Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets

Guillaume Bellec*, Franz Scherr*, Elias Hajek , Darjan Salaj , Robert Legenstein , and Wolfgang Maass

Institute for Theoretical Computer Science, Graz University of Technology, Austria

January 25, 2019

*First authors

Abstract

The way how recurrently connected networks of spiking neurons in the brain acquire powerful information processing capabilities through learning has remained a mystery. This lack of understanding is linked to a lack of learning algorithms for recurrent networks of spiking neurons (RSNNs) that are both functionally powerful and can be implemented by known biological mechanisms. Since RSNNs are simultaneously a primary target for implementations of brain-inspired circuits in neuromorphic hardware, this lack of algorithmic insight also hinders technological progress in that area. The gold standard for learning in recurrent neural networks in machine learning is back-propagation through time (BPTT), which implements stochastic gradient descent with regard to a given loss function. But BPTT is unrealistic from a biological perspective, since it requires a transmission of error signals backwards in time and in space, i.e., from post- to presynaptic neurons. We show that an online merging of locally available information during a computation with suitable top-down learning signals in real-time provides highly capable approximations to BPTT. For tasks where information on errors arises only late during a network computation, we enrich locally available information through feedforward eligibility traces of synapses that can easily be computed in an online manner. The resulting new generation of learning algorithms for recurrent neural networks provides a new understanding of network learning in the brain that can be tested experimentally. In addition, these algorithms provide efficient methods for on-chip training of RSNNs in neuromorphic hardware.

Introduction

A characteristic property of networks of neurons in the brain is that they are recurrently connected: „the brain is essentially a multitude of superimposed and ever-growing loops between the input from the environment and the brain's outputs“ (Buzsaki, 2006). In fact, already (Lorente de Nó, 1938) had proposed that synaptic loops were the basic circuits of the central nervous system, and a large body of experimental work supports this view (Kandel et al., 2000). Recurrent loops of synaptic connections occur both locally within a lamina of a cortical microcircuit, between their laminae, between patches of neural tissue within the same brain area, and between different brain areas. Hence the architecture of neural networks in the brain is fundamentally different from that of feedforward deep neural network models that have gained high attention because of their astounding capability to classify real-world images as well as humans (LeCun et al., 2015). Actually, more recent work has shown that recurrently connected network can solve the same task with fewer parameters (Nayebi et al., 2018).

Recurrently connected neural networks tend to provide functionally superior neural network architectures for tasks that involve a temporal dimension, such as video prediction, gesture recognition, speech recognition, or motor control. Since the brain has to solve similar tasks, and even transforms image recognition into a temporal task via eye-movements, there is a clear functional reason why the brain employs recurrently connected neural networks. In addition, recurrent networks enable the brain to engage memory on several temporal scales, and to represent and continuously update internal states as well as goals. Furthermore the brain is a powerful prediction machine that learns through self-supervised learning to predict the consequences of its actions and of external events. In fact, predictions provide the brain with a powerful strategy for compensating the relative slowness of its sensory feedback.

The computational function of recurrently connected neural networks in the brain arises from a combination of nature and nurture that has remained opaque. In particular, it has remained a mystery how recurrent networks of spiking neurons (RSNNs) can learn. Recurrent networks of artificial neurons are commonly trained in machine learning through BPTT. BPTT can not only be used to implement supervised learning, but – with a suitably defined loss function E – self-supervised, unsupervised and reward based learning. Unfortunately BPTT requires a physically unrealistic propagation of error signals backwards in time. This feature also thwarts an efficient implementation in neuromorphic hardware. It even hinders an efficient implementation of BP or BPTT on GPUs and other standard computing hardware: „backpropagation results in locking – the weights of a network module can only be updated after a full forward propagation of data, followed by loss evaluation, and then finally after waiting for the backpropagation of error gradients“ (Czarnecki et al., 2017). Locking is an issue of particular relevance for applications of BPTT to recurrent neural networks, since this amounts to applications of backpropagation to the unrolled recurrent network, which easily becomes several thousands of layers deep.

We show that BPTT can be represented by a sum of products based on a new factorization or errors gradients with regards to the synaptic weights θ_{ji}. The error gradient is represented here as a sum over t of an eligibility trace $\mathbf{e}_{ji}^t$ until time t - which is independent from error signals - and a learning signal $\mathbf{L}_j^t$ that reaches this synapse at time t, see equation (1). This can be interpreted as on online ***merging*** for every time step t of eligibility traces and learning signals.

The key problem for achieving good learning results with eligibility traces is the online production of suitable learning signals that gate the update of the synaptic weight at time t. In order to achieve the full learning power of BPTT, this learning signal would still have to be complex and questionable from a biological perspective. But several biologically plausible approximations of such online learning signals turn out to work surprisingly well, especially for tasks that recurrent networks of neurons in the brain need to solve. We refer to the resulting learning strategies as *merge* algorithms, because of the central role that online merging of currently available bottom-up and top-down information plays in them.

There exists an abundance of experimental data on learning- or error signals in the brain. A rich literature documents the error-related negativity (ERN) that is recorded by EEG-electrodes from the human brain. The ERN has the form of a sharp negative-going deflection that accompanies behavioral errors, for example in motor control. Remarkable is that the ERN appears very fast, even before direct evidence of a behavioral error becomes accessible through sensory feedback (see e.g. Fig. 4 in (MacLean et al., 2015)), suggesting that it employs an internal error prediction network. Furthermore the amplitude of the ERN correlates with improved performance on subsequent trials ((Gehring et al., 1993), see also the review in (Buzzell et al., 2017)). These results suggest that the ERN is in fact a signal that gates learning. The data of (Buzzell et al., 2017) also shows that the ERN is generated by a distributed system of brain areas, in which posterior cingulate cortex, dorsal anterior cingulate, and parietal cortex assume dominant roles from early stages of development on. Furthermore, error-related activity from additional brain areas – insula, orbitofrontal cortex, and inferior frontal gyrus – increases with age. These experimental data suggest that error signals in the human brain are partially innate, but are complemented and refined during development.

The precise way how these error signals gate synaptic plasticity in the brain is unknown. One conjectured mechanism involves top-down disinhibition of dendrites and neurons, e.g. by activating VIP-interneurons in layer 1, which inhibit somatostatin-positive (SOM+) inhibitory neurons. Hence the activation of VIP neurons temporarily removes the inhibitory lock which

SOM+ neurons hold on activity and plasticity in distal dendrites of pyramidal cells (Pi et al., 2013). Another pathway for top-down regulation of synaptic plasticity involves cholinergic activation of astrocytes (Sugihara et al., 2016). Furthermore the cerebellum is known to play a prominent role in the processing of error signals and gating plasticity (D'Angelo et al., 2016) Most prominently, the neuromodulator dopamine plays an essential role in the control of learning, in particular also for learning of motor skills (Hosp et al., 2011). Experimental data verify that neuromodulators interact with local eligibility traces in gating synaptic plasticity, see (Gerstner et al., 2018) for a review. Of interest for the context of this paper is also the recent discovery that dopaminergic neurons in the mid-brain do not emit a uniform global signal, but rather a multitude of spatially organized signals for different populations of neurons (Engelhard et al., 2018). This distributed architecture of the error-monitoring system in the brain is consistent with the assumption that local populations of neurons receive different learning signals that have been shaped during evolution and development.

Error signals in the brain are from the functional perspective reminiscent of error signals that have turned out to alleviate the need for backprogation of error signals in feedforward neural networks. A particularly interesting version of such signals is called broadcast alignment (BA) in (Samadi et al., 2017) and direct feedback alignment in (Nøkland, 2016). These error signals are sent directly from the output stage of the network to each layer of the feedforward network. If one applies this broadcast alignment idea to the unrolled feedforward version of a recurrent network, one still runs into the problem that an error broadcast to an earlier time-slice or layer would have to go backwards in time. We present a simple method where this can be avoided, which we call *merge #1*.

Besides BA we explore in this paper two other methods for generating learning signals that provide – in combination with eligibility traces – powerful alternatives to BPTT. In *merge #2* we apply the Learning-to-Learn (L2L) framework to train separate neural networks – called error modules – to produce suitable learning signals for large families of learning tasks. But in contrast to the L2L approach of (Wang et al., 2016) and (Duan et al., 2016) we allow the recurrent neural network to modify its synaptic weights for learning a particular task. Only the synaptic weights within the error module are determined on the larger time scale of the outer loop of L2L (see the scheme in Figure 3). We show that this approach opens new doors for learning in recurrent networks of spiking neurons, enabling for example one-shot learning of pattern generation. Our third method, *merge #3*, employs the synthetic gradient approach of (Jaderberg et al., 2016) and (Czarnecki et al., 2017). We show that eligibility traces substantially enhance the power of synthetic gradients, surpassing in some cases even the performance of full BPTT for artificial neural networks. Altogether the *merge* approach suggests that a rich reservoir of algorithmic improvements of network learning waits to be discovered, where one employs dedicated modules and processes for generating learning signals that enable learning without backpropagated error signals. In addition this research is likely to throw light on the functional role of the complex distributed architecture of brain areas that are involved in the generation of learning signals in the brain.

