	This Work
SLAYER	6.36 ± 0.49 %	BPTT	.27M	[43]
C3D	$5.46 \pm 1.06\%$	BPTT	.32M	[46]
IBM EEDN	8.23% (5.51%)	BPTT	64M	[1]

Table 1: Classification error at the DvsGesture task. The IBM EEDN error in parentheses refers to the case with sliding window filter.

3.3 DvsGesture

We test DECOLLE at the more challenging task of learning gestures recorded using a DVS. Amir *et al.* recorded the DvsGesture dataset using a DVS, which comprises 1342 instances of a set of 11 hand and arm gestures, collected from 29 subjects under 3 different lighting conditions [1]. The unique features of each gesture are embedded in the stream of events. The event streams were downsized from 128 $\times$ 128 to 32 $\times$ 32 (events from 4 neighboring pixels were summed together as a common stream) and binned in frames of $1\mathrm{ms}$, the effective time step of the GPU-based simulation (Fig. 4). During training, a sample consisted of 500ms-long slices of the sample. To maximize the use of the dataset, the starting point of the slice was picked randomly, but such that a full 500ms sequence could be constructed. The sequences were presented to the network in mini-batches of 72 samples. Testing sequences were 1800ms-long, each starting from the beginning of each recording (288 testing sequences). Note that since the shortest recording in the test set is 1800ms, this duration was selected to simplify and speed up the evaluation. The classification is obtained by counting spikes at the output starting from a burn-in period of 50ms and selecting as output class the neuron that spiked the most. Test results from the DECOLLE network are reported with the dropout layer kept active, as this provided better results. Contrary to [1], we did not use stochastic decay and the neural network structure is a three-layer convolutional neural network, loosely adapted from [44]. We did not observe significant improvement by adding more than three convolutional layers. In shallow convolutional neural networks, it is common to use larger kernel sizes [27, 28]. Since the input sizes were 32x32, we used 7x7 kernel sizes in DECOLLE to ensure that the receptive field of neurons in the last layer covered the input. The optimal hyperparameters were found by a combination of manual and grid search. The learning rate was divided by 5 every 500 steps.

We compared with C3D and energy-efficient deep networks (EEDN). EEDN is a convolutional deep neural network architecture that can be trained offline (*e.g.* on a GPU) and deployed on the IBM TrueNorth chip [16]. EEDN was applied to DVS gestures and provides an important benchmark on this task [1]. Because EEDN was not designed to utilize the temporal dynamics of the spiking neurons in IBM TrueNorth chip, time is represented using the channel dimension of 2D convolutional networks. This approach limits the length of the sequence that EEDN can process. To overcome this, [1] used a sliding window filter. C3D is a 3D convolutional network commonly used for spatiotemporal classification in videos [46], where the dimensions are time, height and width. Using 3D kernels, C3C can learn spatiotemporal patterns. The network was similar to [46] except that is was adapted for 32x32 frames and using half of the features per layer (see Appendix for network layers). We note that the C3D network is deeper and wider than the DECOLLE network. We found that 16x32x32 frames, where each of the 16 representing 32ms slices of the DVS data performed best.

Overall, DECOLLE's performance is comparable or better than other published SNN implementations that use BP for training ((Tab. 1), (Fig. 4)) and close to much larger C3D networks. DECOLLE reached the reported accuracies after two orders of magnitude fewer iterations and smaller network compared to the IBM EEDN case (Tab. 1), [1]. Interestingly, the first layer of DECOLLE has a low classification accuracy. A similar effect is observed in non-spiking neural networks [35]. The local classifier errors improve for the second and third hidden layers compared to the first hidden layer. This is an indication that the network is able to make use of depth to obtain better accuracy. An examination of the filters learning in the first convolutional layer shows filters of varying frequencies and orientations (Fig. 5). Interstingly, the filters on the positive and negative channels of the DVS are similar, but exhibit small variations that are consistent with motion. This correlation is consistent with the DVS data, where leading edges of one polarity co-occur with trailing edges of opposite polarity.

