Efficient Event-based Delay Learning in Spiking Neural Networks

Balázs Mészáros[1,2], James C. Knight[1], and Thomas Nowotny[1]

[1] *Sussex AI, School of Engineering and Informatics, University of Sussex, Brighton, United Kingdom*
[2] *The Alan Turing Institute, London, United Kingdom*

Abstract

Spiking Neural Networks (SNNs) compute using sparse communication and are attracting increased attention as a more energy-efficient alternative to traditional Artificial Neural Networks (ANNs). While standard ANNs are stateless, spiking neurons are stateful and hence intrinsically recurrent, making them well-suited for spatio-temporal tasks. However, the duration of this intrinsic memory is limited by synaptic and membrane time constants. Delays are a powerful additional mechanism and, in this paper, we propose a novel event-based training method for SNNs with delays, grounded in the EventProp formalism which enables the calculation of exact gradients with respect to weights and delays. Our method supports multiple spikes per neuron and, to the best of our knowledge, is the first delay learning algorithm to be applied to recurrent SNNs. We evaluate our method on a simple sequence detection task, as well as the Yin-Yang, Spiking Heidelberg Digits, Spiking Speech Commands and Braille letter reading datasets, demonstrating that our algorithm can optimise delays from suboptimal initial conditions and enhance classification accuracy compared to architectures without delays. We also find that recurrent delays are particularly beneficial in small networks. Finally, we show that our approach uses less than half the memory of the current state-of-the-art delay-learning method and is up to 26× faster.

1 Introduction

Artificial Neural Networks (ANNs) have gained immense popularity and seen significant improvements over the past decade. However, commonly used models are very energy intensive [1], whereas the human brain has an estimated power budget of only 20 W [2]. It might, therefore, be beneficial to again turn to neuroscience for inspiration. One reason for the brain's better efficiency is that neurons in the brain transmit sparse binary events called spikes, while the units in ANNs typically communicate real-valued activation values densely in time and space. Spiking Neural Networks (SNNs) leverage the sparse communication patterns of biological neurons for machine learning (ML) and are particularly efficient on neuromorphic systems designed to provide efficient hardware platforms for brain-like computing [3, 4, 5]. Like ANNs, SNNs are universal function approximators, suggesting they could enable an energy-efficient future for ML. Therefore, researchers are increasingly focusing on training SNNs on popular ML tasks, for instance, in the fields of computer vision [6] and natural language processing [7]. Since individual spiking neurons rely on hidden temporal dynamics, they have 'implicit recurrence' so, even SNNs with feedforward architectures, have theoretical advantages for temporal processing over feedforward networks of stateless units. However, despite this potential, achieving or beating the performance of ANNs using SNNs remains a significant challenge.

The most commonly used training algorithm in ANNs is gradient descent. However, the non-differentiable transitions at spike times cause mathematical complications when calculating gradients in SNNs. To overcome this issue, some researchers do not train SNNs directly but instead train ANNs and then transfer the weights to an SNN for inference [8, 9]. However, because this approach typically uses spike counts to represent the activations of ANN units, it does not fully leverage the potential energy savings of sparse spiking in SNNs. While there are more efficient encoding alternatives [10], one time step in the ANN is still mapped to many timesteps in the SNN and the efficiency of SNNs is not leveraged at training time.

Another popular solution is to discretise the network dynamics and use Backpropagation Through Time (BPTT) with 'surrogate gradients' to smooth the threshold function in the backward pass [11, 12].

However, this method requires storing neuron state variables at every time step for the backward pass, meaning that memory requirements scale linearly with sequence length. This limits the maximum number of time steps in a trial to a few hundred. Furthermore, this method also does not exploit the increased efficiency of sparse spiking during the backward pass.

Bohte, Kok, and La Poutré [11] were the first to show how to calculate exact gradients in SNNs, by providing recursive relations for the gradient that can be implicitly computed using backpropagation. Alternatively, with some constraints on the time constants of neurons, analytic expressions for the time of the next spike of a leaky integrate-and-fire (LIF) can be derived and differentiated [13, 14], enabling the calculation of gradients. However, more generally, neurons in SNNs exhibit hybrid dynamics (a longstanding focus in optimal control theory [15]), combining continuous changes between spikes with discontinuous state transitions at spike times. The link between neural network training and optimal control has been well established [16], and the adjoint method – a staple of optimal control – has been used to derive gradients for smoothed spiking neuron models without reset [17]. Also using the adjoint method, Wunderlich and Pehle [18] developed the EventProp algorithm for calculating exact gradients in SNNs of integrate-and-fire neurons and 'exponential synapses'. The backward pass in EventProp combines a system of ordinary differential equations for the adjoint variables of the neuron dynamics with purely event-based backward transmission of error signals at spike times, making the best use of the hybrid nature of SNNs. Wunderlich and Pehle tested their method on latency-encoded MNIST [19] and the Yin-Yang datasets [20]. More recently, Nowotny, Turner, and Knight [21] extended EventProp to the more challenging benchmarks of the Spiking Heidelberg Digits (SHD) and Spiking Speech Commands [22], using loss shaping to overcome issues caused by exact gradients not containing information about spike creation and deletion. The time and space complexity of EventProp enables very efficient GPU training of large models on long sequences [21], hardware-in-the-loop training [23], and even training on neuromorphic hardware [24].

Spiking neurons' implicit recurrence is characterised by temporal parameters such as the membrane and synaptic time constants. Research has shown that optimising these [25, 26] can enhance network performance. Delays are another mechanism for temporal processing, and recent work indicates the utility of learnable delays for temporal tasks [27, 28], In biological neural networks, synaptic delays arise naturally due to the spatial structure of the network and can be modified to facilitate coincidence detection [29] and learning [30]. From a computational perspective, the inclusion of delays has been shown to significantly increase network capacity [31] and Maass and Schmitt [32] demonstrated that an SNN with k adjustable delays can compute a much richer class of functions than a network with k adjustable weights. Furthermore, neuromorphic systems such as SpiNNaker [33] and Loihi [34] are specifically designed to accommodate synaptic delays, so that SNNs with delays can still be efficiently deployed. Adding delays to SNNs has recently gained popularity, with several models treating them as learnable parameters and obtaining state-of-the-art performance on classification tasks [27, 28, 35]. Grappolini and Subramoney [36] even showed that networks can be trained to comparable performance with pure synaptic delay learning. However, most of these methods are based on surrogate gradients [27, 37, 28], which do not allow the event-based nature of SNNs to be exploited at training time, and some use temporal convolutions to implement delays [27, 38], which results in large overheads in memory and computation. DelGrad [39] was the first exact gradient-based delay learning method, but so far it has only been implemented in feedforward networks where each neuron only emits one spike per trial. No prior work has implemented delay learning for recurrent connections.

Here, we extend EventProp to incorporate heterogeneous and learnable delays and implement our extended version in mlGeNN [40, 41] – a spike-based ML library built on the GPU-optimized GeNN simulator [42, 43, 44]. GeNN generates GPU kernels for efficiently simulating networks of neurons which communicate with sparse events using a *hybrid* simulation strategy where the forward and backward dynamics of each neuron are updated every timestep, but synapses are only updated in timesteps where their presynaptic neurons produce a spike. This hybrid strategy differs from the purely time-driven approach (typically used to implement SNNs with standard ML libraries), where neurons and synapses are updated every timestep and also from purely event-based simulators where both neurons and synapses are only updated at spike times. All three approaches have advantages and disadvantages. While a purely timestep-based approach is inherently wasteful, standard ML libraries counteract the inherent inefficiency with highly GPU-optimised matrix multiplication routines to multiply neuron outputs with weights. Pure event-based simulators make efficient use of the event-based nature of SNNs and produce precise spike

Free dynamics	**Transition condition**	**Jumps at transition**
Forward:		
$\tau_\mathrm{m}\dot{V} = -V + I$	$(V^-)_{n(k)}\Big\vert_{t_k^\mathrm{spike}} - \vartheta = 0,$	$(V^+)_{n(k)}\Big\vert_{t_k^\mathrm{spike}} = 0$
$\tau_\mathrm{s}\dot{I} = -I$	$(\dot{V})_{n(k)}\Big\vert_{t_k^\mathrm{spike}} \neq 0$	$(I^+)_m\Big\vert_{t_k^\mathrm{spike}+d_{mn(k)}} = (I^-)_m\Big\vert_{t_k^\mathrm{spike}+d_{mn(k)}} + w_{mn(k)}e_{n(k)}$
Backward:		
$\tau_\mathrm{m}\lambda_V' = -\lambda_V - \frac{\partial l_V}{\partial V},$	$t - t_k^\mathrm{spike} = 0,$	$(\lambda_V^-)_{n(k)} = \left[\frac{(\dot{V}^+)_{n(k)}}{(\dot{V}^-)_{n(k)}}(\lambda_V^+)_{n(k)} + \frac{1}{\tau_\mathrm{m}(\dot{V}^-)_{n(k)}}\left[\frac{\partial l_p}{\partial t_k^\mathrm{spike}} + l_V^- - l_V^+\right]\right]\Big\vert_{t_k^\mathrm{spike}}$
$\tau_\mathrm{s}\lambda_I' = -\lambda_I + \lambda_V$	for any k	$+\left[\frac{1}{\tau_\mathrm{m}(\dot{V}^-)_{n(k)}}\right]\Big\vert_{t_k^\mathrm{spike}} \sum_m w_{mn(k)}\left[(\lambda_V^+ - \lambda_I^+)_m\right]\Big\vert_{t_k^\mathrm{spike}+d_{mn(k)}}$
Gradient updates		
Weight learning: $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}w_{ji}} = -\tau_\mathrm{s}\sum_{t_k^\mathrm{spike}\ \text{from}\ i}(\lambda_I)_j\Big\vert_{t_k^\mathrm{spike}+d_{ji}}$		Delay learning: $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}d_{ji}} = -w_{ji}\sum_{t_k^\mathrm{spike}\ \text{from}\ i}(\lambda_I - \lambda_V)_j\Big\vert_{t_k^\mathrm{spike}+d_{ji}}$

Table 1: Forward and backward propagation of a Leaky Integrate-and-Fire (LIF) neuron. V and I are the membrane potential and input current and λ_V and λ_I the corresponding adjoint variables. τ_m and τ_s are the membrane and synaptic time constants. w_{mn} is the synaptic weight and d_{mn} is the synaptic delay from neuron n to neuron m. ϑ is the firing threshold. The dot denotes the derivative with respect to time, and the prime the derivative backwards in time. Superscript "-" and "+" denote the values before and after a transition. t_k^spike denotes the k^th spike and $n(k)$ the index of the neuron that fired this spike. l_p and l_V define the shape of the loss function. Without the transition condition (in other words, with $\vartheta = \infty$), we arrive at the Leaky Integrator (LI) neuron. We highlight all our additions in red.

times, unconstrained by a timestep grid, but this comes at a cost. Only a limited subset of neuron models [45, 46] have dynamics that can be directly interpolated between events, algorithms and data structures become increasingly complicated if recurrent connectivity and delays are required [47] and, the perceived computational advantage of event-based simulation dwindles rapidly as the frequency of events increases [48]. For example, each input in the SHD dataset fires at approximately 10 spikes per second. If we consider a hidden neuron densely connected to these inputs, it receives 7000 spikes per second, meaning that an event-based neuron would have to update 7× more frequently than one simulated using a 1 ms timestep.

Nowotny, Turner, and Knight [21] described the initial GeNN implementation of EventProp, which has subsequently been implemented as a 'compiler' [41] for mlGeNN. Here, we have added delays to the mlGeNN EventProp compiler, enabling the easy exploration of delay learning in a wide range of network architectures. When using our method, increasing the range of delays only requires enlarging a *per-neuron* buffer, resulting in a much lower memory overhead than convolution-based approaches and thus allowing efficient handling of long delays. Our method further allows the outputs of neurons to feed into general spike- and/or voltage-dependent loss functions, offering great flexibility in designing training objectives. Our approach outperforms prior work using EventProp on the SHD and SSC datasets [21] – achieving superior performance with almost 5× fewer parameters. Additionally, we demonstrate a speedup of up to 26× and memory savings of over 2× compared to surrogate-gradient-based dilated convolutions implemented in PyTorch [27].