These *merge* algorithms have an attractive feature from the theoretical perspective: They can be viewed – and analyzed – as approximations to a theoretically ideal: stochastic gradient descent, or BPTT. Merge algorithms are also of particular interest from the perspective of understanding learning in RSNNs of the brain. They tend to provide better learning performance for RSNNs than previously known methods. In addition, in contrast to most of the previously used methods, they do not require biologically unrealistic ingredients. In fact, it turns out that network learning with *merge* provides a novel understanding of refined STDP rules (Clopath et al., 2010) from a network learning perspective, that had been proposed in order to fit detailed experimental data on local synaptic plasticity mechanisms (Ngezahayo et al., 2000; Sjöström et al., 2001; Nevian and Sakmann, 2006).

In addition, *merge* provides a promising new approach for implementing on-chip learning in RSNNs that are implemented in neuromorphic hardware, such as Brainscales (Schemmel et al., 2010), SpiNNaker (Furber et al., 2014) and Loihi (Davies et al., 2018). Backpropagation of error signals in time as well as locking are formidable obstacles for an efficient implementation of BPTT on a neuromorphic chip. These obstacles are alleviated by the *merge* method.

Synaptic plasticity algorithms involving eligibility traces and gating factors have been reviewed for reinforcement learning in (Frémaux and Gerstner, 2016), see (Gerstner et al., 2018) for their relationships to data. We re-define eligibility traces for the different context

of gradient descent learning. Eligibility traces are used in classical reinforcement learning theory (Sutton and Barto, 1998) to relate the (recent) history of network activity to later rewards. This reinforcement learning theory inspired our approach on a conceptual level, but the details of the mathematical analysis become quite different, since we address a different problem: how to approximate error gradients in recurrent neural networks.

We will derive in the first section of Results the basic factorization equation (1) that underlies our *merge* approach. We then discuss applications of *merge #1* to RSNNs with a simple BA-like learning signal. In the subsequent section we will show how an application of L2L in *merge #2* can improve the learning capability of RSNNs. Finally, we show in the last subsection on *merge #3* that adding eligibility traces to the synthetic gradient approach of (Jaderberg et al., 2016) and (Czarnecki et al., 2017) improves learning also for recurrent networks of artificial neurons.

Results

Network models The learning rules that we describe can be applied to a variety of recurrent neural network models: Standard RSNNs consisting of leaky integrate-and-fire (LIF) neurons, LSNNs that also contain adaptive spiking neurons (Bellec et al., 2018), and networks of LSTM (long short-term memory) units (Hochreiter and Schmidhuber, 1997). LSNN (Long short term Spiking Neural Networks) were introduced to capture parts of the function of LSTM network in biologically motivated neural network models. In order to elucidate the link to biology, we focus in the first two variants of *merge* on the LIF neuron model (see Figures 1, 2 and 3). The LIF model is a simplified model of biological neurons: each neuron integrates incoming currents into its membrane potential, and as soon as the membrane potential crosses a threshold from below, the neuron "spikes" and a current is sent to subsequent neurons. Mathematically, the membrane potential is a leaky integrator of a weighted sum of the input currents, and the spike is a binary variable that becomes non-zero when a spike occurs (see equation (18) and (19) in Methods). To enrich the temporal processing capability of the network (see Figure 2), a portion of the neurons in an LSNN have adaptive firing thresholds. The dynamics of the adaptive thresholds is defined in equation (23) in Methods. We also applied a third variant of *merge #3* to LSTM networks to show that *merge* algorithms can be competitive on machine learning benchmarks (see Figure 4).

To describe the common core of these *merge* algorithms, all network models are subsumed under a general formalism. We assume that each neuron j is at time t in an internal state $\mathbf{s}_j^t \in \mathbb{R}^d$ and emits an observable state z_j^t. We also assume that $\mathbf{s}_j^t$ depends on the other neurons only through the vector $\mathbf{z}^{t-1}$ of observable states of all neurons in the network. Then, the network dynamics takes for some functions M and f the form: $\mathbf{s}_j^t = M(\mathbf{s}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{\theta})$ and $z_j^t = f(\mathbf{s}_j^t)$, where $\boldsymbol{\theta}$ is the vector of model parameters (in the models considered here, synaptic weights). For instance for LIF neurons with adaptive thresholds, the internal state $\mathbf{s}_j^t$ of neuron j is a vector of size $d = 2$ formed by the membrane voltage and the adaptive firing threshold, and the observable state $z_j^t \in \{0, 1\}$ indicates whether the neuron spikes at time t. The definition of the functions M and f defining the neuron dynamics for this model are given by equations (18),(19) and (23) in Methods and illustrated in Figure 5.

Mathematical framework for *merge* algorithms The fundamental mathematical law that enables the *merge* approach is that the gradients of BPTT can be factorized as a sum of products between learning signals $\mathbf{L}_j^t$ and eligibility traces $\mathbf{e}_{ji}^t$. We subsume here under the term eligibility trace that information which is locally available at a synapse and does not depend on network performance. The online learning signals $\mathbf{L}_j^t$ are provided externally and could for example quantify how spiking at the current time influences current and future errors. The general goal is to approximate the gradients of the network error function E with respect to the model parameters θ_{ji}. If the error function E depends exclusively on the network spikes $E(\mathbf{z}^0, \ldots, \mathbf{z}^T)$, the fundamental observation for *merge* is that the gradient with respect to the weights can be factorized as follows (see Methods for a proof):

$$\frac{dE}{d\theta_{ji}} = \sum_t \mathbf{L}_j^t \cdot \mathbf{e}_{ji}^t \ , \tag{1}$$

where $\cdot$ is the dot product. We refer to $\mathbf{L}_j^t$ and $\mathbf{e}_{ji}^t$ as the learning signals and eligibility traces