Layer Type	#	Data Type	Dimensions
DVS	2	AEDAT 3.1	128 × 128
Downsample (Sum)	2	Integer	32 × 32
7 × 7 Conv	64	Float	30 × 30
2 × 2 MaxPool	64	Float	15 × 15
Spiking Non-linearity		Binary	
Dropout(p=.5)		Float	
Dense	11	Float	11
7 × 7 Conv	128	Float	13 × 13
Spiking Non-linearity		Binary	
Dropout(p=.5)		Binary	
Dense	11	Float	11
7 × 7 Conv	128	Float	11 × 11
2 × 2 MaxPool	128	Float	5 × 5
Spiking Non-linearity		Binary	
Dropout(p=.5)		Binary	
Dense	11	Float	11

Table 2: DECOLLE Neural network used for the DvsGesture dataset. Note that dense layers are used for the local classifiers only and were not fed to the subsequent convolutional layers. AEDAT 3.1 is a data format used for event-based data. The spiking nonlinearity was always applied after the pooling layers. Dropout layers were left active during testing.

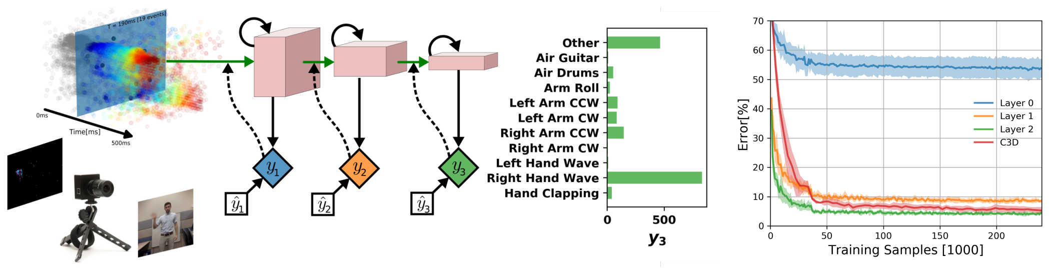

Figure 4: (Left) DECOLLE setup for DvsGesture recognition. Learning was performed on the dataset provided with [1] and consists of 11 gestures. The network consisted of three convolutional layers with max-pooling. A local classifier is attached to every layer and followed by dropout for regularization. DECOLLE is fed with 1ms integer frames. (Right) Classification Error for the DvsGesture task during learning for all local errors associated with the convolutional layers . Shadings indicate standard deviation across runs (5 runs for C3D, 10 runs for DECOLLE)

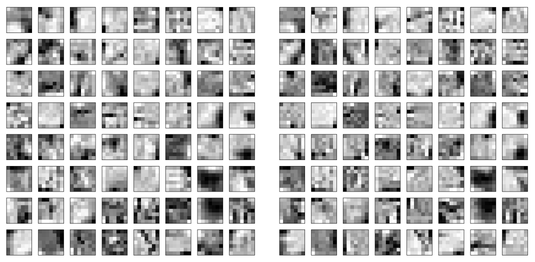

Figure 5: 7 × 7 Filters learned in the positive polarity channel (Left) and negative polarity channel (Right) of the first convolutional layer. The similarity of the kernels across the two polarities reflects the DVS data, where leading edges and trailing edges co-occur with opposite polarities.

4 Conclusion

Understanding and deriving neural and synaptic plasticity rules that can enable hidden weights to learn is an ongoing quest in neuroscience and neuromorphic engineering. From a machine learning perspective, locality and differentiability are key issues of the spiking neuron model operations. While the latter problem is now being tackled with surrogate gradient approaches, how to achieve this in deep networks in a scalable and local fashion is still an open question.