2 Results

The EventProp algorithm is an application of the adjoint method for calculating sensitivities in hybrid dynamical systems [49, 50] to the problem of calculating the gradient of a loss function in an SNN. From an ML perspective, it can be considered an event-driven form of the popular BPTT gradient descent-based learning in RNNs. The difference is that the gradient computation employs a hybrid approach: the derivatives are determined through both continuous differential equations and discrete state transitions of

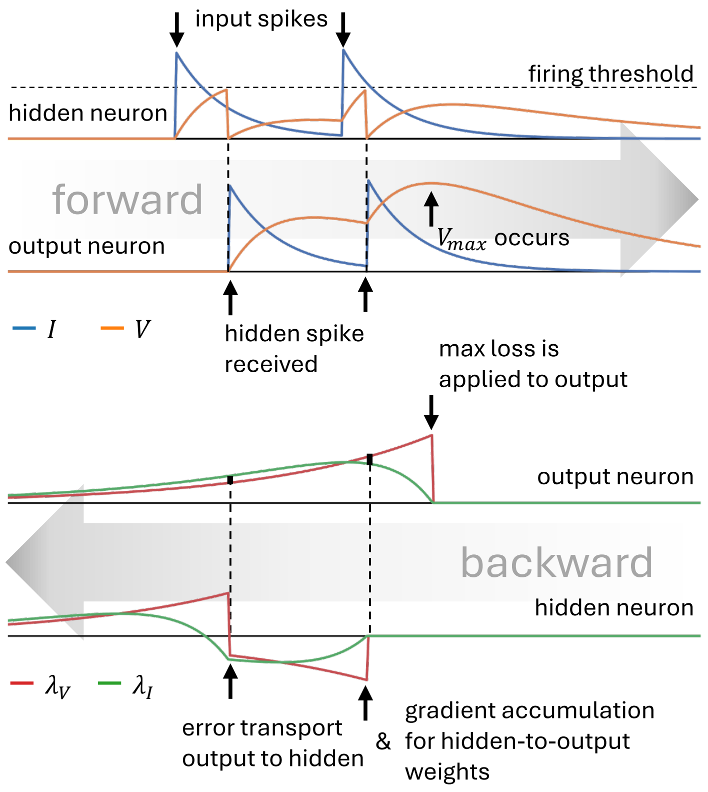

Figure 1: Illustration of the original EventProp formalism without delays. In a minimal example, a network has input neurons, one hidden layer and an output layer. Input spikes cause instantaneous jumps in the hidden I variable (blue lines), which drives the V variable (orange lines). When V reaches the firing threshold, it is reset and spikes are emitted, which instantaneously jump the I variable of the output neurons. This forward pass is followed by a backward pass, where the "blame" of each weight for the eventual loss is calculated. The adjoint variables λ_V and λ_I are proxies of this blame. The calculation occurs backwards in time. Here we illustrate the use of a readout and loss that are based on the maximum voltage of the output neurons. Accordingly, the loss causes a jump in the output λ_V variable at the time when the maximum output voltage occurred in the forward pass. This blame is then transported as jumps of λ_V of hidden neurons at the times when hidden spikes had occurred. Gradient updates to the hidden-to-output weights occur at the same time. Note that the plots are for illustrative purposes only and not to scale, other than matching pairs of λ_V and λ_I being at the same scale.

adjoint variables that occur at saved spike times.

Intuitively, in an SNN, synaptic weights can only affect the network activity, and hence cause loss, at times when a spike is transmitted. The adjoint variables track the potential loss caused by each neuron, and their value at the time of a spike quantifies how much loss that spike contributed to the overall loss, essentially assigning responsibility or "blame" to both the spike event and the synaptic weights involved. For output neurons, blame for loss is directly added to the adjoint variable corresponding to the membrane dynamics (jump in λ_V of the output neuron where $V_{\max}$ occurred in Figure 1), while for internal neurons, it propagates backward from downstream connections (jumps in λ_V of the hidden neuron at saved spike times in Figure 1). The gradient for each synaptic weight (w_{ji}) accumulates across pre-synaptic firing events, capturing the moments when that connection influenced the post-synaptic neuron's behaviour and consequently affected network performance. The differential equations governing the adjoint variables precisely track how input fluctuations drive membrane changes that ultimately impact the loss – whether by triggering spikes or directly affecting the output neuron's contribution to the loss function.

Here, we derive an extended EventProp formalism that extends the original algorithm in both, that it can be applied to SNNs with delays and that it enables learning of suitable delays, i.e. to calculate

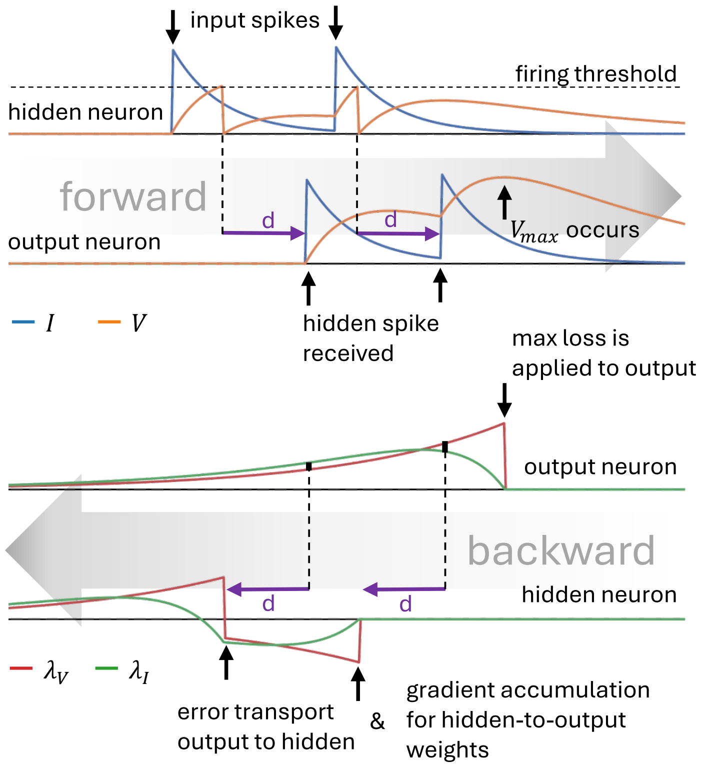

Figure 2: Illustration of the extended EventProp algorithm for SNNs with delays. In essence, the forward pass works in the same way as for networks without delays (Fig. 1), except that the jump in post-synaptic I variables occurs with a delay d. In the backward pass, this translates into backwards transport of blame at the original spike times but transporting values of post-synaptic adjoint variables at a delayed time (in forward time). Gradients with respect to weights and delays are accumulated at the same saved spike times and based on the same delayed quantities.

gradients of a loss function with respect to delays. Although the EventProp formalism accommodates various neuron models [51], for simplicity, we will describe it for the LIF neurons and exponential current-based synapses employed by Wunderlich and Pehle [18].

The forward pass of the SNN is described by first-order ordinary differential equations for the dynamics of the current I and voltage V; and discontinuous jumps in the variables at the occurrence of spikes (see table 1). Note that this only differs from the forward pass described by Wunderlich and Pehle [18] due to the delay d_{mn} between neuron n and neuron m which causes the k-th jump in the network in the current I of neuron m caused by a spike in neuron n at time t_k^{spike} to occur at time $t_k^{\text{spike}} + d_{mn}$, see also Figure 2, top. If all d_{mn} are zero, this reverts back to the original EventProp formalism.

To obtain the backward pass for our network with delays, we take the derivative of the loss function with respect to a weight, w_{ji}. The loss function can depend directly on spike timing (which is compatible with the gradient calculation because LIF neuron spike times vary smoothly with weight changes), or it can be expressed as an integral over the voltage V (where the integral effectively smooths out the discontinuities that would otherwise make the gradient calculation difficult when spikes occur).

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}w_{ji}} = \frac{\mathrm{d}}{\mathrm{d}w_{ji}} \left[l_p(\mathcal{S}) + \int_{t=0}^{T} l_V(V,t)\mathrm{d}t \right], \tag{1}$$

where l_p is the loss term that depends on the set of output spike times $\mathcal{S} \equiv \{t_k^{\text{spike}} \mid k = 1, \dots, N_{\text{spike}}\}$, and l_V is the voltage-dependent loss. Following the adjoint method, we then add Lagrange multipliers λ_I

and λ_V, multiplied by the continuous dynamics, which can be interpreted as dynamic constraints,

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}w_{ji}} = \frac{\mathrm{d}}{\mathrm{d}w_{ji}} \left[l_p(\mathcal{S}) + \int_{t=0}^{T} \left[l_V(V,t) + \lambda_V f_V + \lambda_I f_I \right] \mathrm{d}t \right], \tag{2}$$

where $f_V \equiv \tau_\mathrm{m}\dot{V} + V - I$ encodes the voltage dynamics constraint and $f_I \equiv \tau_\mathrm{s}\dot{I} + I$ the current dynamics constraint.

The essence of the original EventProp derivation was to split the integral in (2) at the N_spike spike times t_k^spike when the jumps occur. Between the jumps, everything is well-defined and the standard adjoint method is easily applied – resulting in the backward dynamics for the adjoint variables. With some work (see derivations by Wunderlich and Pehle [18]), the values of adjoint variables before and after jump times in the backward pass can then be defined so that the remaining expression for the gradient becomes a simple sum over λ_I values at spike times, leading to an event-based weight update rule:

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}w_{ji}} = -\tau_\mathrm{s} \sum_{\substack{t_k^\mathrm{spike} \\ \text{from } i}} (\lambda_I)_j \Big|_{t_k^\mathrm{spike}}. \tag{3}$$

We apply a similar approach here, but in our network with delays, spike emission and arrival times become separate events. We address this by extending the set of spike times to the set of all event times that include both spike emission times t_k^spike and spike arrival times $t_k^\mathrm{spike} + d_{mn(k)}$, where $n(k)$ is the index of the neuron that fired the k^th spike:

$$\mathcal{E} \equiv \mathcal{S} \cup \{ t_k^\mathrm{spike} + d_{m,n(k)} \mid k = 1, \ldots, N_\mathrm{spike}, m = 1, \ldots, N \}. \tag{4}$$

We denote the elements of this set as $t_k^\mathrm{event} \in \mathcal{E}$ and assume that they are ordered such that $t_k^\mathrm{event} \leq t_{k'}^\mathrm{event}$ for $k < k'$. We can then proceed in the same way as in the original EventProp derivation. The resulting backward pass becomes computable because all delays are nonnegative, meaning that by the time we compute the adjoint variables for the spiking neuron i at time t, the adjoint variables for the postsynaptic neurons j receiving the spike at time $t + d_{ji}$ will have already been calculated before, in backward time (see Figure 2, bottom).

After extensive calculations (section 4), we arrive at a formula that remains fully event-based for synaptic actions in the backward pass and thus can be efficiently computed,

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}w_{ji}} = -\tau_\mathrm{s} \sum_{\substack{t_k^\mathrm{spike} \\ \text{from } i}} (\lambda_I)_j \Big|_{t_k^\mathrm{spike} + d_{ji}}. \tag{5}$$

Using the same approach, we can also derive the derivative of the loss function with respect to the delays d_{ji}. Remarkably, the derived backward dynamics of the adjoint variables remain the same, meaning that using the exact same λ_I and λ_V dynamics in the backward pass, we *can also perform gradient descent on synaptic delays in an event-based manner.* The resulting formula for the delay gradients is

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}d_{ji}} = -w_{ji} \sum_{\substack{t_k^\mathrm{spike} \\ \text{from } i}} (\lambda_I - \lambda_V)_j \Big|_{t_k^\mathrm{spike} + d_{ji}}. \tag{6}$$

We thus end up with an event-based weight and delay learning algorithm which, we believe, is the first to enable delay learning in networks with multiple recurrently-connected layers and with multiple spikes per neuron.

2.1 Sequence detection task

To test the effectiveness of our method, we evaluated it in various machine learning settings. First, we generated a simple binary classification task that can be solved with perfect accuracy using optimal delays. Specifically, we used two LIF neurons connected to two LI neurons (see Table 1), with all connection strengths set to 1. The task involves two classes:

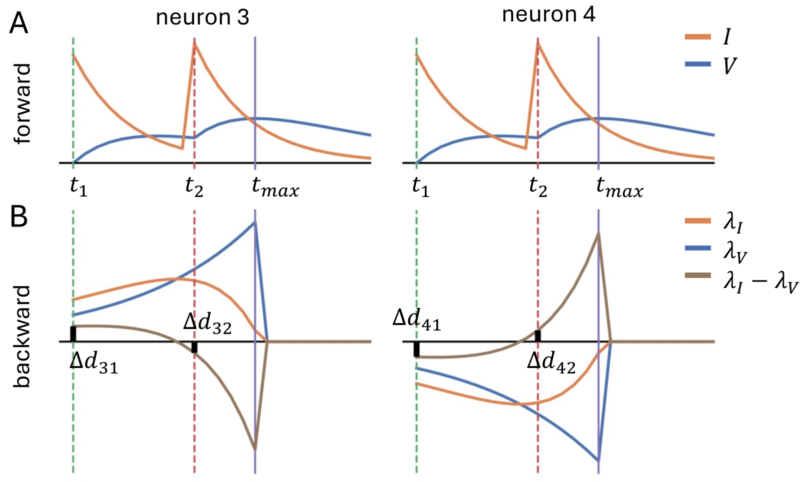

Figure 3: Illustration of the learning updates in the sequence detection task. A) Voltage V and current I in the forward pass. B) Adjoint variables λ_V and λ_I. The loss is injected into λ_V at $t_{\max}$ when the output voltage reached maximum in the forward pass. This then propagates into λ_I and eventually leads to delay updates at the saved spike times t_1, t_2. For neuron 3, the update to d_{32} at t_2 is negative and the update to $d_{3,1}$ at t_1 positive, meaning that (subject to delays being non-negative) the excitatory postsynaptic potential (EPSP) from neuron 2 is moved to earlier and the EPSP from neuron 1 to later, moving them close together. For neuron 4, the opposite is the case, where the updates are such that the EPSPs are moved apart. This is exactly what is needed to increase the maximal output voltage of neuron 3 and decrease the maximal output of neuron 4. When the other input class is active, all roles are inverted, again leading to the correct delay updates.