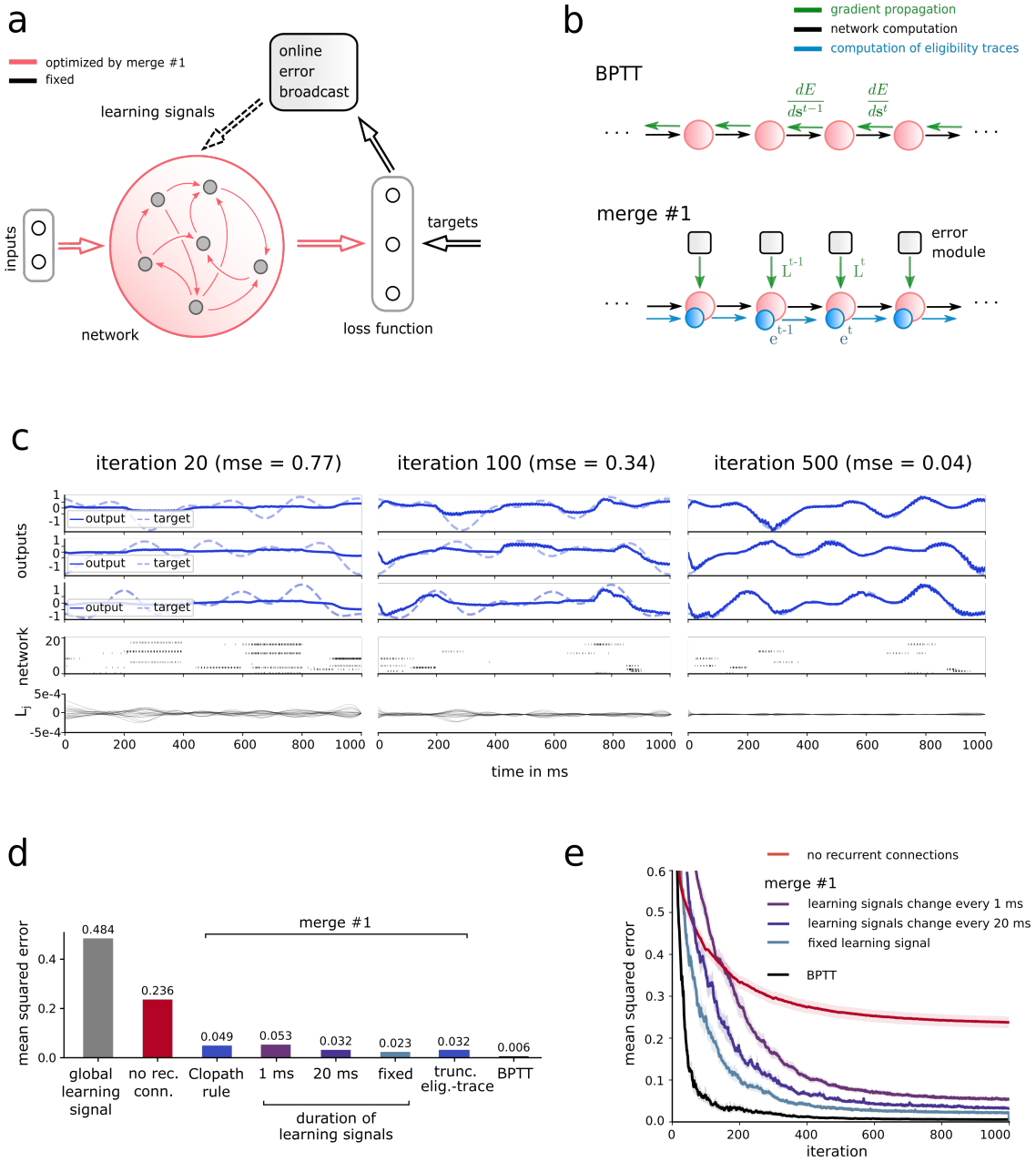

Figure 1: **Scheme and performance of *merge #1*** **a**) Learning architecture for *merge #1*. The error module at the top sends online error signals with random weights to the network that learns. **b**) Temporal dynamics of information flows in BPTT and *merge* algorithms. The propagation of error signals backwards in time of BPTT is replaced in *merge* algorithms by an additional computation that runs forward in time: the computation of eligibility traces. **c**) Evaluation of *merge #1* for a classical benchmark task for learning in recurrent SNNs: Learning to generate a target pattern, extended here to the challenge to simultaneously learn to generate 3 different patterns, which makes credit assignment for errors more difficult. **d**) Mean squared error of several learning algorithms for this task. "Clopath rule" denotes a replacement of the resulting synaptic plasticity rule of *merge #1* by the rule proposed in (Clopath et al., 2010) based on experimental data. **e**) Evolution of the mean squared error during learning.

respectively, see below for a definition. Note that we use distinct notation for the partial derivative $\frac{\partial E(\mathbf{z}_0,\ldots,\mathbf{z}_T)}{\partial \mathbf{z}_0}$, which is the derivative of the mathematical function E with respect to its first variable $\mathbf{z}_0$, and the total derivative $\frac{dE(\mathbf{z}_0,\ldots,\mathbf{z}_T)}{d\mathbf{z}_0}$ which also takes into account how E depends on $\mathbf{z}_0$ indirectly through the other variables $\mathbf{z}_1,\ldots,\mathbf{z}_T$. The essential term for gradient descent learning is the total derivative $\frac{dE}{d\theta_{ji}}$. It has usually been computed with BPTT (Werbos, 1990) or RTRL (Williams and Zipser, 1989).

Eligibility traces The intuition for eligibility traces is that a synapse remembers some of its activation history while ignoring inter neuron dependencies. Since the network dynamics is formalized through the equation $\mathbf{s}_j^t = M(\mathbf{s}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{\theta})$, the internal neuron dynamics isolated from the rest of the network is described by $D_j^{t-1} = \frac{\partial M}{\partial \mathbf{s}_j^{t-1}}(\mathbf{s}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{\theta}) \in \mathbb{R}^{d\times d}$ (recall that d, is the dimension of internal state of a single neuron; $d = 1$ or 2 in this paper). Considering the partial derivative of the state with respect to the synaptic weight $\frac{\partial M}{\partial \boldsymbol{\theta}_{ji}}(\mathbf{s}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{\theta}) \in \mathbb{R}^d$ (written $\frac{\partial \mathbf{s}_j^t}{\partial \theta_{ji}}$ for simplicity), the eligibility traces are defined by the iterative formula:

$$\mathbf{e}_{ji}^t = D_j^{t-1} \cdot \mathbf{e}_{ji}^{t-1} + \frac{\partial \mathbf{s}_j^t}{\partial \theta_{ji}} \ . \tag{2}$$

This eligibility trace amounts for LIF neurons just to a low-pass filtered trace of presynaptic spikes, see equation (20) in Methods. For the cases we have examined, it was sufficient to use an even simpler truncated version of this eligibility trace, that has value zero unless the presynaptic neuron spikes at the preceding time step. For adaptive neurons in LSNNs and for LSTM units the computation of eligibility traces becomes less trivial, see equations (24) and (33). But they can still be computed in an online manner forward in time, along with the network computation. We show later that this term holds a crucial role when working memory has to be engaged in the tasks to be learnt.

Note that in RTRL for networks of rate-based (sigmoidal) neurons (Williams and Zipser, 1989), the error gradients are computed forward in time by multiplying the full Jacobian $\mathbf{J}$ of the network dynamics with the tensor $\frac{d\mathbf{s}_k^t}{d\theta_{ji}}$ that computes the dependency of the state variables with respect to the parameters: $\frac{d\mathbf{s}_k^t}{d\theta_{ji}} = \sum_{k'} \mathbf{J}_{kk'}^t \cdot \frac{d\mathbf{s}_{k'}^{t-1}}{d\theta_{ji}} + \frac{\partial \mathbf{s}_k^t}{\partial \theta_{ji}}$ (see equation (12) in (Williams and Zipser, 1989)). Denoting with n the number of neurons, this requires $O(n^4)$ multiplications, which is computationally prohibitive in simulations, whereas BPTT or network simulation requires only $O(n^2)$ multiplications. In *merge*, the eligibility traces are $n \times n$ matrices, and D_j^t are $d \times d$ matrices which are the restriction of the full Jacobian $\mathbf{J}$ to the neuron specific dynamics. As a consequence, only $O(n^2)$ multiplications are required in the forward propagation of eligibility traces, their computation is not more costly than BPTT or simply simulating the network.

Theoretically ideal learning signals To satisfy equation (1), the learning signals $\mathbf{L}_j^t \in \mathbb{R}^d$ can be defined by the following formula, where we write $\frac{\partial z_j^t}{\partial \mathbf{s}_j^t}$ for $\frac{\partial f}{\partial \mathbf{s}_j^t}(\mathbf{s}_j^t)$:

$$\mathbf{L}_j^t \stackrel{\text{def}}{=} \frac{dE}{dz_j^t} \frac{\partial z_j^t}{\partial \mathbf{s}_j^t} \ . \tag{3}$$

Recall that $\frac{dE}{dz_j^t}$ is a total derivative and quantifies how much a current change of spiking activity might influence future errors. As a consequence, a direct computation of the term $\mathbf{L}_j^t$ needs to back-propagate gradients from the future as in BPTT. However we show that *merge* tends to work well if the ideal term $\mathbf{L}_j^t$ is replaced by an online approximation $\widehat{\mathbf{L}}_j^t$. In the following three sections we consider three concrete approximation methods, that define three variants of *merge*. If not clearly stated otherwise, all the resulting gradient estimations described below can be computed online and depend only on quantities accessible within the neuron j or the synapse $i \to j$ at the current time t.

Synaptic plasticity rules that emerge from this approach The learning algorithms *merge #1* and *merge #2* that will be discussed in the following are for networks of

spiking neurons. Resulting local learning rules are very similar to previously proposed and experimentally supported synaptic plasticity rules. They have the general form (learning signal) × (postsynaptic term) × (presynaptic term) as previously proposed 3-factor learning rules (Frémaux and Gerstner, 2016; Gerstner et al., 2018). The general form is given in equation (21), where p_j^t denotes a postsynaptic term and the last factor $\hat{z}_i^{t-1}$ denotes the presynaptic term (20). These last two terms are similar to the corresponding terms in the plasticity rule of (Clopath et al., 2010). It is shown in Figure 1d that one gets very similar results if one replaces the rule (21) that emerges from our approach by the Clopath rule.

The version (4) of this plasticity rule for *merge #1* contains the specific learning signal that arises in broadcast alignment as first factor. For synaptic plasticity of adapting neurons in LSNNs the last term, the eligibility trace, becomes a bit more complex because it accumulates information over a longer time span, see equation (24). The resulting synaptic plasticity rules for LSTM networks are given by equation (33).

Merge #1: Learning signals that arise from broadcast alignment

A breakthrough result for learning in feedforward deep neural networks was the discovery that a substantial portion of the learning power of backprop can be captured if the backpropagation of error signals through chains of layers is replaced by layer specific direct error broadcasts, that consists of a random weighted sum of the errors that are caused by the network outputs; typically in the last layer of the network (Samadi et al., 2017; Nøkland, 2016). This heuristic can in principle also be applied to the unrolled version of a recurrent neural network, yielding a different error broadcast for each layer of the unrolled network, or equivalently, for each time-slice of the computation in the recurrent network. This heuristic would suggest to send to each time slice error broadcasts that employ different random weight matrices. We found that the best results can be achieved if one chooses the same random projection of output errors for each time slice (see Figure 1d and e).