We presented a novel synaptic plasticity rule, DECOLLE, derived from a surrogate gradient approach with linear scaling in the number of neurons. The rule draws on recent work in surrogate gradient descent in spiking neurons and local learning with layerwise classifiers. The linear scalability is obtained through a (instantaneous) rate-based cost function on the local classifier. The simplicity of the DECOLLE rule equation makes it amenable for direct exploitation of existing machine learning software libraries. Thanks to the surrogate gradient approach, the updates computed through automatic differentiation are equal to the DECOLLE update. This enables the leveraging of a wide variety of machine learning frameworks for implementing online learning of SNNs.

Updates in DECOLLE are performed at every time step, in accordance with the continuity of the leaky I&F dynamics. This can lead to a large number of updates and inefficient implementations in hardware. To tackle this problem, updates can be made in an error-triggered fashion, as discussed in [40]. A direct consequence of the local classifiers is the lack of cross-layer adaptation of the layers. To tackle this problem, one could use meta-learning to adapt the random matrix in the classifier. In effect, the meta-learning loop would act as the outer loop in the synthetic gradients approach [23]. The notion that a "layer" of neurons specialized in solving certain problems and sensory modalities is natural in computational neurosciences and can open multiple investigation avenues for understanding learning and plasticity in the brain.

DECOLLE is a departure from standard SNNs trained with Hebbian spike-timing-dependent plasticity, as it uses a normative learning rule that is partially derived from first principles. Models of this type can make use of standard processors where it makes the most sense (i.e. readout, cost functions etc.) and neuromorphic dedicated hardware for the rest. Because it leverages the best of both worlds, DECOLLE is poised to make SNNs take off in event-based computer vision.

Acknowledgments

EN was supported by the Intel Corporation, the National Science Foundation under grant 1652159, and by the Korean Institute of Science and Technology. JK was supported by a scholarship within the FITweltweit programme of the German Academic Exchange Service (DAAD). HM was supported by the Swiss National Fund. JK, HM, EN wrote the paper, conceived the experiments. JK and EN ran the experiments and analyzed the data. The authors declare that this study received funding from Intel Corporation. The funder was not involved in the study design, collection, analysis, interpretation of data, the writing of this article or the decision to submit it for publication.

References

[1] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, et al. A low power, fully event-based gesture recognition system. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7243–7252, 2017.

[2] Pierre Baldi, Peter Sadowski, and Zhiqin Lu. Learning in the machine: The symmetries of the deep learning channel. *Neural Networks*, 95:110–133, 2017.

[3] C. Bartolozzi and G. Indiveri. Silicon synaptic homeostasis. In *Brain Inspired Cognitive Systems, BICS 2006*, pages 1–4, October 2006.

[4] Sergey Bartunov, Adam Santoro, Blake Richards, Luke Marris, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Advances in Neural Information Processing Systems*, pages 9368–9378, 2018.

[5] Atılım Günes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

[6] Guillaume Bellec, Darjan Salaj, Anand Subramoney, Robert Legenstein, and Wolfgang Maass. Long short-term memory and learning-to-learn in networks of spiking neurons. *arXiv preprint arXiv:1803.09574*, 2018.

[7] Guillaume Bellec, Franz Scherr, Elias Hajek, Darjan Salaj, Robert Legenstein, and Wolfgang Maass. Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets. *arXiv preprint arXiv:1901.09049*, 2019.

[8] Sander M Bohte, Joost N Kok, and Johannes A La Poutré. Spikeprop: backpropagation for networks of spiking neurons. In *ESANN*, pages 419–424, 2000.

[9] J. Brader, W. Senn, and S. Fusi. Learning real world stimuli in a neural network with spike-driven synaptic dynamics. *Neural Computation*, 19:2881–2912, 2007.

[10] R. Brette, M. Rudolph, T. Carnevale, M. Hines, D. Beeman, J.M. Bower, M. Diesmann, A. Morrison, P.H. Harris Jr. F.C. Goodman, M. Zirpe, T. Natschläger, D. Pecevski, B. Ermentrout, M. Djurfeldt, A. Lansner, O. Rochel, T. Vieville, E. Muller, A.P. Davison, S. El Boustani, and A. Destexhe. Simulation of networks of spiking neurons: A review of tools and strategies. *Journal of Computational Neuroscience*, 23(3):349–398, Dec 2007.