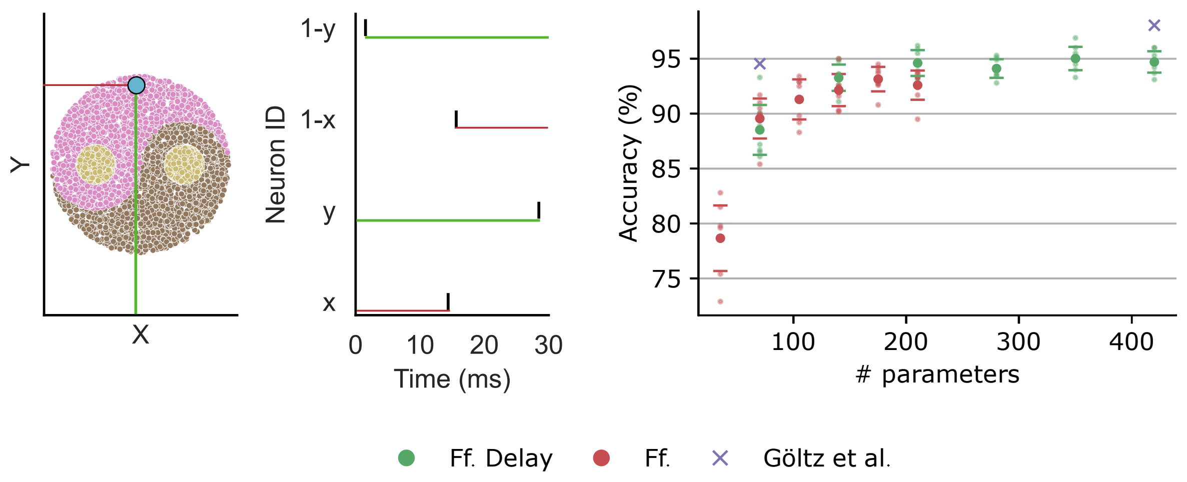

Figure 4: **Left** The Yin-Yang (YY) dataset, with temporal encoding of example datapoint highlighted by a blue dot. **Right** We generate separate training, validation, and test sets with 5000, 1000 and 1000 examples respectively; and report the test performance using the model which performed best on the validation set. We look at feedforward networks with and without delays. The purple crosses shows the results reported by Göltz et al. [39]. The points show average accuracy, and the error bars show standard deviation over 8 runs. We also show all individual results as smaller data points.

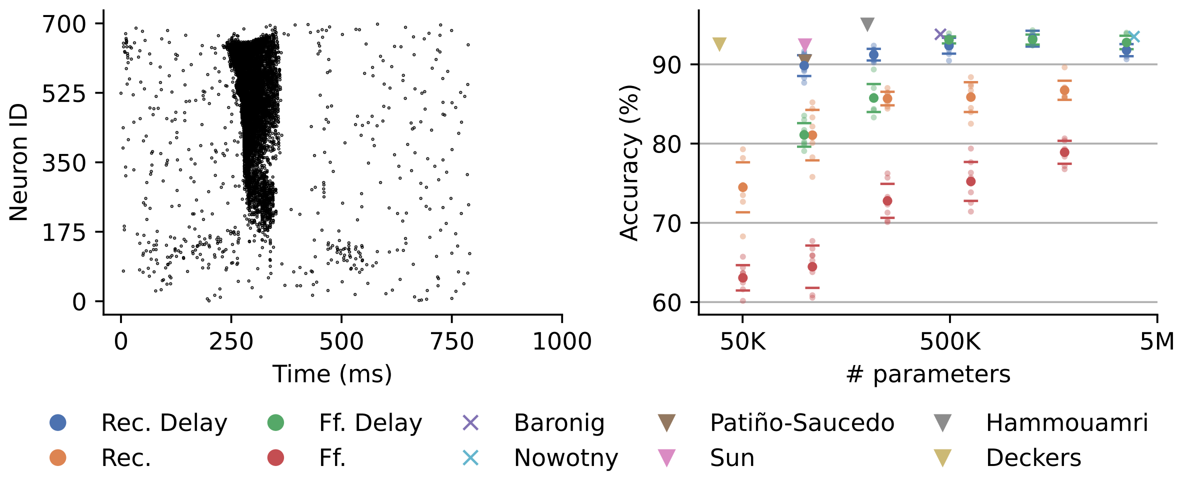

Figure 5: **Left** An example of a speaker saying "five" from the Spiking Heidelberg Digits (SHD) dataset. **Right** The SHD dataset does not have a validation set; we perform early stopping when training accuracy does not improve for 15 epochs and report the corresponding test accuracy. Ff denotes a model with 2 feedforward hidden layers, and Rec denotes a model with one recurrently connected hidden layer. We implemented models with 128, 256, 512 and 1024 hidden neurons. We also show SOTA results by Baronig et al. [52], and previous EventProp results by Nowotny, Turner, and Knight [21]. The triangles show results of other delay learning methods that appear to have used the test set for validation [27, 28, 35]. The points show average accuracy, and the error bars show standard deviation over 8 runs. We also show all individual results as smaller data points.

- Class 1: The first input neuron emits a spike at 0 ms, and the second emits a spike at 10 ms.
- Class 2: The first input neuron emits a spike at 10 ms, and the second emits a spike at 0 ms.

The output of the network is determined based on the maximum voltage reached by the two output neurons, each corresponding to one of the output classes. The class associated with the neuron that reaches the higher voltage is selected as the network's prediction. We can observe that having 10 in the diagonals and 0 everywhere else in our 2×2 delay matrix solves the task. We started with the least optimal delay distribution – delays on the diagonals being 0, and 10 everywhere else – and, with the learning rate set to 1, we achieve 100 % accuracy after encountering both examples 6 times. This result demonstrates that, by introducing our delay updates into the learning framework, SNNs become capable of not only coincidence but sequence detection. Figure 3 illustrates the gradient calculation in this task.

2.2 Yin-Yang dataset

We also experimented with the Yin-Yang dataset [20], which has been tested using both EventProp (without delays) [18] and DelGrad [39], see Figure 4. Similarly to DelGrad, we looked at feedforward networks and varied the size of the hidden layer from 5 to 30. We initialised all delays at 0, allowing them to evolve during training and trained using the time-invariant mean squared error loss [39], see 4 for derivation. Our findings (Figure 4) are very similar to DelGrad – with a fixed number of parameters, networks perform similarly (i.e. halving the number of hidden neurons does not decrease performance if delays are introduced). If the number of parameters is not a constraining factor, training delays *and* weights is always advantageous.

2.3 Spiking Heidelberg Digits

Nowotny, Turner, and Knight [21] achieved state-of-the-art results on SHD with a 'delay line' approach, which involved creating 10 copies of the input and cumulatively delaying each by 30 ms. While this architecture achieved high accuracy, it required a large number of parameters, so we instead experimented with learnt delays in the input-to-hidden and hidden-to-hidden connections. For controlling the training dynamics in the hidden population, a target firing rate needs to be set, with the corresponding spike regularisation strength. We kept our target firing rate fixed at 14 spikes per example and treated the regularisation strength as a tunable hyperparameter, which we optimised using 10-fold cross-validation, leaving one speaker out of the training set in each fold. Tuning this parameter was crucial (particularly for networks with recurrent connectivity) and once we identified the best-performing model in cross-validation, we retrained it with the same parameters on the full training set. We enforced early stopping if training accuracy did not improve for 15 epochs. Using this methodology, our best model achieved a training accuracy of $98.47 \pm 0.40\,\%$ and a test accuracy of $93.24 \pm 1.00\,\%$ as depicted in Figure 5. This configuration included 512 hidden neurons with recurrent connections. The feedforward delays were initialised from a uniform distribution in the range of $0 - 150$ ms, while the recurrent delays were all initialised to 0 ms. The difference between these results and those reported using the 'delay line' approach ($93.5 \pm 0.7\,\%$ [21]) are not statistically significant ($p = 0.442$, t-test, $n = 8$), and we achieved them with around 5 times fewer parameters. We also experimented with different hidden layer sizes and feedforward models. We found that decreasing the hidden neuron number to 256 does not significantly decrease the accuracy for either architecture. However, if we decrease the number of neurons even further to 128, we observe a significant drop for the feedforward architecture but not for the recurrent one. Increasing the hidden layer to 1024 neurons leads to overfitting.

While state-of-the-art models – such as the work by Hammouamri, Khalfaoui-Hassani, and Masquelier [27], Deckers et al. [35], and Sun et al. [53] – reported higher accuracies, these were obtained using the test set for validation and early stopping, rather than using a separate validation set. Schöne et al. [54] mention that they also evaluate in this way to achieve a "fair comparison to others". However, as Baronig et al. [52] argued, not only is this not methodologically "clean" but it may also not be entirely fair due to potential overfitting (we note that the highest test accuracy we observed was 95.32 %). Furthermore, we also note that model performance on the SHD dataset is nearing saturation, with the best-performing models achieving an accuracy of around 93 %. Following Isaksson et al. [55] and Nowotny [56], we calculated the Bayesian confidence intervals with naive assumptions on error rates. 93 % accuracy has overlapping confidence intervals with higher accuracies (e.g., 94 % and 95 %), indicating that further improvements in accuracy are likely not statistically meaningful given the test set size (2264) [56].

2.4 Spiking Speech Commands

SSC is significantly more challenging than SHD as the audio recordings were created in noisy environments, and the dataset has more classes. We initially experimented with single recurrent hidden layer architectures similar to those employed by Nowotny, Turner, and Knight [21] and, after replacing the delay line inputs with learnable delays, we achieved similar performance. Interestingly, we observed little to no benefit of adding delays to larger networks but, as we decreased the number of hidden neurons, the delays became highly beneficial. These architectures were extremely robust to decreasing the number of hidden neurons. While many state-of-the-art models use deeper architectures with more hidden layers [27, 58, 52], we found that deeper architectures with recurrent connections became highly unstable even without delays in the connections. Therefore, to improve upon previous results, we explored deeper *feedforward* architectures with delays and found the best performing architecture to be a model with 2 feedforward hidden layers.

Our results are shown in Figure 6. Our best model achieved a training accuracy of 79.6 ± 1.0 , a validation accuracy of $78.1 \pm 1.0\,\%$ (n=8) and a test accuracy of $76.1 \pm 1.0\,\%$. We also experimented with smaller models, which as expected, achieved a lower training accuracies. Compared to other SNNs with delays, we observe that we outperform Sadovsky, Jakubec, and Jarina [57], with their test accuracy results at 72.03 %. Deckers et al. [35] introduced a constrained adaptive LIF neuron model to a delayed network and reached 80.23 % test accuracy. They also tested LIF models, achieving 75.94 %, which we slightly outperformed.

2.5 Braille letter reading

As highlighted by Walters et al. [60], most neuromorphic benchmarks contain more spatial than temporal information. Although the SHD and SSC datasets *do* contain temporal information, it may not be important enough to require the fine timesteps our approach enables. Therefore, we also evaluated our

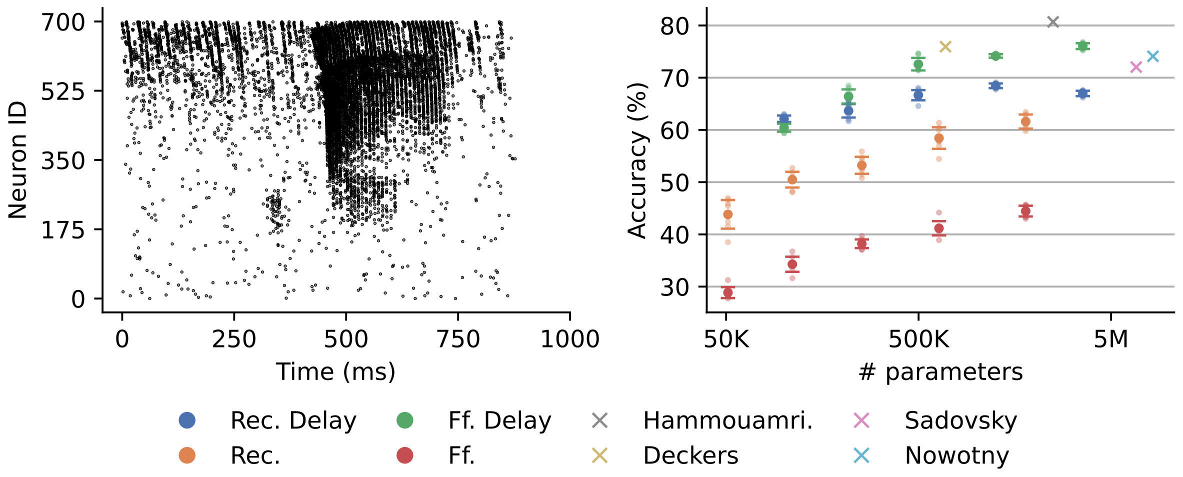

Figure 6: **Left** An example of a speaker saying "two" from the Spiking Speech Commands (SSC) dataset. **Right** For SSC, which has separate training, validation, and test sets, we apply early stopping when validation accuracy no longer improves and report the corresponding test accuracy. Ff denotes a model with 2 feedforward hidden layers, and Rec denotes a model with one recurrently connected hidden layer. We implemented models with 128, 256 and 512 hidden neurons. Crosses show results from other delay learning models [27, 35, 57], and we also show previous Eventprop results by Nowotny, Turner, and Knight [21]. The points show average accuracy, and the error bars show standard deviation over 8 runs. We also show all individual results as smaller data points.