Definition of *merge #1*: *Merge #1* defines a learning signal that only considers the instantaneous error of network outputs and ignores the influence of the current activity on future errors. As justified in Methods, this means that the approximation of the learning signal $\widehat{\mathbf{L}}_j^t$ is defined by replacing the total error derivative $\frac{dE}{dz_j^t}$ with the partial derivative $\frac{\partial E}{\partial z_j^t}$. Crucially, this replacement makes it possible to compute the learning signal in real-time, whereas the total derivative needs information about future errors which should be back-propagated through time for an exact computation.

To exhibit an equation that summarizes the resulting weight update, we consider a network of LIF neurons and output neurons formalized by k leaky readout neurons y_k^t with decay constant ξ. If E is defined as the squared error between the readouts y_k^t and their targets $y_k^{*,t}$, and the weight updates are implemented with gradient descent and learning rate η, this yields (the proof and more general formula (40) are given in Methods):

$$\Delta\theta_{ji}^{\text{rec}} = \eta \sum_t \Big(\sum_k \theta_{kj}^{\text{out}} (y_k^{*,t} - y_k^t) \Big) \sum_{t' \le t} \xi^{t-t'} p_j^{t'} \hat{z}_i^{t'-1} \,, \tag{4}$$

where $p_j^{t'}$ is a function of the post-synaptic membrane voltage (the pseudo-derivative, see Methods) and $\hat{z}_i^{t'}$ is a trace of preceding pre-synaptic spikes with a decay constant α. This is a three-factor learning rule of a type that is commonly used to model experimental data (Gerstner et al., 2018). However a less common feature is that, instead of a single global error signal, learning signals are neuron-specific weighted sums of different error signals arising from different output neurons k. For a more complex neuron model such as LIF with adaptive thresholds, which are decisive for tasks involving working memory (Bellec et al., 2018), the eligibility traces are given in equation (24) and the learning rule in equation (40).

To model biology, a natural assumption is that synaptic connections to and from readout neurons are realized through different neurons. Therefore it is hard to conceive that the feedback weights are exactly symmetric to the readout weights, as required for gradient descent according to equation (4) that follows from the theory. Broadcast alignment (Lillicrap et al., 2016) suggests the replacement of θ_{kj}^{out} by a random feedback matrix. We make the same choice to define a learning rule (42) with mild assumptions: the learning signal is a neuron-specific random projection of error signals. Hence we replaced the weights θ_{kj}^{out} with random

feedback weights denoted B_{kj}^{random}. Other authors (Zenke and Ganguli, 2018; Kaiser et al., 2018) have derived learning rules similar to (42) for feedforward networks of spiking neurons but not for recurrent ones. We test the learning performance of *merge #1* for two generic computational tasks for recurrent neural networks: generation of a temporal pattern, and storing selected information in working memory.

Pattern generation task #1.1 Pattern generation is an important component of motor systems. We asked whether the simple learning setup of *merge #1* endows RSNNs with the capability to learn to generate patterns in a supervised manner.

Task: To this end, we considered a pattern generation task which is an extension of the task used in (Nicola and Clopath, 2017). In this task, the network should autonomously generate a three-dimensional target signal for 1 s. Each dimension of the target signal is given by the sum of four sinusoids with random phases and amplitudes. Similar to (Nicola and Clopath, 2017), the network received a clock input that indicates the current phase of the pattern (see Methods).

Implementation: The network consisted of 600 recurrently connected LIF neurons. All neurons in this RSNN projected to three linear readout neurons. All recurrent and output weights were plastic, see Figure 1a. A single learning trial, consisted of a 1 s simulation where the network produced a three-dimensional output pattern and gradients were computed using *merge #1*. Network weights were updated after each learning trial (see Methods for details).

Performance: Figure 1c shows the spiking activity of a randomly chosen subset of 20 of 600 neurons in the RSNN along with the output of the three readout neurons after application of *merge #1* for 20, 100 and 500 seconds, respectively. In this representative example, the network achieved a very good fit to the target signal (normalized mean squared error 0.04). Panel d shows the averaged mean squared errors (mse) for several variants of *merge #1* and a few other learning methods.

As an attempt to bridge a gap between phenomenological measurements of synaptic plasticity and functional learning models, we addressed the question whether a synaptic plasticity rule that was fitted to data in (Clopath et al., 2010) could reproduce the function of the second and third factors in equation (4). These two terms ($\hat{z}_i^{t-1}$ and p_j^t) couple the presynaptic and postsynaptic activity in a multiplicative manner. In the model of long term potentiation fitted to data by (Clopath et al., 2010), the presynaptic term is identical but the postsynaptic term includes an additional non-linear factor depending on a filtered version of the membrane voltage. We found that a replacement of the plasticity rule of *merge #1* by the Clopath rule had little impact on the result, as shown in Figure 1d under the name "Clopath rule" (see equation (43) in Methods for a precise definition of this learning rule). We also asked whether these pre- and postsynaptic factors could be simplified further. When replacing the trace $\hat{z}_i^{t-1}$ and p_j^t by the binary variable z_i^{t-1}, this just caused an increase of the mse from 0.023 to 0.032. In comparison, when the network has no recurrent connections or when the learning signal is replaced by a uniform global learning signal ($B_{kj} = \frac{1}{\sqrt{n}}$ with n the number of neurons) the mse increased by one order of magnitude to 0.236 and 0.484, respectively. This indicated the importance of diverse learning signals and recurrent connections in this task.

Panel e shows that a straightforward application of BA to the unrolled RSNNs, with new random matrices for error broadcast at every ms, or every 20 ms, worked less well. Finally, while *merge #1* managed to solve the task very well (Figure 1d), BPTT achieved an even lower mean squared error (black line in Figure 1e).

Store-recall task #1.2 Many learning tasks for brains involve some form of working memory. We therefore took a simple working memory task and asked whether *merge #1* enables an LSNN to learn this task, in spite of a substantial delay between the network decision to store information and the time when the network output makes an error.

Task: The network received a sequence of binary values encoded by alternating activity of two groups of input neurons ("Value 0" and "Value 1" in Figure 2a, top). In addition, it received command inputs, STORE and RECALL, encoded likewise by dedicated groups of input neurons. The task for the network was to output upon a RECALL command the value that was present in the input at the time of the most recent STORE command. In other words, the network had to store a bit (at a STORE command) and recall it at a RECALL command. After a STORE, a RECALL instructions was given during each subsequent time

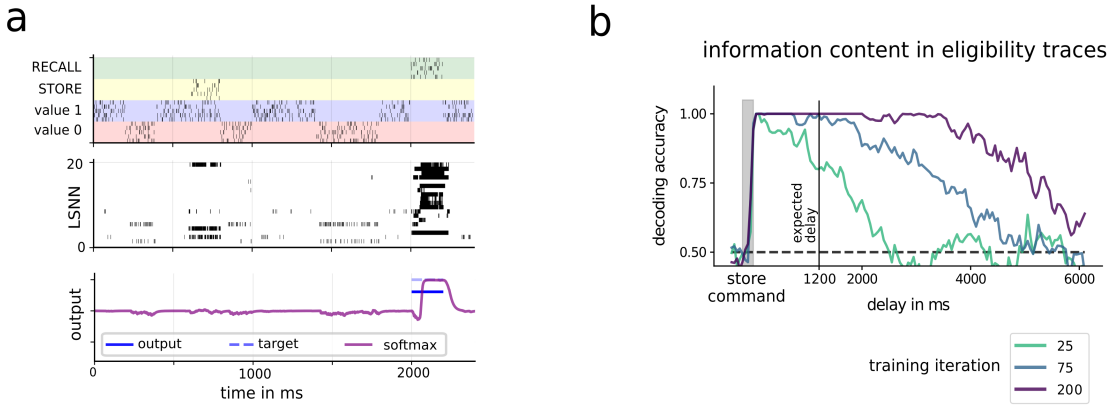

Figure 2: **Testing *merge #1* on a task where errors arise after a significant delay: store-recall task a**) The task requires to store the value 0 or 1 that is currently provided by other input neurons when a STORE command is given, and to recall it when a RECALL command is given. It is solved with *merge #1* for an LSNN. **b**) Information content of eligibility traces (estimated via a linear classifier) about the bit that was to be stored.

period of length $D = 200$ ms with probability $p_{\text{command}} = \frac{1}{6}$. This resulted in an expected delay of $\frac{D}{p_{\text{command}}} = 1.2$ s between STORE and RECALL instruction. The next STORE appeared in each subsequent period of length D with probability p_{command}. We considered the task as solved when the misclassification rate on the validation set reached a value below 5%.

Implementation: We used a recurrent LSNN network consisting of 10 standard LIF neurons and 10 LIF neurons with adaptive thresholds. The input neurons marked in blue and red at the top of Figure 2a produced Poisson spike trains with time varying rates. An input bit to the network was encoded by spiking activity at 50 Hz in the corresponding input channels for a time period of $D = 200$ ms. STORE and RECALL instructions were analogously encoded through firing of populations of other Poisson input neurons at 50 Hz. Otherwise input neurons were silent. The four groups of input channels consisted of 25 neurons each. During a recall cue, the binary network output is encoded by the sign of the time-averaged membrane potential of a linear readout neuron. An error signal is provided only during the duration of a RECALL command. It assumes value 1 at time points when the binary network output differs from the target output bit, otherwise is assumes value 0. The network was trained with *merge #1*. Parameters were updated every 2.4 s during training.