[11] E. Chicca, F. Stefanini, and G. Indiveri. Neuromorphic electronic circuits for building autonomous cognitive systems. *Proceedings of IEEE*, 2013.

[12] C. Clopath, L. Büsing, E. Vasilaki, and W. Gerstner. Connectivity reflects coding: a model of voltage-based stdp with homeostasis. *Nature Neuroscience*, 13(3):344–352, 2010.

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

[14] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, P. Joshi, A. Lines, A. Wild, and H. Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, PP(99):1–1, 2018.

[15] C. Eliasmith and C.H. Anderson. *Neural engineering: Computation, representation, and dynamics in neurobiological systems*. MIT Press, 2004.

[16] Steven K Esser, Paul A Merolla, John V Arthur, Andrew S Cassidy, Rathinakumar Appuswamy, Alexander Andreopoulos, David J Berg, Jeffrey L McKinstry, Timothy Melano, Davis R Barch, et al. Convolutional networks for fast, energy-efficient neuromorphic computing. *PNAS*, 113:11441–11446, 2016.

[17] W. Gerstner and W. Kistler. *Spiking Neuron Models. Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[18] Wulfram Gerstner, Werner M Kistler, Richard Naud, and Liam Paninski. *Neuronal dynamics: From single neurons to networks and models of cognition*. Cambridge University Press, 2014.

[19] R. Gütig and H. Sompolinsky. The tempotron: a neuron that learns spike timing–based decisions. *Nature Neuroscience*, 9:420–428, 2006.

[20] Frank L. Maldonado Huayaney, Stephen Nease, and Elisabetta Chicca. Learning in silicon beyond STDP: A neuromorphic implementation of multi-factor synaptic plasticity with calcium-based dynamics. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 63(12):2189–2199, dec 2016.

[21] Dongsung Huh and Terrence J Sejnowski. Gradient descent for spiking neural networks. *arXiv preprint arXiv:1706.04698*, 2017.

[22] Laxmi R Iyer, Yansong Chua, and Haizhou Li. Is neuromorphic mnist neuromorphic? analyzing the discriminative power of neuromorphic datasets in the time domain. *arXiv preprint arXiv:1807.01013*, 2018.

[23] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. *arXiv preprint arXiv:1608.05343*, 2016.

[24] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34, 2001.

[25] J. Kaiser, H. Mostafa, and E. Neftci. Synaptic plasticity for deep continuous local learning. *arXiv preprint arXiv:1812.10766*, Nov 2018.

[26] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[27] Jonas Kubilius, Martin Schrimpf, Ha Hong, Najib J Majaj, Rishi Rajalingham, Elias B Issa, Kohitij Kar, Pouya Bashivan, Jonathan Prescott-Roy, Kailyn Schmidt, et al. Brain-like object recognition with high-performing shallow recurrent anns. *arXiv preprint arXiv:1909.06161*, 2019.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[29] Jun Haeng Lee, Tobi Delbruck, and Michael Pfeiffer. Training deep spiking neural networks using backpropagation. *Frontiers in Neuroscience*, 10, 2016.

[30] P. Lichtsteiner, C. Posch, and T. Delbruck. An 128x128 120dB 15μs-latency temporal contrast vision sensor. *IEEE J. Solid State Circuits*, 43(2):566–576, 2008.

[31] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014.

[32] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7, 2016.

[33] W. Maass, T. Natschläger, and H. Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, 2002.

[34] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, et al. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.

[35] Hesham Mostafa, Vishwajith Ramesh, and Gert Cauwenberghs. Deep supervised learning using local errors. *arXiv preprint arXiv:1711.06756*, 2017.