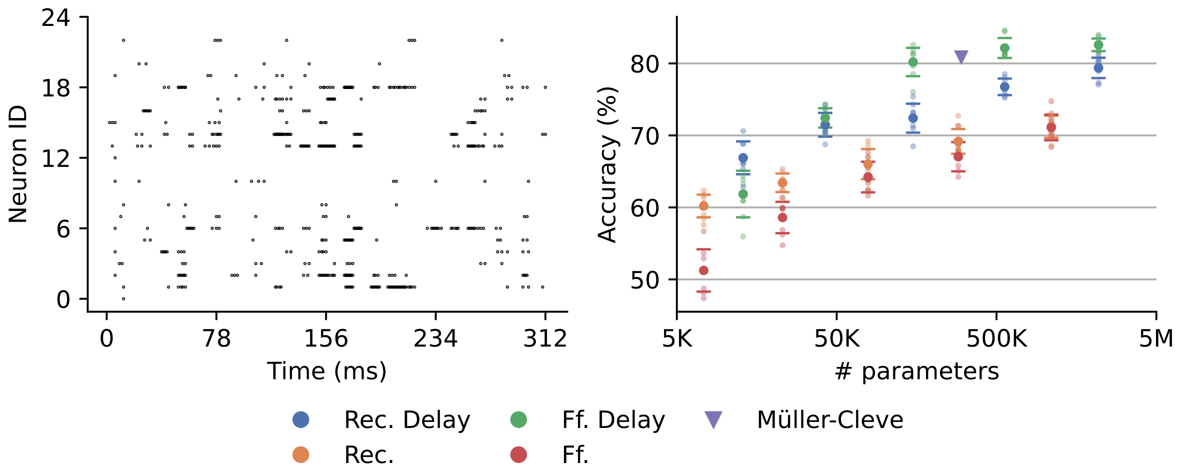

Figure 7: **Left** Example of the letter 'S' from the braille letter reading dataset. We split the dataset into training, validation and test set in a ratio of 70 : 10 : 20, respectively. We tune our hyperparameters based on validation results and report the corresponding test accuracy. **Right** Ff denotes a model with 2 feedforward hidden layers and Rec denotes a model with one recurrently connected hidden layer. We implemented models with 128, 256, 512 and 1024 neurons. The triangle depicts validation results reported by Müller-Cleve et al. [59]. The points show average accuracy, and the error bars show standard deviation over 8 runs. We also show results from all individual results with smaller data points.

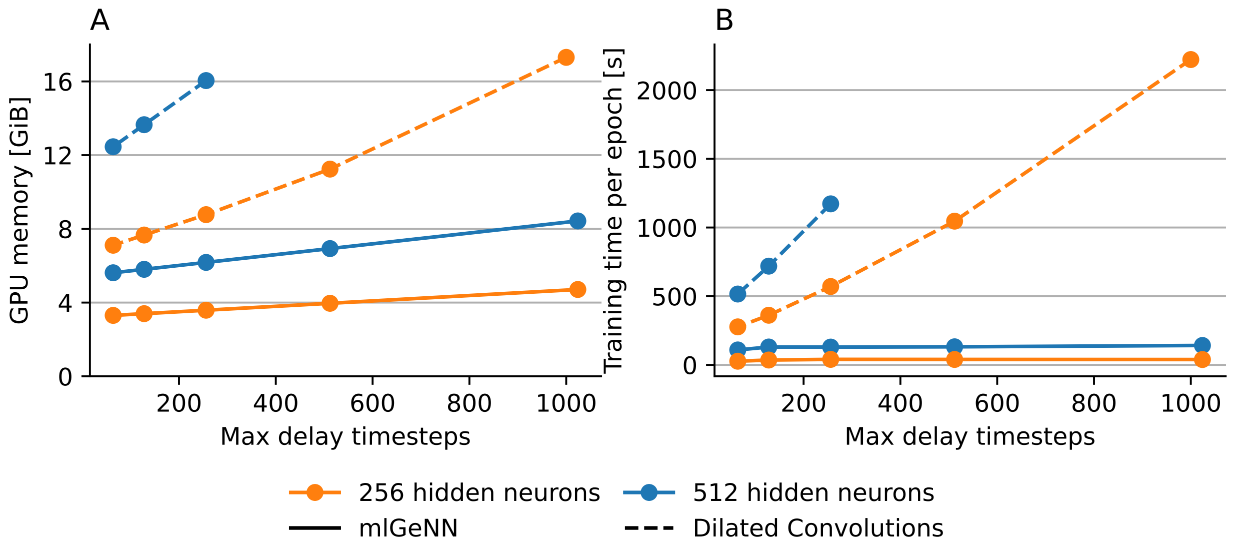

Figure 8: Comparing cost of training networks with delays using EventProp and mlGeNN against Dilated Convolutions (DC) implemented using SpikingJelly and PyTorch. **(A)** Peak GPU memory usage. Missing data points indicate where PyTorch ran out of GPU memory. **(B)** Time to train one SHD epoch. All experiments were performed on a workstation with an NVIDIA RTX A5000 GPU. All models have two feedforward hidden layers and use batch size 256 and 1 ms timesteps.

approach on a braille letter reading dataset [59].

We again trained, validated and tested 2-layer feedforward and single-layer recurrent networks with and without delays and with hidden layers of various sizes $(64, 128, 256, 512, 1024)$ on the $70 : 10 : 20$ training, validation, and test splits provided by Müller-Cleve et al. [59]. The results shown in Figure 7 show a similar pattern to the SSC results – introducing delays in large recurrent networks shows no benefit, but adding them to smaller networks significantly improves performance.

Our best-performing model had two feedforward hidden layers, with 1024 neurons each, and achieved $83.1 \pm 1.5\,\%$ on the test set. This model outperforms the recurrent network with a single hidden layer of 450 neurons and 8 input copies described in the original publication [59], which obtained a test accuracy of $80.9 \pm 0.3\,\%$. Our smaller feedforward network with hidden layers of size 256 achieved a test accuracy of $81.0 \pm 0.7\,\%$ so also outperformed Müller-Cleve et al. [59], with half the number of parameters. Additionally, Müller-Cleve et al. [59] evaluated their models using an $80 : 20$ training-test split without a validation set, which is problematic for the same reasons identified in SHD dataset studies. Pedersen et al. [61] previously trained on this dataset but simplified it to 7 classes to accommodate the Xylo chip [62]. We are not aware of any other research involving delay learning on this dataset.

2.6 Computational performance

Finally, we benchmarked our training procedure against the Dilated Convolution implementation provided by Hammouamri, Khalfaoui-Hassani, and Masquelier [27] using PyTorch 2.5.1 and SpikingJelly 0.0.0.0.15 [63]. Because the Dilated Convolution method does not support recurrent delays, we benchmarked feedforward models with 2 hidden layers. We measured the peak memory utilisation of mlGeNN using the "nvidia-smi" command line tool – as GeNN allocates memory from CUDA directly – and of PyTorch using torch.cuda.max_memory_allocated – so details of the memory allocator are disregarded.

The benchmarking results are illustrated in Figure 8. Because increasing the maximum number of delay slots in mlGeNN only involves increasing the size of a *per-neuron* buffer rather than increasing the size of a *per-synapse* kernel, longer delays have much lower memory requirements in mlGeNN (Figure 8A). Similarly, the increasing computational cost of convolving larger and larger kernels means that training time increases rapidly when using Dilated Convolutions (Figure 8B), whereas with mlGeNN, there is only a very small initial increase in training time as the maximum number of delay timesteps increases. Other gradient-based delay learning methods [38, 37, 28] do not use temporal convolutions, so their memory and computational requirements would not grow *as* quickly, but they are all still based on dense BPTT so – based on benchmarking performed by Nowotny, Turner, and Knight [21] – we would still expect mlGeNN to be significantly faster and use less memory.

The slight increase in mlGeNN training time observed as the maximum number of delay timesteps increases is likely due to the effects of caching on the delay buffers. These buffers are allocated in GPU global memory, so the efficiency of updating them depends on whether they persist in the GPU's caches for long enough for locality in the delays to be exploited. At the large batch sizes used for these experiments, the complete buffers will not fit in the 6 MiB L2 cache of the RTX A5000 GPU so the effects seen here are likely due to an increase in the rate at which delay buffer data gets evicted from the cache as the size of the buffers increases.

3 Discussion

Delays in Spiking Neural Networks have been extensively studied from both machine learning and computational neuroscience perspectives, with recent interest spurred by their ability to improve performance in machine learning applications [37, 27]. Delays can be efficiently implemented on several neuromorphic chips [34, 33], but, as is the case with much of current neuromorphic research, there is a lack of focus on delay *learning* algorithms that could be implemented on future neuromorphic hardware. Instead, many methods rely on arithmetically intensive approaches that are only practical on GPUs, such as convolutions in networks trained with BPTT with surrogate gradients. Our method combines the best of both worlds: its theoretical foundations enable efficient implementation on neuromorphic hardware (EventProp has already been implemented in SpiNNaker 2 [33]), while our GeNN-based implementation takes advantage of readily available GPU hardware. This versatility allows us to address a broader range of platforms while improving performance on complex temporal tasks.

The beneficial scaling of EventProp allows long sequence lengths and hence finer timesteps, which might offer enhanced precision in spatio-temporal tasks. However, leading models on SHD and SSC employ large timesteps of 10 or even 25 ms [27, 58] and, while these coarser time grids may have been primarily chosen to fit within GPU memory, they could also simplify the tasks by reducing their effective sequence length (assuming high temporal precision is not required). This reflects a broader challenge in SNN research: while neuromorphic architectures are appealing for their energy efficiency and temporal precision, it remains unclear how much temporal precision is needed for any given task and how it is best exploited in practice. Timestep length also has additional relevance for delay learning, as delays are discretised to integer multiples of the timestep. However, our previous work indicates that the speech recognition tasks considered here may not require precise delays [64].

Another open question regards initialisation. While synaptic weights can be initialised in a straightforward manner by sampling from a normal distribution centred around zero, the initialisation of delays is less intuitive. Experimental observations of delay distributions are challenging to interpret [65], and the optimal distribution may depend heavily on the task. We followed the common approach of initialising delays with values sampled from uniform distributions [27]. However, interestingly, we observed that while a few delays grew considerably, most remained relatively short. This distribution might reflect a small-world network structure, where most neurons connect to nearby neighbours (short delays) with only a few long-range connections (long delays).

As we show across multiple datasets, recurrent delays can offer substantial advantages when implementing networks with stricter size constraints. However, initialising recurrent delay distributions poses additional computational challenges. Initialising feedforward delays within the range $[0, d_{\max}]$ is logical, as adding a homogeneous delay x would yield the same outcomes within the range $[x, d_{\max} + x]$. However, this symmetry does not extend to recurrent delays, making their initialisation significantly more complex. Recent studies have focused on optimising delay distributions at the network level [66], but the question of layer-specific initialisation remains open.

This work showed the benefit of delays on datasets commonly used to benchmark SNNs. Exploring them in other tasks where they could be particularly beneficial (e.g. sound localisation [67], or motion detection [68, 69]) is an interesting avenue for future work.

4 Methods

4.1 Theory

4.1.1 Learning Weights in networks with delay

We start by defining our two differential equations in the implicit form for the membrane potentials and input currents, respectively.

$$f_V \equiv \tau_\mathrm{m}\dot{V} + V - I = 0 \tag{7}$$
$$f_I \equiv \tau_\mathrm{s}\dot{I} + I = 0 \tag{8}$$

In the following, we will assume that all event times $\mathcal{E}$ are distinct, both in terms of spikes occurring and of spikes arriving. In continuous time, this is not unlikely, but also, as argued in [18], the equations do not break down if spikes occur or arrive at the same time.

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}w_{ji}} = \frac{\mathrm{d}}{\mathrm{d}w_{ji}}\left[l_p(\mathcal{S}) + \sum_{t_k^\mathrm{event}\in\mathcal{E}} \int_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \left[l_V(V,t) + \lambda_V\cdot f_V + \lambda_I \cdot f_I\right]\mathrm{d}t\right] \tag{9}$$
$$\frac{\partial f_V}{\partial w_{ji}} = \tau_\mathrm{m}\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial V}{\partial w_{ji}} + \frac{\partial V}{\partial w_{ji}} - \frac{\partial I}{\partial w_{ji}} \tag{10}$$
$$\frac{\partial f_I}{\partial w_{ji}} = \tau_\mathrm{s}\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial I}{\partial w_{ji}} + \frac{\partial I}{\partial w_{ji}} \tag{11}$$

We apply the derivative and use (10) and (11) to obtain

$$\begin{aligned}\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}w_{ji}} = &\sum_{t_k^\mathrm{spike}\in\mathcal{S}} \frac{\partial l_p}{\partial t_k^\mathrm{spike}}\frac{\mathrm{d}t_k^\mathrm{spike}}{\mathrm{d}w_{ji}} \\ &+ \sum_{t_k^\mathrm{event}\in\mathcal{E}} \int_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \left[\frac{\partial l_V}{\partial V}\cdot\frac{\partial V}{\partial w_{ji}} + \lambda_V\cdot\left(\tau_\mathrm{m}\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial V}{\partial w_{ji}} + \frac{\partial V}{\partial w_{ji}} - \frac{\partial I}{\partial w_{ji}}\right)\right. \\ &\qquad\left. + \lambda_I\cdot\left(\tau_\mathrm{s}\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial I}{\partial w_{ji}} + \frac{\partial I}{\partial w_{ji}}\right)\right]\mathrm{d}t \\ &\quad + l_{V,k+1}^{-}\frac{\mathrm{d}t_{k+1}^\mathrm{event}}{\mathrm{d}w_{ji}} - l_{V,k}^{+}\frac{\mathrm{d}t_k^\mathrm{event}}{\mathrm{d}w_{ji}}\end{aligned} \tag{12}$$

Using partial integration, we obtain

$$\int_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \lambda_V\cdot\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial V}{\partial w_{ji}}\mathrm{d}t = -\int_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \dot{\lambda}_V\cdot\frac{\partial V}{\partial w_{ji}}\mathrm{d}t + \left[\lambda_V\cdot\frac{\partial V}{\partial w_{ji}}\right]_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \tag{13}$$

and

$$\int_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \lambda_I\cdot\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial I}{\partial w_{ji}}\mathrm{d}t = -\int_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \dot{\lambda}_I\cdot\frac{\partial I}{\partial w_{ji}}\mathrm{d}t + \left[\lambda_I\cdot\frac{\partial I}{\partial w_{ji}}\right]_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}}. \tag{14}$$