Performance: Figure 2a shows a network trained with *merge #1* that stores and recalls a bit accurately. This network reached a misclassification rate on separate validation runs below 5% in 195 iterations with *merge #1*. For comparison, the same accuracy was reached within 60 iterations with full BPTT. We found that adaptive neurons are essential for these learning procedures to succeed: A network of 20 non-adapting LIF neurons could not solve this task, even if it was trained with BPTT.

It might appear surprising that *merge #1* is able to train an LSNN for this task, since the learning signal is only non-zero during a RECALL command. This appears to be problematic, because in order to reduce errors the network has to learn to handle information from the input stream in a suitable manner during a much earlier time window: during a STORE command, that appeared on average 1200 ms earlier. We hypothesized that this was made possible because eligibility traces can hold information during this delay. In this way a learning signal could contribute to the modification of the weight of a synapse that had been activated much earlier, for example during a STORE command. According to the theory (equation (24) in the methods), eligibility traces of adapting neurons decay with a time constant comparable to that of the threshold adaptation. To verify experimentally that this mechanism makes it possible to hold the relevant information, we quantified the amount of information about the bit to be stored that is contained in the eligibility traces. This information was estimated via the decoding accuracy of linear classifiers, and the results are reported in Figure 2b. While

the neuron adaptation time constants were set to 1.2 s, we found that the decoding accuracy quickly rises above chance even for much longer delays. After 200 training iterations, the relevant bit can be decoded up to 4 s after the store signal arrived with high accuracy ($> 90\%$ of the trials).

Altogether the results in Figure 1 and 2 suggest that *merge #1* enables RSNNs to learn the most fundamental tasks which RSNNs have to carry out in the brain: to generate desired temporal patterns and to carry out computations that involve a working memory. Pattern generation tasks were also used for testing the performance of FORCE training for RSNNs in (Nicola and Clopath, 2017). While FORCE training has not been argued to be biologically plausible because of the use of a non-local plasticity rule and the restriction of plasticity to readout neurons, *merge #1* only engages mechanisms that have been argued to be biologically plausible. Hence it provides a concrete hypothesis how recurrent networks of neurons in the brain can learn to solve the most fundamental tasks which such networks are conjectured to carry out. More generally, we conjecture that *merge #1* can solve all learning tasks that have been demonstrated to be solvable by FORCE training. However we do not want to claim that *merge #1* can solve all learning tasks for RSNNs that can be solved by BPTT according to (Bellec et al., 2018). But the power of *merge* algorithms can be substantially enhanced by using more sophisticated learning signals than just random linear combinations of errors as in broadcast alignment. Several neural systems in the brain receive raw error signals from the periphery and output highly processed learning cues for individual brain areas and populations of neurons. We propose that such neural systems have been refined by evolution and development to produce learning signals that enable more powerful versions of *merge* algorithms, such as the ones that we will discuss in the following.

For implementations of *merge* algorithms in neuromorphic hardware – in order to enable efficient on-chip learning of practically relevant tasks – another reservoir of mechanisms becomes of interest that also can make use of concrete aspects of specific neuromorphic hardware. For example, in order to approximate the performance of BPTT for training LSNNs for speech recognition, a test on the popular benchmark dataset TIMIT suggests that the accuracy of *merge #1* (0.601) can be brought closer to that achieved by BPTT (0.647; see (Bellec et al., 2018)) by simply using the current values of readout weights, rather than random weights, for broadcasting current error signals. This yields an accuracy of 0.632.

Merge #2: Refined learning signals that emerge from L2L

The construction and proper distribution of learning signals appears to be highly sophisticated in the brain. Numerous areas in the human brain appear to be involved in the production of the error-related negativity and the emission of neuromodulatory signals (see the references given in the Introduction). A closer experimental analysis suggests diversity and target-specificity even for a single neuromodulator (Engelhard et al., 2018). Hence it is fair to assume that the construction and distribution of error signals in the brain has been optimized through evolutionary processes, through development, and prior learning. A simple approach for capturing possible consequences of such an optimization in a model is to apply L2L to a suitable family of learning tasks. The outer loop of L2L models opaque optimization processes that shape the distribution of error signals in the brain on the functional level. We implement this optimization by an application of BPTT to a separate error module in the outer loop of L2L. Since this outer loop is not meant to model an online learning process, we are not concerned here by the backpropagation through time that is required in the outer loop. In fact, one can expect that similar results can be achieved through an application of gradient-free optimization methods in the outer loop, but at a higher computational cost for the implementation.

It is argued in (Brea and Gerstner, 2016) that one-shot learning is one of two really important learning capabilities of the brain that are not yet satisfactorily explained by current models in computational neuroscience. We show here that *merge #2* explains how RSNNs can learn a new movement trajectory in a single trial. Simultaneously we show that given movement trajectories of an end-effector of an arm model can be learnt without requiring an explicitly learnt or constructed inverse model. Instead, a suitably trained error module can acquire the capability to produce learning signals for the RSNN so that the RSNN learns via *merge* to minimize deviations of the end-effector from the target trajectory in Euclidean

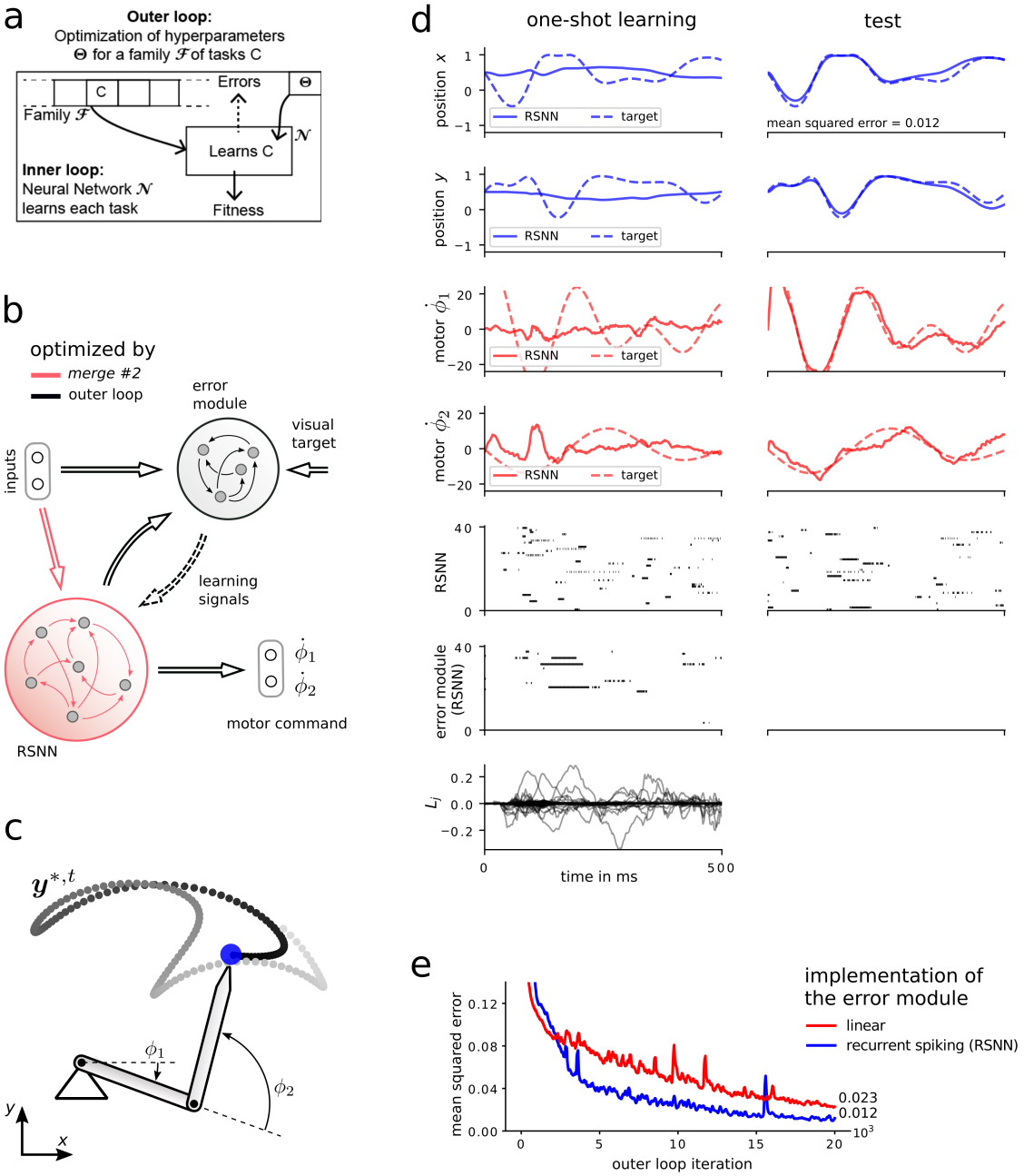

Figure 3: **Scheme and performance of *merge #2*** **a**) Learning-to-Learn (LTL) scheme. **b**) Learning architecture for *merge #2*. In this demo the angular velocities of the joints were controlled by a recurrent network of spiking neurons (RSNN). A separate error module was optimized in the outer loop of L2L to produce suitable learning signals. **c**) Randomly generated target movements $\boldsymbol{y}^{*,t}$ (example shown) had to be reproduced by the tip of an arm with two joints. **d**) Demonstration of one-shot learning for a randomly sampled target movement. During the training trial the error module sends learning signals (bottom row) to the network. After a single weight update the target movement can be reproduced in a test trial with high precision. **e**) One-shot learning performance improved during the course of outer loop optimization. Two error module implementations were compared.

space, rather than errors in "muscle space", i.e., in terms of the joint angles that are controlled by the RSNN.