[36] E. Neftci, E. Chicca, G. Indiveri, and R.J. Douglas. A systematic method for configuring VLSI networks of spiking neurons. *Neural Computation*, 23(10):2457–2497, Oct 2011.

[37] 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*, 36(6):51–63, Nov 2019.

[38] EO Neftci, C Augustine, S Paul, and Georgios Detorakis. Event-driven random back-propagation: Enabling neuromorphic deep learning machines. *Frontiers in Neuroscience*, 11:324, Jun 2017.

[39] Garrick Orchard, Ajinkya Jayawant, Gregory K. Cohen, and Nitish Thakor. Converting static image datasets to spiking neuromorphic datasets using saccades. *Frontiers in Neuroscience*, 9, nov 2015.

[40] M Payvand, M. Fouda, A Eltawil, F Kurdahi, and E. Neftci. Error-triggered three-factor learning dynamics for crossbar arrays. In *2020 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Mar 2020. (accepted).

[41] Jean-Pascal Pfister, Taro Toyoizumi, David Barber, and Wulfram Gerstner. Optimal spike-timing-dependent plasticity for precise action potential firing in supervised learning. *Neural computation*, 18(6):1318–1348, 2006.

[42] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

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

[44] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[45] David Sussillo and Larry F Abbott. Generating coherent patterns of activity from chaotic neural networks. *Neuron*, 63(4):544–557, 2009.

[46] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[47] Robert Urbanczik and Walter Senn. Learning by the dendritic prediction of somatic spiking. *Neuron*, 81(3):521–528, 2014.

[48] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[49] Ronald J Williams and David Zipser. Gradient-based learning algorithms for recurrent networks and their computational complexity. *Backpropagation: Theory, architectures, and applications*, 433, 1995.

[50] Friedemann Zenke and Surya Ganguli. Superspike: Supervised learning in multi-layer spiking neural networks. *arXiv preprint arXiv:1705.11146*, 2017.

5 Supplementary Information

5.1 Implementation of DECOLLE using Autodifferentiation

The equations of DECOLLE are very similar to that of a simple recurrent neural network. However, rather than performing backpropagation through-time, the derivatives of U_i are computed by propagating the traces P_i forward in time as follows:

```
for $n = 0...T$ do
  {Advance State}
  for $l < 1...L$ do
    $U_i^l = \sum_j W_{ij}^l P_j^l - \rho R_i^l$
    $S_i = STOPGRAD(\Theta(U_i) - \sigma(U_i)) + \sigma(U_i)$
    if Sign-concordant feedback matrix then
      $Y_i = STOPGRAD(\sum_j G_{ij} S_j - \sum_j H_{ij} S_j) + \sum_j H_{ij} S_j$
    else
      $Y_i = \sum_j G_{ij} S_j$
    end if
    $L^l = f_{loss}(Y_k^l, \hat{Y}_k^l)$
    $W_{ij}^l = W_{ij}^l + \eta GRAD(L^l, W_{ij}^l)$
    $P_i^l = \alpha P_i^l + (1 - \alpha_i) Q_i^l$
    $Q_i^l = \beta Q_i^l + (1 - \beta_i) STOPGRAD(S_i^{l-1})$
    $R_i^l = \gamma R_i^l + (1 - \gamma_i) STOPGRAD(S_i^{l-1})$
  end for
end for
```

where f_{loss} is the loss function, *e.g.* MSE loss. STOPGRAD prevents the flow of gradients by setting them to zero. STOPGRAD(A-B)+B as above is a common construct used to compute gradients using a separate subgraph. In this case, it is used to implement the surrogate gradient. Note that the time variable does not appear in the variables. This underlines that DECOLLE computation does not need to store the state histories, and that the variables necessary for computing the gradient (GRAD) are available within the same step. In our implementation, the weights are updated online, at every time step. The cost of making the parameter update is no more than accumulating the gradients at each time step, since parameter memory and dynamical state memory are the same. Therefore, there is no overhead in updating online. Note that this may not be the case in dedicated neuromorphic hardware or AI accelerators that require different memory structures for storing parameter memory.