Inserting this into (12), we get

$$\begin{aligned}\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}w_{ji}} &= \sum_{t_k^{\text{spike}}\in\mathcal{S}} \frac{\partial l_p}{\partial t_k^{\text{spike}}}\frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}w_{ji}} \\ &+ \sum_{t_k^{\text{event}}\in\mathcal{E}} \int_{t_k^{\text{event}}}^{t_{k+1}^{\text{event}}} \left[\left(\frac{\partial l_V}{\partial V} - \tau_{\mathrm{m}}\dot{\lambda}_V + \lambda_V\right)\cdot\frac{\partial V}{\partial w_{ji}} + \left(-\tau_{\mathrm{s}}\dot{\lambda}_I + \lambda_I - \lambda_V\right)\cdot\frac{\partial I}{\partial w_{ji}}\right]\mathrm{d}t \\ &\quad + \tau_{\mathrm{m}}\left[\lambda_V\cdot\frac{\partial V}{\partial w_{ji}}\right]_{t_k^{\text{event}}}^{t_{k+1}^{\text{event}}} + \tau_{\mathrm{s}}\left[\lambda_I\cdot\frac{\partial I}{\partial w_{ji}}\right]_{t_k^{\text{event}}}^{t_{k+1}^{\text{event}}} \\ &\quad + l_{V,k+1}^{-}\frac{\mathrm{d}t_{k+1}^{\text{event}}}{\mathrm{d}w_{ji}} - l_{V,k}^{+}\frac{\mathrm{d}t_k^{\text{event}}}{\mathrm{d}w_{ji}}\end{aligned} \tag{15}$$

where the last two terms arise from the derivative of the bounds of the integral in the Leibniz rule. We now define the backwards dynamics of the adjoint variables as before,

$$\tau_{\mathrm{m}}\lambda_V^{'} = -\lambda_V - \frac{\partial l_V}{\partial V} \tag{16}$$

$$\tau_{\mathrm{s}}\lambda_I^{'} = -\lambda_I + \lambda_V \tag{17}$$

which cancels the terms containing $\frac{\partial V}{\partial w_{ji}}$ and $\frac{\partial I}{\partial w_{ji}}$, and we get

$$\begin{aligned}\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}w_{ji}} = \sum_{t_k^{\text{spike}}\in\mathcal{S}} \frac{\partial l_p}{\partial t_k^{\text{spike}}}\frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}w_{ji}} + \sum_{t_k^{\text{event}}\in\mathcal{E}} \Bigg(l_{V,k}^{-}\frac{\mathrm{d}t_k^{\text{event}}}{\mathrm{d}w_{ji}} - l_{V,k}^{+}\frac{\mathrm{d}t_k^{\text{event}}}{\mathrm{d}w_{ji}} \\ + \left[\tau_{\mathrm{m}}\left(\lambda_V^{-}\cdot\frac{\partial V^{-}}{\partial w_{ji}} - \lambda_V^{+}\cdot\frac{\partial V^{+}}{\partial w_{ji}}\right) + \tau_{\mathrm{s}}\left(\lambda_I^{-}\cdot\frac{\partial I^{-}}{\partial w_{ji}} - \lambda_I^{+}\cdot\frac{\partial I^{+}}{\partial w_{ji}}\right)\right]\Bigg|_{t_k^{\text{event}}}\Bigg)\end{aligned} \tag{18}$$

We now focus on the spike emission times $t_k^{\text{spike}} \in \mathcal{S}$. Before the jump at t_k^{spike} we have,

$$(V^{-})_{n(k)} - \vartheta = 0, \tag{19}$$

where $n(k)$ denotes the spiking neuron at event k. If we take the derivative of this equation, we get, using the chain rule,

$$\left(\frac{\partial V^{-}}{\partial w_{ji}}\right)_{n(k)} + (\dot{V}^{-})_{n(k)}\frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}w_{ji}} = 0 \tag{20}$$

$$\Rightarrow \quad \frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}w_{ji}} = -\frac{1}{(\dot{V}^{-})_{n(k)}}\left(\frac{\partial V^{-}}{\partial w_{ji}}\right)_{n(k)}, \tag{21}$$

and after the jump,

$$(V^{+})_{n(k)} = 0 \tag{22}$$

$$\Rightarrow \quad \left(\frac{\partial V^{+}}{\partial w_{ji}}\right)_{n(k)} + (\dot{V}^{+})_{n(k)}\frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}w_{ji}} = 0\,. \tag{23}$$

Inserting (21) into (23) we obtain as usual

$$\left(\frac{\partial V^{+}}{\partial w_{ji}}\right)_{n(k)} = \frac{(\dot{V}^{+})_{n(k)}}{(\dot{V}^{-})_{n(k)}}\left(\frac{\partial V^{-}}{\partial w_{ji}}\right)_{n(k)}. \tag{24}$$

For the current $I_{n(k)}$, there is no jump at t_k^{spike}, and also not in its derivative: $(I^{+})_{n(k)} = (I^{-})_{n(k)}$ and $(\dot{I}^{+})_{n(k)} = (\dot{I}^{-})_{n(k)}$ implies

$$\left(\frac{\partial I^{+}}{\partial w_{ji}}\right)_{n(k)} = \left(\frac{\partial I^{-}}{\partial w_{ji}}\right)_{n(k)}. \tag{25}$$

Let us now consider what happens when the spike k at t_k^{spike} is received at all the postsynaptic neurons m at times $t_k^{\text{spike}} + d_{mn(k)}$ (i.e. we look at $\mathcal{E} \setminus \mathcal{S}$). At these times, the input current of the receiving neurons jumps,

$$(I^+)_m = (I^-)_m + w_{mn(k)}. \tag{26}$$

By taking the derivative with respect to w_{ji}, we get

$$\left(\frac{\partial I^+}{\partial w_{ji}}\right)_m + (\dot{I}^+)_m \frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}w_{ji}} = \left(\frac{\partial I^-}{\partial w_{ji}}\right)_m + (\dot{I}^-)_m \frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}w_{ji}} + \delta_{in(k)}\delta_{jm}, \tag{27}$$

where we have used that $\frac{\mathrm{d}(t_k^{\text{spike}}+d_{mn(k)})}{\mathrm{d}w_{ji}} = \frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}w_{ji}}$. Now, using the dynamics equations for I, we also have

$$\tau_{\mathrm{s}}(\dot{I}^+)_m = \tau_{\mathrm{s}}(\dot{I}^-)_m - w_{mn(k)}, \tag{28}$$

and hence,

$$\begin{aligned}\left(\frac{\partial I^+}{\partial w_{ji}}\right)_m &= \left(\frac{\partial I^-}{\partial w_{ji}}\right)_m + \tau_{\mathrm{s}}^{-1} w_{mn(k)} \frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}w_{ji}} + \delta_{in(k)}\delta_{jm} \\ &= \left(\frac{\partial I^-}{\partial w_{ji}}\right)_m + \left[\frac{1}{\tau_{\mathrm{s}}(\dot{V}^-)_{n(k)}} w_{mn(k)} \left(\frac{\partial V^-}{\partial w_{ji}}\right)_{n(k)}\right]\Bigg|_{t_k^{\text{spike}}+d_{mn(k)}} + \delta_{in(k)}\delta_{jm}\end{aligned} \tag{29}$$

where we have used (21) to replace $\frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}w_{ji}}$. Since we have $(V^+)_m = (V^-)_m$ for non-spiking neurons,

$$\left(\frac{\partial V^+}{\partial w_{ji}}\right)_m + (\dot{V}^+)_m \frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}w_{ji}} = \left(\frac{\partial V^-}{\partial w_{ji}}\right)_m + (\dot{V}^-)_m \frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}w_{ji}} \tag{30}$$

From equation (26) and the dynamics equations for V we know

$$\tau_{\mathrm{m}}(\dot{V}^+)_m = \tau_{\mathrm{m}}(\dot{V}^-)_m + w_{mn(k)}. \tag{31}$$

Putting this together, we get

$$\left(\frac{\partial V^+}{\partial w_{ji}}\right)_m = \left(\frac{\partial V^-}{\partial w_{ji}}\right)_m - \tau_{\mathrm{m}}^{-1} w_{mn(k)} \frac{\mathrm{d}t_k^{\text{event}}}{\mathrm{d}w_{ji}} \tag{32}$$

$$= \left(\frac{\partial V^-}{\partial w_{ji}}\right)_m + \left[\frac{1}{\tau_{\mathrm{m}}(\dot{V}^-)_{n(k)}} w_{mn(k)} \left(\frac{\partial V^-}{\partial w_{ji}}\right)_{n(k)}\right]\Bigg|_{t_k^{\text{spike}}+d_{mn(k)}} \tag{33}$$

We now can insert the expressions (21), (25), (24) and (33) into (18) and reorder terms according to which spike the jumps originate from, we get

$$\begin{aligned}\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}w_{ji}} = \sum_{t_k^{\text{spike}}\in\mathcal{S}} \Bigg[&\left(\frac{\partial V^-}{\partial w_{ji}}\right)_n \left[\tau_{\mathrm{m}} \left(\lambda_V^- - \frac{(\dot{V}^+)_n}{(\dot{V}^-)_n}\lambda_V^+\right)_n + \frac{1}{(\dot{V}^-)_n}\left(-\frac{\partial l_p}{\partial t_k^{\text{spike}}} + l_V^+ - l_V^-\right)\right] \\ &+ \tau_{\mathrm{s}}(\lambda_I^- - \lambda_I^+)_n \left(\frac{\partial I^-}{\partial w_{ji}}\right)_n \Bigg]\Bigg|_{t_k^{\text{spike}}} \\ &+ \sum_m \left[\tau_{\mathrm{m}}(\lambda_V^- - \lambda_V^+)_m \left(\frac{\partial V^-}{\partial w_{ji}}\right)_m + \tau_{\mathrm{s}}(\lambda_I^- - \lambda_I^+)_m \left(\frac{\partial I^-}{\partial w_{ji}}\right)_m\right]\Bigg|_{t_k^{\text{spike}}+d_{mn(k)}} \\ &+ \left[\left(\frac{\partial V^-}{\partial w_{ji}}\right)_n \frac{1}{(\dot{V}^-)_n}\right]\Bigg|_{t_k^{\text{spike}}} \left[w_{mn}(\lambda_I^+ - \lambda_V^+)_m\right]\big|_{t_k^{\text{spike}}+d_{mn(k)}} - \left[\tau_{\mathrm{s}}\delta_{in(k)}\delta_{jm}(\lambda_I^+)_m\right]\big|_{t_k^{\text{spike}}+d_{mn(k)}}.\end{aligned} \tag{34}$$

Thus, the update of λ_V of the spiking neuron stays the same as without delays, apart from taking the receiving neurons' corresponding λ_V and λ_I at the delayed time.

$$(\lambda_V^-)_{n(k)} = \left[\frac{(\dot{V}^+)_{n(k)}}{(\dot{V}^-)_{n(k)}}(\lambda_V^+)_{n(k)} + \frac{1}{\tau_\mathrm{m}(\dot{V}^-)_{n(k)}}\left[\frac{\partial l_p}{\partial t_k^\mathrm{spike}} + l_V^- - l_V^+\right]\right]\Bigg|_{t_k^\mathrm{spike}} \tag{35}$$
$$+ \left[\frac{1}{\tau_\mathrm{m}(\dot{V}^-)_{n(k)}}\right]\Bigg|_{t_k^\mathrm{spike}} \sum_m w_{mn(k)} \left[(\lambda_V^+ - \lambda_I^+)_m\right]\big|_{t_k^\mathrm{spike}+d_{mn(k)}}$$

$$(\lambda_V^-)_m = (\lambda_V^+)_m, \text{ if } m \neq n(k) \tag{36}$$

$$\lambda_I^- = \lambda_I^+. \tag{37}$$

The gradient is then given by

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}w_{ji}} = -\tau_\mathrm{s} \sum_{t_k^\mathrm{spike}\in\mathcal{S}} \delta_{in(k)}(\lambda_I)_j\Big|_{t_k^\mathrm{spike}+d_{jn(k)}} = -\tau_\mathrm{s} \sum_{\substack{t_k^\mathrm{spike} \\ \text{from } i}} (\lambda_I)_j\Big|_{t_k^\mathrm{spike}+d_{ji}}. \tag{38}$$

4.1.2 Learning Delays

In the following, we will derive the gradients for delays d_{ji} similarly to our weight gradient derivations.