Definition of *merge #2*: The main characteristic of *merge #2* is that the learning signals $\widehat{\boldsymbol{L}}_j^t$ are produced by a trained error module, which is modeled as a recurrent network of spiking neurons with synaptic weights $\boldsymbol{\Psi}$. It receives the input $\boldsymbol{x}^t$, the spiking activity in the network $\boldsymbol{z}^t$ and target signals $\boldsymbol{y}^{*,t}$. Note that the target signal is not necessarily the target output of the network, but can be more generally a target state vector of some controlled system. The output of the error module is denoted by ℓ_j^t, and is used in place of the error derivative $\frac{dE}{dz_j^t}$ in the definition of the learning signals according to equation (3). Thus, *merge #2* uses approximate learning signals of the form $\widehat{\boldsymbol{L}}_j^t = \ell_j^t \frac{\partial z_j^t}{\partial \boldsymbol{s}_j^t}$.

We employ the concept of Learning-to-Learn (L2L) to enable the network with its adjacent error module to solve a family $\mathcal{F}$ of one-shot learning tasks, i.e. each task C of the family $\mathcal{F}$ requires the network to learn a movement from a single demonstration. The L2L setup introduces a nested optimization procedure that consists of two loops: An inner loop and an outer loop as illustrated in Figure 3a. In the inner loop we consider a particular task C, entailing a training trial and a testing trial. During the training trial, the network has synaptic weights $\boldsymbol{\theta}_{\text{init}}$, it receives an input it has never seen and generates a tentative output. After a single weight update using *merge #2*, the network starts the testing trial with new weights $\boldsymbol{\theta}_{\text{test},C}$. It receives the same input for a second time and its performance is evaluated using the cost function $\mathcal{L}_C(\boldsymbol{\theta}_{\text{test},C})$. In the outer loop we optimize $\boldsymbol{\theta}_{\text{init}}$ and the error module parameters $\boldsymbol{\Psi}$ in order to minimize the cost $\mathcal{L}_C$ over many task instances C from the family $\mathcal{F}$. Formally, the optimization problem solved by the outer loop is written as:

$$\min_{\boldsymbol{\Psi},\boldsymbol{\theta}_{\text{init}}} \quad \mathbb{E}_{C\sim\mathcal{F}}\left[\mathcal{L}_C(\boldsymbol{\theta}_{\text{test},C})\right] \tag{5}$$

$$\text{s.t.:} \quad (\boldsymbol{\theta}_{\text{test},C})_{ji} = (\boldsymbol{\theta}_{\text{init}})_{ji} - \eta \textstyle\sum_t \widehat{\boldsymbol{L}}_j^t \cdot \boldsymbol{e}_{ji}^t \tag{6}$$

$$(\widehat{\boldsymbol{L}}_j^t \text{ and } \boldsymbol{e}_{ji}^t \text{ are obtained during the training trial for task } C \text{ using } \boldsymbol{\Psi} \text{ and } \theta_{\text{init}}),$$

where η represents a fixed learning rate.

One-shot learning task #2.1 It is likely that prior optimization processes on much slower time scales, such as evolution and development, have prepared many species of animals to learn new motor skills much faster than shown in Figure 1d. Humans and other species can learn a new movement by observing just one or a few examples. Therefore we restricted the learning process here to a single trial (one-shot learning, or imitation learning).

Another challenge for motor learning arises from the fact that motor commands $\dot{\Phi}^t$ have to be given in "muscle space" (joint angle movements), whereas the observed resulting movement $\boldsymbol{y}^t$ is given in Euclidean space. Hence an inverse model is usually assumed to be needed to infer joint angle movements that can reduce the observed error. We show here that an explicit inverse model is not needed, since its function can be integrated into the learning signals from the error module of *merge #2*.

Task: Each task C in the family $\mathcal{F}$ consisted of learning a randomly generated target movement $\boldsymbol{y}^{*,t}$ of the tip of a two joint arm as shown in Figure 3c. The task was divided into a training and a testing trial, with a single weight update in between according to equation (6).

Implementation: An RSNN, consisting of 400 recurrently connected LIF neurons, learnt to generate the required motor commands, represented as the angular velocities of the joints $\dot{\Phi}^t = (\dot{\phi}_1^t, \dot{\phi}_2^t)$, in order to produce the target movement. The full architecture of the learning system is displayed in Figure 3b. The error module consisted of 300 LIF neurons, which were also recurrently connected. The input $\boldsymbol{x}^t$ to the network was the same across all trials and was given by a clock-like signal. The input to the error module contained a copy of $\boldsymbol{x}^t$, the spiking activity $\boldsymbol{z}^t$ of the main network, as well as the target movement $\boldsymbol{y}^{*,t}$ in Euclidean space. Importantly, the error module had no access to actual errors of the produced motor commands. For outer loop optimization we viewed the learning process as a dynamical system for which we applied BPTT. Gradients were computed using batches of different tasks to approximate the expectation in the outer loop objective.

Performance: After sufficiently long training in the outer loop of L2L, we tested the learning capabilities of *merge #2* on a random target movement, and show in Figure 3d training and testing trial in the left and right column respectively. In fact, after the error module had sent learning signals to the network during the training trial, it was usually silent during testing, since it realized that the reproduced movement was correct. Therefore, the network was endowed with one-shot learning capabilities by *merge #2*, after initial weights $\boldsymbol{\theta}_{\text{init}}$ and the error module parameters $\boldsymbol{\Psi}$ had been optimized in the outer loop.

Figure 3e summarizes the mean squared error between the target $\boldsymbol{y}^{*,t}$ and actual movement $\boldsymbol{y}^t$ in the testing trial (blue curve). The red curve reports the same for a linear error module. The error is reported for different stages of the outer loop optimization.

We considered also the case when one uses instead of the eligibility trace as defined in equation (20) just a truncated one, given by $e_{ji}^t = z_j^{t-1}$, and found this variation to work just as well in this task (not shown).

Altogether we have shown that *merge #2* enables one-shot learning of pattern generation by an RSNN. This is apparently the first time that one-shot learning of pattern generation has been demonstrated for an RSNN. In addition, we have shown that the learning architecture for *merge #2* supports a novel solution to motor learning, where no separate construction or learning of an inverse model is required. More precisely, the error module can be trained to produce learning signals that enable motor learning without the presence of an inverse model. It will be interesting to see whether there are biological data that support this simplified architecture. Another interesting feature of the resulting paradigm for motor learning is that the considered tasks can be accomplished without sensory feedback about the actual trajectory of the arm movement. Instead the error module just receives efferent copies of the spiking activity of the main network, and hence implicitly also of its motor commands.

Merge #3: Producing learning signals through synthetic gradients

We show here that the use of biologically inspired eligibility traces also improves some state-of-the-art algorithms in machine learning, specifically the synthetic gradient approach for learning in feedforward and recurrent (artificial) neural networks (Jaderberg et al., 2016) and (Czarnecki et al., 2017). Synthetic gradients provide variants of backprop and BPTT that tend to run more efficiently because they avoid that synaptic weights can only be updated after a full network simulation followed by a full backpropagation of error gradients („locking"). We show here that the performance of synthetic gradient methods for recurrent neural networks significantly increases when they are combined with eligibility traces. The combination of these two approaches is in a sense quite natural, since synthetic gradients can be seen as online learning signals for *merge*. In comparison with *merge #2*, one does not need to consider here a whole family of learning tasks for L2L and a computationally intensive outer loop. Hence the production of learning signals via synthetic gradient approaches tends to be computationally more efficient. So far it also yields better results in applications to difficult tasks for recurrent artificial neural networks. In principle it is conceivable that biological learning systems also follow a strategy whereby learning signals for different temporal phases of a learning process are aligned among each other through some separate process. This is the idea of synthetic gradients.