5.2 Complexity Overhead for Various Spiking Neuron Gradient-Based Training Approaches

We provide additional detail on the complexity of DECOLLE compared to other learning methods. In the current implementation of DECOLLE, all weight updates are applied immediately, thus no additional memory is necessary to accumulate the gradients. In all other methods presented in (Tab. 3), the weight updates are applied in an epoch-wise fashion, which requires an additional variable to store the accumulated weights. However, this is an implementation choice which could have been made for methods other than DECOLLE as well. For this reason, the overhead of accumulating gradients in epoch-wise learning is ignored in the following calculations.

DECOLLE The state of P and Q must be maintained. These states are readily available from the forward pass, and therefore do not need to be stored specifically for learning. Space complexity is therefore $O(1)$. Each weight update requires MN_r multiplications to obtain M local errors. Each of these are multiplied by the number of inputs pN, resulting in $O(pNM + MN_r)$ time complexity, where p is the faction of connected neurons. Similarly to [35], the random weights in G^l can be computed using a random number generator, which requires one seed value per layer.

Superspike When using the Van Rossum Distance (VRD), the Superspike learning rule requires one trace per connection, resulting in a space complexity of $O(pNM)$. The additional complexity here compared to DECOLLE is caused by the additional filter in the Van Rossum distance. Note that if the learning is applied directly to membrane potentials, the space and time complexity is similar to that of DECOLLE.

eProp In the case when no future errors are used, the complexity of e-Prop [7] is similar to that of SuperSpike.

RTRL and BPTT The complexity of these techniques are discussed in detail in [49].

Method	Space	Time
DECOLLE	$O(1)$	$O(MN_r + pNM)$
SuperSpike (VRD)	$O(pNM)$	$O(pNM)$
e-Prop 1	$O(pNM)$	$O(pNM)$
RTRL	$O(pNM^2)$	$O(p^2N^2M^2)$
BPTT	$O(NT)$	$O(pNMT)$

Table 3: Complexity analysis of the gradient computation. N: Input neurons, M: Neurons in Layer, T: Length of Backpropagated Sequence, N_r: number of readout neurons in DECOLLE, p: ratio of connected neurons/total possible connections.

5.3 C3D Network

We used a standard 3D convolution network (C3D) for comparison with DECOLLE. In C3D, the temporal dimension is taken into account in the third dimension of the 3D convolution.

Layer Type	#	Data Type	Dimensions
DVS	2	AEDAT 3.1	128×128
Downsample (Sum) and Frame	2	Binary	$16 \times 32 \times 32$
3×3 Conv ReLU	32	Binary	$16 \times 32 \times 32$
$1 \times 2 \times 2$ MaxPool	32	Binary	$16 \times 16 \times 16$
3×3 Conv ReLU	64	Binary	$16 \times 16 \times 16$
$2 \times 2 \times 2$ MaxPool	64	Binary	$8 \times 8 \times 8$
3×3 Conv ReLU	128	Binary	$8 \times 8 \times 8$
3×3 Conv ReLU	128	Binary	$8 \times 8 \times 8$
$2 \times 2 \times 2$ MaxPool	128	Binary	$4 \times 4 \times 4$
3×3 Conv ReLU	256	Binary	$4 \times 4 \times 4$
3×3 Conv ReLU	256	Binary	$4 \times 4 \times 4$
$2 \times 2 \times 2$ MaxPool	256	Binary	$2 \times 2 \times 2$
3×3 Conv ReLU	256	Binary	$2 \times 2 \times 2$
3×3 Conv ReLU	256	Binary	$2 \times 2 \times 2$
$2 \times 2 \times 2$ MaxPool	256	Binary	$1 \times 2 \times 2$
Dense ReLU	1024	Float	1024
Dense ReLU	512	Float	512
Dense Softmax	11	Float	11