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}d_{ji}} = \frac{\mathrm{d}}{\mathrm{d}d_{ji}}\left[l_p(\mathcal{S}) + \sum_{t_k^\mathrm{event}\in\mathcal{E}} \int_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \left[l_V(V,t) + \lambda_V \cdot f_V + \lambda_I \cdot f_I\right]\mathrm{d}t\right] \tag{39}$$

$$\frac{\partial f_V}{\partial d_{ji}} = \tau_\mathrm{m}\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial V}{\partial d_{ji}} + \frac{\partial V}{\partial d_{ji}} - \frac{\partial I}{\partial d_{ji}} \tag{40}$$

$$\frac{\partial f_I}{\partial d_{ji}} = \tau_\mathrm{s}\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial I}{\partial d_{ji}} + \frac{\partial I}{\partial d_{ji}} \tag{41}$$

Therefore,

$$\begin{aligned}\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}d_{ji}} = &\sum_{t_k^\mathrm{spike}\in\mathcal{S}} \frac{\partial l_p}{\partial t_k^\mathrm{spike}}\frac{\mathrm{d}t_k^\mathrm{spike}}{\mathrm{d}d_{ji}} \\ &+ \sum_{t_k^\mathrm{event}\in\mathcal{E}} \int_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \left[\frac{\partial l_V}{\partial V}\cdot\frac{\partial V}{\partial d_{ji}} + \lambda_V \cdot \left(\tau_\mathrm{m}\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial V}{\partial d_{ji}} + \frac{\partial V}{\partial d_{ji}} - \frac{\partial I}{\partial d_{ji}}\right)\right. \\ &\qquad \left. + \lambda_I \cdot \left(\tau_\mathrm{s}\frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial I}{\partial d_{ji}} + \frac{\partial I}{\partial d_{ji}}\right)\right]\mathrm{d}t \\ &+ l_{V,k+1}^- \frac{\mathrm{d}t_{k+1}^\mathrm{event}}{\mathrm{d}d_{ji}} - l_{V,k}^+ \frac{\mathrm{d}t_k^\mathrm{event}}{\mathrm{d}d_{ji}}.\end{aligned} \tag{42}$$

Then, using partial integration,

$$\int_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \lambda_V \cdot \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial V}{\partial d_{ji}}\mathrm{d}t = -\int_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \dot{\lambda}_V \cdot \frac{\partial V}{\partial d_{ji}}\mathrm{d}t + \left[\lambda_V \cdot \frac{\partial V}{\partial d_{ji}}\right]_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \tag{43}$$

$$\int_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \lambda_I \cdot \frac{\mathrm{d}}{\mathrm{d}t}\frac{\partial I}{\partial d_{ji}}\mathrm{d}t = -\int_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \dot{\lambda}_I \cdot \frac{\partial I}{\partial d_{ji}}\mathrm{d}t + \left[\lambda_I \cdot \frac{\partial I}{\partial d_{ji}}\right]_{t_k^\mathrm{event}}^{t_{k+1}^\mathrm{event}} \tag{44}$$

and hence,

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}d_{ji}} = \sum_{t_k^{\text{spike}} \in S} \frac{\partial l_p}{\partial t_k^{\text{spike}}} \frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}d_{ji}} \tag{45}$$

$$\sum_{t_k^{\text{event}} \in \mathcal{E}} \left[\int_{t_k^{\text{event}}}^{t_{k+1}^{\text{event}}} \left(\frac{\partial l_V}{\partial V} - \tau_{\text{m}} \dot{\lambda}_V + \lambda_V \right) \cdot \frac{\partial V}{\partial d_{ji}} + (-\tau_{\text{s}} \dot{\lambda}_I + \lambda_I - \lambda_V) \cdot \frac{\partial I}{\partial d_{ji}} \right] \mathrm{d}t$$

$$+ \tau_{\text{m}} \left[\lambda_V \cdot \frac{\partial V}{\partial d_{ji}} \right]_{t_k^{\text{event}}}^{t_{k+1}^{\text{event}}} + \tau_{\text{s}} \left[\lambda_I \cdot \frac{\partial I}{\partial d_{ji}} \right]_{t_k^{\text{event}}}^{t_{k+1}^{\text{event}}} + l_{V,k+1}^{-} \frac{\mathrm{d}t_{k+1}^{\text{event}}}{\mathrm{d}d_{ji}} - l_{V,k}^{+} \frac{\mathrm{d}t_k^{\text{event}}}{\mathrm{d}d_{ji}}.$$

If we now define the adjoint dynamics as usual, the terms in the integral disappear, and we are left with

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}d_{ji}} = \sum_{t_k^{\text{spike}} \in \mathcal{S}} \frac{\partial l_p}{\partial t_k^{\text{spike}}} \frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}d_{ji}} \tag{46}$$

$$+ \sum_{t_k^{\text{event}} \in \mathcal{E}} l_{V,k}^{-} \frac{\mathrm{d}t_k^{\text{event}}}{\mathrm{d}d_{ji}} - l_{V,k}^{+} \frac{\mathrm{d}t_k^{\text{event}}}{\mathrm{d}d_{ji}} + \left[\tau_{\text{m}} \left(\lambda_V^{-} \cdot \frac{\partial V^{-}}{\partial d_{ji}} - \lambda_V^{+} \cdot \frac{\partial V^{+}}{\partial d_{ji}} \right) + \tau_{\text{s}} \left(\lambda_I^{-} \cdot \frac{\partial I^{-}}{\partial d_{ji}} - \lambda_I^{+} \cdot \frac{\partial I^{+}}{\partial d_{ji}} \right) \right] \Bigg|_{t_k^{\text{event}}}.$$

Let's now again first consider the spike emission times t_k^{spike} and the spiking neuron $n(k)$. Before the jump:

$$\left(\frac{\partial V^{-}}{\partial d_{ji}} \right)_{n(k)} + (\dot{V}^{-})_{n(k)} \frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}d_{ji}} = 0 \tag{47}$$

$$\Rightarrow \quad \frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}d_{ji}} = -\frac{1}{(\dot{V}^{-})_{n(k)}} \left(\frac{\partial V^{-}}{\partial d_{ji}} \right)_{n(k)}, \tag{48}$$

and after the jump:

$$\left(\frac{\partial V^{+}}{\partial d_{ji}} \right)_{n(k)} + (\dot{V}^{+})_{n(k)} \frac{\mathrm{d}t_k^{\text{spike}}}{\mathrm{d}d_{ji}} = 0 \tag{49}$$

$$\Rightarrow \quad \left(\frac{\partial V^{+}}{\partial d_{ji}} \right)_{n(k)} = \frac{(\dot{V}^{+})_{n(k)}}{(\dot{V}^{-})_{n(k)}} \left(\frac{\partial V^{-}}{\partial d_{ji}} \right)_{n(k)}. \tag{50}$$

There is no jump in $I_{n(k)}$ or its time derivative at t_k^{spike} which analogous to above implies

$$\left(\frac{\partial I^{+}}{\partial d_{ji}} \right)_{n(k)} = \left(\frac{\partial I^{-}}{\partial d_{ji}} \right)_{n(k)}. \tag{51}$$

Turning to spike arrival times $t_k^{\text{event}} \in \mathcal{E} \backslash \mathcal{S}$, when the spike at t_k^{spike} arrives at the post-synaptic neurons m, we get

$$(I^{+})_m = (I^{-})_m + w_{mn(k)}, \tag{52}$$

and hence,

$$\left(\frac{\partial I^{+}}{\partial d_{ji}} \right)_m + (\dot{I}^{+})_m \frac{\mathrm{d}t_k^{\text{event}}}{\mathrm{d}d_{ji}} = \left(\frac{\partial I^{-}}{\partial d_{ji}} \right)_m + (\dot{I}^{-})_m \frac{\mathrm{d}t_k^{\text{event}}}{\mathrm{d}d_{ji}}. \tag{53}$$

Using the dynamics of I, (52) implies

$$\tau_{\text{s}} (\dot{I}^{+})_{m(l,n)} = \tau_{\text{s}} (\dot{I}^{-})_{m(l,n)} - w_{mn}, \tag{54}$$

and hence

$$\left(\frac{\partial I^+}{\partial d_{ji}}\right)_m = \left(\frac{\partial I^-}{\partial d_{ji}}\right)_m + \tau_\mathrm{s}^{-1} w_{mn(k)} \frac{\mathrm{d}t_k^\mathrm{event}}{\mathrm{d}d_{ji}} \tag{55}$$

$$= \left(\frac{\partial I^-}{\partial d_{ji}}\right)_m - \frac{1}{\tau_\mathrm{s}(\dot{V}^-)_{n(k)}} w_{mn(k)} \left(\frac{\partial V^-}{\partial d_{ji}}\right)_{n(k)} + \delta_{in(k)}\delta_{jm}\frac{w_{mn(k)}}{\tau_\mathrm{s}}, \tag{56}$$

where the term involving the spiking neuron $n(k)$ stems from the derivative of the spike time t_k^event with respect to d_{ji} using (48) and the last term from the derivative of the delay by itself. For the voltages,

$$\left(\frac{\partial V^+}{\partial d_{ji}}\right)_m + (\dot{V}^+)_m \frac{\mathrm{d}t_k^\mathrm{event}}{\mathrm{d}d_{ji}} = \left(\frac{\partial V^-}{\partial d_{ji}}\right)_m + (\dot{V}^-)_m \frac{\mathrm{d}t_k^\mathrm{event}}{\mathrm{d}d_{ji}}, \tag{57}$$

and using the dynamics of V and (52),

$$\tau_\mathrm{m}(\dot{V}^+)_m = \tau_\mathrm{m}(\dot{V}^-)_m + w_{mn(k)}, \tag{58}$$

which put together gives

$$\left(\frac{\partial V^+}{\partial d_{ji}}\right)_m = \left(\frac{\partial V^-}{\partial d_{ji}}\right)_m - \tau_\mathrm{m}^{-1} w_{mn} \frac{\mathrm{d}t_k^\mathrm{event}}{\mathrm{d}d_{ji}} \tag{59}$$

$$= \left(\frac{\partial V^-}{\partial d_{ji}}\right)_m + \frac{1}{\tau_\mathrm{m}(\dot{V}^-)_{n(k)}} w_{mn(k)} \left(\frac{\partial V^-}{\partial d_{ji}}\right)_{n(k)} - \delta_{in(k)}\delta_{jm}\frac{w_{mn(k)}}{\tau_\mathrm{m}}, \tag{60}$$

where the last term again arises from the derivative of the delay $d_{mn(k)}$ in t_k^event with respect to d_{ji}. Taking everything together we get

$$\frac{d\mathcal{L}}{dd_{ji}} = \sum_{t_k^\mathrm{spike}\in\mathcal{S}} \left[\left(\frac{\partial V^-}{\partial d_{ji}}\right)_n \left[\tau_\mathrm{m}\left(\lambda_V^- - \frac{(\dot{V}^+)_n}{(\dot{V}^-)_n}\lambda_V^+\right)_n + \frac{1}{(\dot{V}^-)_n}\left(-\frac{\partial l_p}{\partial t_k^\mathrm{spike}} + l_V^+ - l_V^-\right)\right] \right. \tag{61}$$

$$\left. + \tau_\mathrm{s}(\lambda_I^- - \lambda_I^+)_n \left(\frac{\partial I^-}{\partial d_{ji}}\right)_n \right]\Bigg|_{t_k^\mathrm{spike}} \tag{62}$$

$$+ \sum_m \left[\tau_\mathrm{m}(\lambda_V^- - \lambda_V^+)_m \left(\frac{\partial V^-}{\partial d_{ji}}\right)_m + \tau_\mathrm{s}(\lambda_I^- - \lambda_I^+)_m \left(\frac{\partial I^-}{\partial d_{ji}}\right)_m \right]\Bigg|_{t_k^\mathrm{spike}+d_{mn(k)}} \tag{63}$$

$$+ \left[\left(\frac{\partial V^-}{\partial d_{ji}}\right)_n \frac{1}{(\dot{V}^-)_n} \right]\Bigg|_{t_k^\mathrm{spike}} \left[w_{mn}(\lambda_I^+ - \lambda_V^+)_m \right]\Big|_{t_k^\mathrm{spike}+d_{mn(k)}} \tag{64}$$

$$- \left[w_{mn(k)}\delta_{in(k)}\delta_{jm}(\lambda_I^+ - \lambda_V^+)_m \right]\Big|_{t_k^\mathrm{spike}+d_{mn(k)}}. \tag{65}$$

So, using the usual trick

$$\frac{(\dot{V}^+)_{n(k)}}{(\dot{V}^-)_{n(k)}} = \frac{\vartheta}{\tau_\mathrm{m}(\dot{V}^-)_{n(k)}} + 1, \tag{66}$$

we again arrive at the same jump conditions as usual,

$$(\lambda_V^-)_{n(k)} = \left[(\lambda_V^+)_{n(k)} + \frac{1}{\tau_\mathrm{m}(\dot{V}^-)_{n(k)}} \left[\vartheta \cdot (\lambda_V^+)_{n(k)} + \frac{\partial l_p}{\partial t_k^\mathrm{spike}} + l_V^- - l_V^+ \right] \right]\Bigg|_{t_k^\mathrm{spike}} \tag{67}$$

$$+ \left[\frac{1}{\tau_\mathrm{m}(\dot{V}^-)_{n(k)}} \right]\Bigg|_{t_k^\mathrm{spike}} \sum_m w_{mn(k)} \left[(\lambda_V^+ - \lambda_I^+)_m \right]\Big|_{t_k^\mathrm{spike}+d_{mn(k)}}$$

$$(\lambda_V^-)_m = (\lambda_V^+)_m, \text{ if } m \neq n(k) \tag{68}$$

$$\lambda_I^- = \lambda_I^+, \tag{69}$$

but the gradient updates take the form

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}d_{ji}} = -\sum_{t_k^\mathrm{spike}\in\mathcal{S}} w_{ji}\delta_{in(k)} \left(\lambda_I - \lambda_V\right)_j\Big|_{t_k^\mathrm{spike}+d_{jn(k)}} = -w_{ji} \sum_{\substack{t_k^\mathrm{spike} \\ \text{from } i}} \left(\lambda_I - \lambda_V\right)_j\Big|_{t_k^\mathrm{spike}+d_{ji}}. \tag{70}$$

Table 2: Yin-yang parameters

Architecture	Feedforward
Number of hidden layers	1
Number of hidden neurons	30/25/20/15/10/5
Input-to-hidden weight init.	$\mathcal{N}(2.0,\ 0.78)$
Hidden-to-out weight init.	$\mathcal{N}(0.93,\ 0.1)$
Ff delay init.	$\mathcal{U}(0,0)$
DT	0.01
Weight LR	0.001
Weight LR schedule	$0.998 \times \textit{epoch}$
Delay LR init.	0.01
Delay LR schedule	$0.998 \times \textit{epoch}$