Definition of *merge #3*: When BPTT is used for tasks that involve long time series, the algorithm is often truncated to shorter intervals of length Δt, and the parameters are updated after the processing of each interval. This variant of BPTT is called truncated BPTT. We show a schematic of the computation performed on the interval $[t - \Delta t, t]$ in the first panel of Figure 4b. Reducing the length Δt of the interval has two benefits: firstly, the parameter updates are more frequent; and secondly, it requires less memory storage because all inputs $\mathbf{x}^{t'}$ and network states $\mathbf{s}_j^{t'}$ for $t' \in [t - \Delta t, t]$ need to be stored temporarily for back-propagating gradients. The drawback is that the error gradients become more approximative, in particular, the relationship between error and network computation happening outside of this interval cannot be captured by the truncated error gradients. As illustrated in the last panel of Figure 4b, *merge #3* uses the same truncation scheme but alleviates the drawback of truncated BPTT in two ways: it uses eligibility traces to include information about the network history before $t - \Delta t$, and an error module that predicts errors after t. Thanks to eligibility traces, all input and recurrent activity patterns that happened before $t - \Delta t$ have left a footprint that can be combined with the learning signals $L_j^{t'}$ with $t' \in [t - \Delta t, t]$. Each

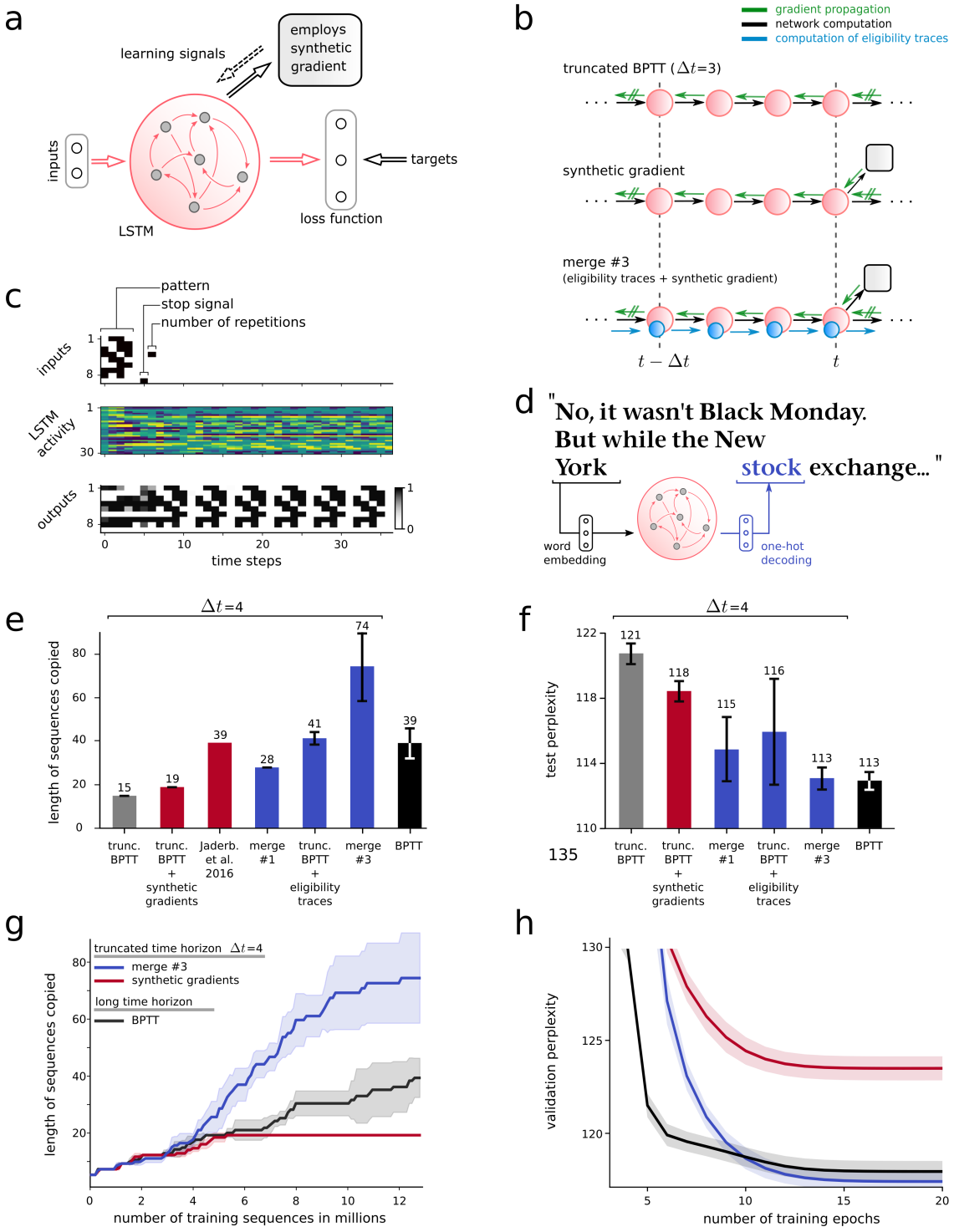

Figure 4: **Scheme and performance of *merge #3*** **a**) Learning architecture for *merge #3*. The error module is implemented through synthetic gradients. **b**) Scheme of learning rules used in panels **e-h**. Δt is the number of time steps through which the gradients are allowed to flow for truncated BPTT or for the computation of synthetic gradients. **c**,**e**,**g**) Copy-repeat task: (**c**) example trial, (**e**) performance of different algorithms for the copy-repeat task, (**g**) Learning progress for 3 different algorithms. This task was used as a benchmark in (Jaderberg et al., 2016) and (Graves et al., 2014). **d**,**f**,**h**) Word prediction task: (**d**) example of sequence, (**f**) performance summary for different learning rules, (**h**) learning progression. An epoch denotes a single pass through the complete dataset.

of these learning signals summarizes the neuron influence on the next errors, but due to the truncation, errors performed outside of the current interval cannot trivially be taken into account. In *merge #3*, the error module computes learning signals $L_j^{t'}$ that anticipate the predictable components of the future errors.

Merge #3 combines eligibility traces and learning signals to compute error gradients according to equation (1), rather than equation (12) that is canonically used to compute gradients for BPTT. To compute these gradients when processing the interval $[t-\Delta t, t]$, the eligibility traces and learning signals are computed solely from data available within that interval, without requiring the rest of the data. The eligibility traces are computed in the forward direction according to equation (2). For the learning signals $L_j^{t'}$, the difficulty is to estimate the gradients $\frac{dE}{dz_j^{t'}}$ for t' between $t-\Delta t$ and t. These gradients are computed by back-propagation from $t'=t$ back to $t'=t-\Delta$ with the two recursive formulas:

$$\frac{dE}{d\mathbf{s}_j^{t'}} = \frac{dE}{dz_j^{t'}}\frac{\partial z_j^{t'}}{\partial \mathbf{s}_j^{t'}} + \frac{dE}{d\mathbf{s}_j^{t'+1}}\frac{\partial \mathbf{s}_j^{t'+1}}{\partial \mathbf{s}_j^{t'}} \tag{7}$$

$$\frac{dE}{dz_j^{t'}} = \frac{\partial E}{\partial z_j^{t'}} + \sum_i \frac{dE}{d\mathbf{s}_i^{t'+1}}\frac{\partial \mathbf{s}_i^{t'+1}}{\partial z_j^{t'}} \ , \tag{8}$$

which are derived by application of the chain rule at the nodes $\mathbf{s}_j^t$ and z_j^t of the computational graph represented in Figures 5 ($\frac{\partial \mathbf{s}_i^{t+1}}{\partial z_j^t}$ is a notation short-cut for $\frac{\partial M}{\partial z_j^t}(\mathbf{s}_i^t, \mathbf{z}^t, \mathbf{x}^t, \boldsymbol{\theta})$, and i range over all neurons to which neuron j is synaptically connected). This leaves open the choice of the boundary condition $\frac{dE}{d\mathbf{s}_j^{t+1}}$ that initiates the back-propagation of these gradients at the end of the interval $[t-\Delta t, t]$. In most implementations of truncated BPTT, one chooses $\frac{dE}{d\mathbf{s}_j^{t+1}} = 0$, as if the simulation would terminate after time t. We use instead a feed-forward neural network SG parametrized by Ψ that outputs a boundary condition SG_j associated with each neuron j: $\frac{dE}{d\mathbf{s}_j^{t+1}} = \mathrm{SG}_j(\mathbf{z}^t, \Psi)$. This strategy was proposed by (Jaderberg et al., 2016) under the name "synthetic gradients". The synthetic gradients are combined with the error gradients computed within the interval $[t-\Delta t, t]$ to define the learning signals. Hence, we also see SG as a key component of the an error module in analogy with those of *merge #1* and #2.