Table 3: SHD parameters

Architecture	Recurrent	Feedforward
Number of hidden layers	1	2
Number of hidden neurons	1024/512/256/128	1024/512/256/128
Spike reg. strength (layer-wise)	$5 \cdot 10^{-11}$	$5 \cdot 10^{-12}, 5 \cdot 10^{-11}$
Input-to-hidden weight init.	$\mathcal{N}(0.03,\ 0.01)$	$\mathcal{N}(0.03,\ 0.01)$
Ff hidden-to-hidden weight init.	✗	$\mathcal{N}(0.02,\ 0.03)$
Rec hidden-to-hidden weight init.	$\mathcal{N}(0.0,\ 0.02)$	✗
Hidden-to-out weight init.	$\mathcal{N}(0.02,\ 0.03)$	$\mathcal{N}(0.02,\ 0.03)$
Ff delay init.	$\mathcal{U}(0,150)$	$\mathcal{U}(0,100)$
Rec delay init.	$\mathcal{U}(0,0)$	✗
DT	1.0	1.0
Weight LR	0.001	0.001
Weight LR schedule	1.05^{batch}	1.05^{batch}
Delay LR init.	0.1	0.1
Delay LR schedule	✗	✗

4.1.3 Time-invariant mean squared error loss

Following [39], we use for the Yin-yang benchmark the time-invariant mean squared error loss of output spike times

$$\mathcal{L}_{\Delta\mathrm{MSE}} = \frac{1}{2} \sum_{i \neq c} \left(t_i - t_c - \Delta_t\right)^2 , \tag{71}$$

where c is the true class of the current input and t_i, t_c denote the first spike time in the respective output neurons. In the EventProp formalism, this is a spike-time dependent loss l_p and, therefore, drives jumps in λ_V^i in output neuron i at spike times t_k^{spike} in the backward pass (see Table 1) by

$$\frac{\partial l_p}{\partial t_k^{\mathrm{spike}}} = \begin{cases} (t_i - t_c - \Delta_t) & \text{if } n(k) = i, t_k^{\mathrm{spike}} = t_i, i \neq c \\ \sum_{i \neq c} -(t_i - t_c - \Delta_t) & \text{if } n(k) = c, t_k^{\mathrm{spike}} = t_c \\ 0 & \text{otherwise} \end{cases} \tag{72}$$

4.2 Implementation

We implemented all of our work in the mlGeNN framework [41, 40] to exploit the the efficiency of event-based learning. In all of our experiments, we used the parameters from previous EventProp work [21], apart from spike regularisation strengths, number of hidden layers and recurrent connections. We did not

Table 4: SSC parameters

Architecture	Recurrent	Feedforward
Number of hidden layers	1	2
Number of hidden neurons	1024/512/256/128/64	1024/512/256/128/64
Spike reg. strength (layer-wise)	$5 \cdot 10^{-11}$	$5 \cdot 10^{-12}, 5 \cdot 10^{-12}$
Input-to-hidden weight init.	$\mathcal{N}(0.03,\ 0.01)$	$\mathcal{N}(0.03,\ 0.01)$
Ff hidden-to-hidden weight init.	✗	$\mathcal{N}(0.02,\ 0.03)$
Rec hidden-to-hidden weight init.	$\mathcal{N}(0.0,\ 0.02)$	✗
Hidden-to-out weight init.	$\mathcal{N}(0.02,\ 0.3)$	$\mathcal{N}(0.02,\ 0.03)$
Ff delay init.	$\mathcal{U}(0, 150)$	$\mathcal{U}(0, 50)$
Rec delay init.	$\mathcal{U}(0, 0)$	✗
DT	1.0	1.0
Weight LR	0.001	0.001
Weight LR schedule	1.05^{batch}	1.05^{batch}
Delay LR init.	0.1	0.1
Delay LR schedule	✗	✗

Table 5: Braille reading parameters

Architecture	Recurrent	Feedforward
Number of hidden layers	1	2
Number of hidden neurons	1024/512/256/128/64	1024/512/256/128/64
Spike reg. strength (layer-wise)	$1 \cdot 10^{-10}$	$1 \cdot 10^{-10}, 1 \cdot 10^{-10}$
Input-to-hidden weight init.	$\mathcal{N}(0.03,\ 0.01)$	$\mathcal{N}(0.03,\ 0.01)$
Ff hidden-to-hidden weight init.	✗	$\mathcal{N}(0.02,\ 0.03)$
Rec hidden-to-hidden weight init.	$\mathcal{N}(0.0,\ 0.02)$	✗
Hidden-to-out weight init.	$\mathcal{N}(0.02,\ 0.03)$	$\mathcal{N}(0.02,\ 0.03)$
Ff delay init.	$\mathcal{U}(0, 0)$	$\mathcal{U}(0, 0)$
Rec delay init.	✗	✗
DT	4.0	4.0
Weight LR	0.0015	0.0015
Weight LR schedule	✗	✗
Delay LR init.	0.025	0.025
Delay LR schedule	✗	✗

implement heterogeneous and trainable time constants, so that the independent effect of delays would be more clear. For our experiments on the SHD and SSC datasets, we adopted the data augmentation approaches described by Nowotny, Turner, and Knight [21], which were designed to improve generalization. Specifically, we implemented the following augmentations:

- Input Shifting: We randomly shifted all inputs by a value within the range of (-40,40).
- Input Blending: We blended two inputs from the same class by aligning their centers of mass and randomly selecting spikes from each input with a probability of 0.5.

For SSC we only used the shift augmentation. For the Yin-Yang dataset we decreased the learning rate on both weights and delays at the end of each epoch. On SHD and SSC, we implemented an "ease-in" scheduler on the weight learning rate, starting from 0.001 times the learning rate, increasing it at the end of each batch, until it reached the final value. For our chosen hyperparameters, see Tables 2, 3, 4 and 5. GeNN already provided an efficient implementation of spike transmission with per-synapse delays [43] – allowing the EventProp forward pass to be implemented efficiently. However, the λ_V transitions in the backward pass require access to postsynaptic λ values with a per-synapse delay ($\left[(\lambda_V^+ - \lambda_I^+)_m\right]|_{t_k^{\text{spike}}+d_{mn(k)}}$ from Equation: 35). This required a small extension to GeNN's existing system for providing delayed access to postsynaptic variables from a synapse model [42] in order to enable it to use the per-synapse delays used for spike transmission in the forward pass.

Code availability

All experiments were carried out using the GeNN 5.1.0 [70] an mlGeNN 2.3.0 [71]. The latest versions of both libraries are also available at `https://github.com/genn-team/`. The code to train and evaluate the models described in this work are available at `https://github.com/mbalazs98/deventprop`

Data availability

The data underlying our results are available at `https://figshare.com/s/e4a041d66c93355f586a`.

Acknowledgments

This work was funded by the EPSRC (grants EP/V052241/1 and EP/S030964/1) and the EU (grant no. 945539). Additionally, BM was funded by a Leverhulme Trust studentship and by The Alan Turing Institute. Compute time was provided through Gauss Centre for Supercomputing (application numbers 21018, 30182 and 61883) and EPSRC (grant number EP/T022205/1) and local GPU hardware was provided by an NVIDIA hardware grant award.

References

[1] David Patterson, Joseph Gonzalez, Quoc Le, Chen Liang, Lluis-Miquel Munguia, Daniel Rothchild, David So, Maud Texier, and Jeff Dean. "Carbon emissions and large neural network training". In: *arXiv preprint arXiv:2104.10350* (2021).

[2] Louis Sokoloff. "The brain as a chemical machine". In: *Progress in brain research* 94 (1992), pp. 19–33.

[3] Catherine D Schuman, Thomas E Potok, Robert M Patton, J Douglas Birdwell, Mark E Dean, Garrett S Rose, and James S Plank. "A survey of neuromorphic computing and neural networks in hardware". In: *arXiv preprint arXiv:1705.06963* (2017).

[4] Conrad D James, James B Aimone, Nadine E Miner, Craig M Vineyard, Fredrick H Rothganger, Kristofor D Carlson, Samuel A Mulder, Timothy J Draelos, Aleksandra Faust, Matthew J Marinella, et al. "A historical survey of algorithms and hardware architectures for neural-inspired and neuromorphic computing applications". In: *Biologically Inspired Cognitive Architectures* 19 (2017), pp. 49–64.

[5] Chetan Singh Thakur, Jamal Lottier Molin, Gert Cauwenberghs, Giacomo Indiveri, Kundan Kumar, Ning Qiao, Johannes Schemmel, Runchun Wang, Elisabetta Chicca, Jennifer Olson Hasler, et al. "Large-scale neuromorphic spiking array processors: A quest to mimic the brain". In: *Frontiers in neuroscience* 12 (2018), p. 891.

[6] Abhronil Sengupta, Yuting Ye, Robert Wang, Chiao Liu, and Kaushik Roy. "Going deeper in spiking neural networks: VGG and residual architectures". In: *Frontiers in neuroscience* 13 (2019), p. 95.

[7] Rui-Jie Zhu, Qihang Zhao, Guoqi Li, and Jason Eshraghian. "SpikeGPT: Generative Pre-trained Language Model with Spiking Neural Networks". In: *Transactions on Machine Learning Research* (2024). ISSN: 2835-8856. URL: `https://openreview.net/forum?id=gcf1anBL9e`.

[8] Bodo Rueckauer, Iulia-Alexandra Lungu, Yuhuang Hu, Michael Pfeiffer, and Shih-Chii Liu. "Conversion of continuous-valued deep networks to efficient event-driven networks for image classification". In: *Frontiers in neuroscience* 11 (2017), p. 682.

[9] Ana Stanojevic, Stanisław Woźniak, Guillaume Bellec, Giovanni Cherubini, Angeliki Pantazi, and Wulfram Gerstner. "High-performance deep spiking neural networks with 0.3 spikes per neuron". en. In: *Nature Communications* 15.1 (Aug. 2024), p. 6793. ISSN: 2041-1723. DOI: 10.1038/s41467-024-51110-5. URL: https://www.nature.com/articles/s41467-024-51110-5 (visited on 08/19/2024).

[10] Christoph Stöckl and Wolfgang Maass. "Optimized spiking neurons can classify images with high accuracy through temporal coding with two spikes". In: *Nature Machine Intelligence* 3.3 (2021), pp. 230–238.

[11] Sander M. Bohte, Joost N. Kok, and Han La Poutré. "Error-backpropagation in temporally encoded networks of spiking neurons". en. In: *Neurocomputing* 48.1-4 (Oct. 2002), pp. 17–37. ISSN: 09252312. DOI: 10.1016/S0925-2312(01)00658-0. URL: https://linkinghub.elsevier.com/retrieve/pii/S0925231201006580 (visited on 09/08/2023).

[12] Emre O Neftci, Hesham Mostafa, and Friedemann Zenke. "Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks". In: *IEEE Signal Processing Magazine* 36.6 (2019), pp. 51–63.

[13] Julian Göltz, Laura Kriener, Andreas Baumbach, Sebastian Billaudelle, Oliver Breitwieser, Benjamin Cramer, Dominik Dold, Akos Ferenc Kungl, Walter Senn, Johannes Schemmel, et al. "Fast and energy-efficient neuromorphic deep learning with first-spike times". In: *Nature machine intelligence* 3.9 (2021), pp. 823–835.

[14] Iulia M Comsa, Krzysztof Potempa, Luca Versari, Thomas Fischbacher, Andrea Gesmundo, and Jyrki Alakuijala. "Temporal coding in spiking neural networks with alpha synaptic function". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 8529–8533.

[15] Paul I Barton and Cha Kun Lee. "Modeling, simulation, sensitivity analysis, and optimization of hybrid systems". In: *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 12.4 (2002), pp. 256–289.

[16] Kukan Selvaratnam. "Learning methods of recurrent spiking neural networks-transient and oscillatory spike trains". In: *Systems, control and information* 13.3 (2000), pp. 95–104.

[17] Dongsung Huh and Terrence J Sejnowski. "Gradient descent for spiking neural networks". In: *Advances in neural information processing systems* 31 (2018).

[18] Timo C Wunderlich and Christian Pehle. "Event-based backpropagation can compute exact gradients for spiking neural networks". In: *Scientific Reports* 11.1 (2021), p. 12829.

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[20] Laura Kriener, Julian Göltz, and Mihai A Petrovici. "The Yin-Yang dataset". In: *Proceedings of the 2022 Annual Neuro-Inspired Computational Elements Conference*. 2022, pp. 107–111.

[21] Thomas Nowotny, James Paul Turner, and James Courtney Knight. "Loss shaping enhances exact gradient learning with Eventprop in Spiking Neural Networks". In: *Neuromorphic Computing and Engineering* (2025). URL: http://iopscience.iop.org/article/10.1088/2634-4386/ada852.

[22] Benjamin Cramer, Yannik Stradmann, Johannes Schemmel, and Friedemann Zenke. "The Heidelberg spiking data sets for the systematic evaluation of spiking neural networks". In: *IEEE Transactions on Neural Networks and Learning Systems* 33.7 (2020), pp. 2744–2757.

[23] Christian Pehle, Luca Blessing, Elias Arnold, Eric Müller, and Johannes Schemmel. *Event-based Backpropagation for Analog Neuromorphic Hardware*. arXiv:2302.07141 [q-bio]. Feb. 2023. DOI: 10.48550/arXiv.2302.07141. URL: http://arxiv.org/abs/2302.07141 (visited on 12/17/2024).

[24] Gabriel Béna, Timo Wunderlich, Mahmoud Akl, Bernhard Vogginger, Christian Mayr, and Hector Andres Gonzalez. "Event-based backpropagation on the neuromorphic platform SpiNNaker2". In: *NeurIPS 2024 Workshop Machine Learning with new Compute Paradigms*.

[25] Wei Fang, Zhaofei Yu, Yanqi Chen, Timothée Masquelier, Tiejun Huang, and Yonghong Tian. "Incorporating learnable membrane time constant to enhance learning of spiking neural networks". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 2661–2671.

[26] Nicolas Perez-Nieves, Vincent CH Leung, Pier Luigi Dragotti, and Dan FM Goodman. "Neural heterogeneity promotes robust learning". In: *Nature communications* 12.1 (2021), p. 5791.

[27] Ilyass Hammouamri, Ismail Khalfaoui-Hassani, and Timothée Masquelier. "Learning Delays in Spiking Neural Networks using Dilated Convolutions with Learnable Spacings". In: *The Twelfth International Conference on Learning Representations.*

[28] Pengfei Sun, Yansong Chua, Paul Devos, and Dick Botteldooren. "Learnable axonal delay in spiking neural networks improves spoken word recognition". In: *Frontiers in Neuroscience* 17 (2023), p. 1275944.

[29] Armin H Seidl, Edwin W Rubel, and David M Harris. "Mechanisms for adjusting interaural time differences to achieve binaural coincidence detection". In: *Journal of Neuroscience* 30.1 (2010), pp. 70–80.

[30] Sara L Bengtsson, Zoltán Nagy, Stefan Skare, Lea Forsman, Hans Forssberg, and Fredrik Ullén. "Extensive piano practicing has regionally specific effects on white matter development". In: *Nature neuroscience* 8.9 (2005), pp. 1148–1150.

[31] Eugene M Izhikevich. "Polychronization: computation with spikes". In: *Neural computation* 18.2 (2006), pp. 245–282.

[32] Wolfgang Maass and Michael Schmitt. "On the complexity of learning for spiking neurons with temporal coding". In: *Information and Computation* 153.1 (1999), pp. 26–46.

[33] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. "The SpiNNaker project". In: *Proceedings of the IEEE* 102.5 (2014), pp. 652–665.

[34] Mike Davies, Narayan Srinivasa, Tsung-Han Lin, Gautham Chinya, Yongqiang Cao, Sri Harsha Choday, Georgios Dimou, Prasad Joshi, Nabil Imam, Shweta Jain, et al. "Loihi: A neuromorphic manycore processor with on-chip learning". In: *Ieee Micro* 38.1 (2018), pp. 82–99.

[35] Lucas Deckers, Laurens Van Damme, Werner Van Leekwijck, Ing Jyh Tsang, and Steven Latré. "Co-learning synaptic delays, weights and adaptation in spiking neural networks". In: *Frontiers in Neuroscience* 18 (2024), p. 1360300.

[36] Edoardo Grappolini and Anand Subramoney. "Beyond weights: deep learning in spiking neural networks with pure synaptic-delay training". In: *Proceedings of the 2023 International Conference on Neuromorphic Systems.* 2023, pp. 1–4.

[37] Sumit B Shrestha and Garrick Orchard. "Slayer: Spike layer error reassignment in time". In: *Advances in neural information processing systems* 31 (2018).

[38] Xiangwen Wang, Xianghong Lin, and Xiaochao Dang. "A delay learning algorithm based on spike train kernels for spiking neurons". In: *Frontiers in neuroscience* 13 (2019), p. 252.

[39] Julian Göltz, Jimmy Weber, Laura Kriener, Peter Lake, Melika Payvand, and Mihai A Petrovici. "DelGrad: Exact gradients in spiking networks for learning transmission delays and weights". In: *arXiv preprint arXiv:2404.19165* (2024).

[40] James Paul Turner, James C Knight, Ajay Subramanian, and Thomas Nowotny. "mlGeNN: accelerating SNN inference using GPU-enabled neural networks". In: *Neuromorphic Computing and Engineering* 2.2 (June 2022). Publisher: IOP Publishing, p. 024002. ISSN: 2634-4386. DOI: 10.1088/2634-4386/ac5ac5. URL: https://iopscience.iop.org/article/10.1088/2634-4386/ac5ac5.

[41] James C Knight and Thomas Nowotny. "Easy and efficient spike-based Machine Learning with mlGeNN". In: *Proceedings of the 2023 Annual Neuro-Inspired Computational Elements Conference.* 2023, pp. 115–120.

[42] Esin Yavuz, James Turner, and Thomas Nowotny. "GeNN: a code generation framework for accelerated brain simulations". In: *Scientific Reports* 6.1 (May 7, 2016). Publisher: Nature Publishing Group, p. 18854. ISSN: 2045-2322. DOI: 10.1038/srep18854. URL: https://www.nature.com/articles/srep18854.

[43] James C. Knight and Thomas Nowotny. "GPUs Outperform Current HPC and Neuromorphic Solutions in Terms of Speed and Energy When Simulating a Highly-Connected Cortical Model". In: *Frontiers in Neuroscience* 12 (December 2018), pp. 1–19. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00941. URL: https://www.frontiersin.org/article/10.3389/fnins.2018.00941/full.

[44] James C Knight, Anton Komissarov, and Thomas Nowotny. "PyGeNN: a Python library for GPU-enhanced neural networks". In: *Frontiers in Neuroinformatics* 15 (2021), p. 659005.

[45] Romain Brette. "Event-driven simulation of integrate-and-fire neurons with exponential synaptic conductances". In: ().

[46] Romain Brette. "Exact Simulation of Integrate-and-Fire Models with Exponential Currents". In: *Neural Computation* 19.10 (Oct. 2007), pp. 2604–2609. ISSN: 0899-7667, 1530-888X. DOI: `10.1162/neco.2007.19.10.2604`. URL: `https://direct.mit.edu/neco/article/19/10/2604-2609/7220` (visited on 06/13/2025).

[47] Romain Brette, Michelle Rudolph, Ted Carnevale, Michael Hines, David Beeman, James M Bower, Markus Diesmann, Abigail Morrison, Philip H Goodman, Frederick C Harris, Milind Zirpe, Thomas Natschläger, Dejan Pecevski, Bard Ermentrout, Mikael Djurfeldt, Anders Lansner, Olivier Rochel, Thierry Vieville, Eilif Muller, Andrew P Davison, Sami El Boustani, and Alain Destexhe. "Simulation of networks of spiking neurons: A review of tools and strategies". In: *Journal of Computational Neuroscience* 23.3 (Dec. 12, 2007), pp. 349–398. ISSN: 1573-6873. DOI: `10.1007/s10827-007-0038-6`. URL: `http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2638500&tool=pmcentrez&rendertype=abstract` (visited on 07/11/2014).

[48] Abigail Morrison, Carsten Mehring, Theo Geisel, a D Aertsen, and Markus Diesmann. "Advancing the boundaries of high-connectivity network simulation with distributed computing." In: *Neural computation* 17.8 (Aug. 2005), pp. 1776–801. ISSN: 0899-7667. DOI: `10.1162/0899766054026648`. URL: `http://www.ncbi.nlm.nih.gov/pubmed/15969917`.

[49] Santos Galán, William F Feehery, and Paul I Barton. "Parametric sensitivity functions for hybrid discrete/continuous systems". In: *Applied Numerical Mathematics* 31.1 (1999), pp. 17–47.

[50] E. N. Rozenvasser. "General sensitivity equations of discontinuous systems". In: *Avtomat. i Telemekh.* (3 1967), pp. 52–56.

[51] Christian-Gernot Pehle. "Adjoint equations of spiking neural networks". PhD thesis. Heidelberg University, 2021.

[52] Maximilian Baronig, Romain Ferrand, Silvester Sabathiel, and Robert Legenstein. "Advancing Spatio-Temporal Processing in Spiking Neural Networks through Adaptation". In: *arXiv preprint arXiv:2408.07517* (2024).

[53] Pengfei Sun, Jibin Wu, Paul Devos, and Dick Botteldooren. "Towards parameter-free attentional spiking neural networks". In: *Neural Networks* 185 (2025), p. 107154.

[54] Mark Schöne, Neeraj Mohan Sushma, Jingyue Zhuge, Christian Mayr, Anand Subramoney, and David Kappel. "Scalable event-by-event processing of neuromorphic sensory signals with deep state-space models". In: *2024 International Conference on Neuromorphic Systems (ICONS)*. IEEE. 2024, pp. 124–131.

[55] Anders Isaksson, Mikael Wallman, Hanna Göransson, and Mats G Gustafsson. "Cross-validation and bootstrapping are unreliable in small sample classification". In: *Pattern Recognition Letters* 29.14 (2008), pp. 1960–1965.

[56] Thomas Nowotny. "Two challenges of correct validation in pattern recognition". In: *Frontiers in Robotics and AI* 1 (2014), p. 5.

[57] Erik Sadovsky, Maros Jakubec, and Roman Jarina. "Speech command recognition based on convolutional spiking neural networks". In: *2023 33rd International Conference Radioelektronika (RADIOELEKTRONIKA)*. IEEE. 2023, pp. 1–5.

[58] Alexandre Bittar and Philip N Garner. "A surrogate gradient spiking baseline for speech command recognition". In: *Frontiers in Neuroscience* 16 (2022), p. 865897.

[59] Simon F Müller-Cleve, Vittorio Fra, Lyes Khacef, Alejandro Pequeño-Zurro, Daniel Klepatsch, Evelina Forno, Diego G Ivanovich, Shavika Rastogi, Gianvito Urgese, Friedemann Zenke, et al. "Braille letter reading: A benchmark for spatio-temporal pattern recognition on neuromorphic hardware". In: *Frontiers in Neuroscience* 16 (2022), p. 951164.

[60] Ben Walters, Yeshwanth Bethi, Taylor Kergan, Binh Nguyen, Amirali Amirsoleimani, Jason K Eshraghian, Saeed Afshar, and Mostafa Rahimi Azghadi. "NeuroMorse: a temporally structured dataset for neuromorphic computing". In: *Neuromorphic Computing and Engineering* (2025).

[61] Jens E Pedersen, Steven Abreu, Matthias Jobst, Gregor Lenz, Vittorio Fra, Felix Christian Bauer, Dylan Richard Muir, Peng Zhou, Bernhard Vogginger, Kade Heckel, et al. "Neuromorphic intermediate representation: A unified instruction set for interoperable brain-inspired computing". In: *Nature Communications* 15.1 (2024), p. 8122.

[62] Hannah Bos and Dylan Muir. "Sub-mw neuromorphic snn audio processing applications with rockpool and xylo". In: *Embedded Artificial Intelligence*. River Publishers, 2023, pp. 69–78.

[63] Wei Fang, Yanqi Chen, Jianhao Ding, Zhaofei Yu, Timothée Masquelier, Ding Chen, Liwei Huang, Huihui Zhou, Guoqi Li, and Yonghong Tian. "SpikingJelly: An open-source machine learning infrastructure platform for spike-based intelligence". In: *Science Advances* 9.40 (2023), eadi1480. DOI: 10.1126/sciadv.adi1480. eprint: https://www.science.org/doi/pdf/10.1126/sciadv.adi1480. URL: https://www.science.org/doi/abs/10.1126/sciadv.adi1480.

[64] Balázs Mészáros, James C Knight, and Thomas Nowotny. "Learning Delays Through Gradients and Structure: Emergence of Spatiotemporal Patterns in Spiking Neural Networks". In: *Frontiers in Computational Neuroscience* 18 (), p. 1460309.

[65] Harvey A Swadlow. "Physiological properties of individual cerebral axons studied in vivo for as long as one year". In: *Journal of neurophysiology* 54.5 (1985), pp. 1346–1362.

[66] Filippo Moro, Pau Vilimelis Aceituno, Laura Kriener, and Melika Payvand. "The Role of Temporal Hierarchy in Spiking Neural Networks". In: *arXiv preprint arXiv:2407.18838* (2024).

[67] Lloyd A Jeffress. "A place theory of sound localization." In: *Journal of comparative and physiological psychology* 41.1 (1948), p. 35.

[68] Antoine Grimaldi and Laurent U Perrinet. "Learning hetero-synaptic delays for motion detection in a single layer of spiking neurons". In: *2022 IEEE International Conference on Image Processing (ICIP)*. IEEE. 2022, pp. 3591–3595.

[69] Antoine Grimaldi and Laurent U Perrinet. "Learning heterogeneous delays in a layer of spiking neurons for fast motion detection". In: *Biological Cybernetics* 117.4 (2023), pp. 373–387.

[70] James Knight, Thomas Nowotny, James Paul Turner, Esin Yavuz, Fawad Ali, Mengchi Zhang, Anton Komissarov, Ben Evans, Garibaldi Pineda-Garcia, Kanishk Kalra, Alan Diamond, Obaid Ur Rehman, Christoph Ostrau, Alex Cope, Ajay Subramanian, Alex Dewar, Marcel Stimberg, Felix Benjamin Kern, FabianSchubert, Lev E. Givon, Edward Stevinson, Xilin Huang, Anindya Ghosh, and Edward Stevinson. *genn-team/genn: GeNN 5.1.0*. Version 5.1.0. Nov. 2024. DOI: 10.5281/zenodo.14051978. URL: https://doi.org/10.5281/zenodo.14051978.

[71] James Knight, James Paul Turner, Ajay Subramanian, Thomas Nowotny, Isabella Forero, Balázs Mészáros, Adrian D'Alessandro, Jorge Felipe Gaviria, and FabianSchubert. *genn-team/ml_genn: ml_genn_2_3_0*. Version ml_genn_2_3_0. Dec. 2024. DOI: 10.5281/zenodo.14258972. URL: https://doi.org/10.5281/zenodo.14258972.