To train SG, back-ups of the gradients $\frac{dE}{d\mathbf{s}_j^{t+1}}$ are estimated from the boundary conditions at the end of the next interval $[t, t+\Delta t]$. In a similar way as value functions are approximated in reinforcement learning, these more informed gradient estimates are used as targets to improve the synthetic gradients $\mathrm{SG}(\mathbf{z}^t, \Psi)$. This is done in *merge #3* by computing simultaneously the gradients $\frac{dE'}{d\boldsymbol{\theta}}$ and $\frac{dE'}{d\Psi}$ of an error function E' that combines the error function E, the boundary condition, and the mean squared error between the synthetic gradient and its targeted back-up (see Algorithm 1 in Methods for details). These gradients are approximated to update the network parameters with any variant of stochastic gradient descent.

It was already discussed that the factorization (1) used in *merge* is equivalent to BPTT, and that both compute the error gradients $\frac{dE}{d\theta_{ji}}$. In the subsection dedicated to *merge #3* of Methods, we formalize the algorithm when the simulation is truncated into intervals of length Δt. We then show under the assumption that the synthetic gradients are optimal, i.e. $\mathrm{SG}_j(\mathbf{z}^t, \Psi) = \frac{dE}{d\mathbf{s}_j^{t+1}}$, that both truncated BPTT and *merge #3* compute the correct error gradients $\frac{dE}{d\theta_{ji}}$. However, this assumption is rarely satisfied in simulations, firstly because the parameters Ψ may not converge instantaneously to some optimum; and even then, there could be unpredictable components of the future errors that cannot be estimated correctly. Hence, a more accurate model is to assume that the synthetic gradients $\mathrm{SG}_j(\mathbf{z}^t, \Psi)$ are noisy estimators of $\frac{dE}{d\mathbf{s}_j^{t+1}}$. Under this weaker assumption it turns out that *merge #3* produces estimators of the error gradients $\widehat{\frac{dE}{d\theta_{ji}}}^{\mathrm{merge}}$ that are better than those produced with truncated BPTT with

synthetic gradients $\widehat{\frac{dE}{d\theta_{ji}}}^{\text{SG}}$. Formally, this result can be summarized as:

$$\mathbb{E}\left[\left(\frac{dE}{d\theta_{ji}} - \widehat{\frac{dE}{d\theta_{ji}}}^{\text{merge}}\right)^2\right] \leq \mathbb{E}\left[\left(\frac{dE}{d\theta_{ji}} - \widehat{\frac{dE}{d\theta_{ji}}}^{\text{SG}}\right)^2\right] , \tag{9}$$

where the $\mathbb{E}$ is the stochastic expectation. The proof of this result will be published in a later version of the paper. To summarize the proof, we compare in detail the terms computed with *merge #3* and BPTT. The derivation reveals that both algorithms compute a common term that combines partial derivatives $\frac{\partial \mathbf{s}^t}{\partial \theta_{ji}}$ with errors, $E(\mathbf{z}^{t'})$ with t and t' being accessible within the interval $[t - \Delta t, t]$. The difference between the two algorithms is that truncated BPTT combines all the partial derivatives $\frac{\partial \mathbf{s}^t}{\partial \theta_{ji}}$ with synthetic gradients that predict future errors. Instead, the *merge* algorithm holds these partial derivatives in eligibility traces to combine them later with a better informed learning signals that do not suffer from the noisy approximations of the synthetic gradients.

Copy-repeat task #3.1 The copy-repeat task was introduced in (Graves et al., 2014) to measure how well artificial neural networks can learn to memorize and process complex patterns. It was also used in (Jaderberg et al., 2016) to compare learning algorithms for recurrent neural networks with truncated error propagation.

Task: The task is illustrated in Figure 4c. It requires to read a sequence of 8-bit characters followed by a "stop" character and a character that encodes the requested number of repetitions. In the subsequent time steps the network is trained to repeat the given pattern as many times as requested, followed by a final "stop" character. We used the same curriculum of increased task complexity as defined in (Jaderberg et al., 2016), where the authors benchmarked variants of truncated BPTT and synthetic gradients: the pattern length and the number of repetitions increase alternatively each time the network solves the task (the task is considered solved when the average error is below 0.15 bits per sequence). The performance of the learning algorithms are therefore measured by the length of the largest sequence for which the network could solve the task.

Implementation: The recurrent network consisted of 256 LSTM units. The component SG of the error module was a feedfoward network with one hidden layer of 512 rectified linear units (the output layer of the synthetic gradient did not have non-linear activation functions). The network states were reset to zero at the beginning of each sequence and the mean error and the error gradients were averaged over a batch of 256 independent sequences. Importantly, we used for all training algorithms a fixed truncation length $\Delta t = 4$, with the exception of a strong baseline BPTT for which the gradients were not truncated.

Results: The activity and output of a trained LSTM solving the task is displayed in Figure 4c. The performance of various learning algorithms is summarized in Figure 4e and g. Truncated BPTT alone solves the task for sequences of 15 characters, and 19 characters when enhanced with synthetic gradients. With a different implementation of the task and algorithm, (Jaderberg et al., 2016) reported that sequences of 39 characters could be handled with synthetic gradients. When BPTT is replaced by *merge* and uses eligibility traces, the network learnt to solve the task for sequences of length 41 when the synthetic gradients were set to zero (this algorithm is referred as "truncated BPTT + eligibility traces" in Figure 4e and f). The full *merge #3* algorithm that includes eligibility traces and synthetic gradients solved the task for sequences of 74 characters. This is an improvement over *merge #1* that handles sequences of 28 characters, even if the readout weights are not replaced by random error broadcasts. All these results were achieved with a truncation length $\Delta t = 4$. In contrast when applying full back-propagation through the whole sequence, we reached only 39 characters.

Word prediction task #3.2 We also considered a word-level prediction task in a corpus of articles extracted from the Wall Street Journal, provided by the so-called Penn Treebank dataset. As opposed to the previous copy-repeat task, this is a practically relevant task. It has been standardized to provide a reproducible benchmark task. Here, we used the same implementation and baseline performance as provided freely by the Tensorflow tutorial on recurrent neural networks*.

*https://www.tensorflow.org/tutorials/sequences/recurrent

Task: As indicated in Figure 4d, the network reads the whole corpus word after word. At each time step, the network has to read a word and predict the following one. The training, validation and test sets consist of texts of 929k, 73k, and 82k words. The sentences are kept in a logical order such that the context of dozens of words matters to accurately predict the following ones. The vocabulary of the dataset is restricted to the 10k most frequent words, and the words outside of this vocabulary are replaced with a special unknown word token.

Implementation: For all algorithms, the parameters were kept identical to those defined in the Tensorflow tutorial, with two exceptions: firstly, the networks had a single layer of 200 LSTM units instead of two to simplify the implementation, and because the second did not seem to improve performance with this configuration; secondly, the truncation length was reduced from $\Delta t = 20$ to $\Delta t = 4$ for synthetic gradients and *merge #3* to measure how our algorithms compensate for it. The synthetic gradients are computed by a one hidden layer of 400 rectified linear units. For a detailed description of model and a list of parameters we refer to the methods.

Results: The error is measured for this task by the word-level perplexity, which is the exponential of the mean cross-entropy loss. Figure 4e and f summarize our results. After reduction to a single layer, the baseline perplexity of the model provided in the Tensorflow tutorial for BPTT was 113 with a truncation length $\Delta t = 20$. Full BPTT is not practical in this case because the data consists of one extremely long sequence of words. In contrast, in a perplexity increased to 121 when the same model was also trained with BPTT, but a shorter truncation length $\Delta t = 4$. The model performance improved back to 118 with synthetic gradients, and to 116 with eligibility traces. Applying the *merge #1* algorithm with the true readout weights as error broadcasts resulted in a perplexity of 115. When combining both eligibility traces and synthetic gradients in *merge #3*, the performance improved further and we achieved a perplexity of 113.

To further investigate the relevance of eligibility traces for *merge #3* we considered the case where the eligibility trace were truncated. Instead of using the eligibility trace as defined in equation (33) we used $\boldsymbol{e}_{ji}^t = \frac{\partial \mathbf{s}_j^t}{\partial \theta_{ji}}$. This variation of *merge #3* lead to significantly degraded performance and resulted in test perplexity of 122.67 (not shown).

All together Figure 4e and f show that eligibility traces improve truncated BPTT more than synthetic gradients. Furthermore, if eligibility traces are combined with synthetic gradients in *merge #3*, one arrives at an algorithm that outperforms full BPTT for the copy-repeat task, and matches the performance of BPTT ($\Delta t = 20$) for the Penn Treebank word prediction task.

Discussion

The functionally most powerful learning method for recurrent neural nets, an approximation of gradient descent for a loss function via BPTT, requires propagation of error signals backwards in time. Hence this method does not reveal how recurrent networks of neurons in the brain learn. In addition, propagation of error signals backwards in time requires costly work-arounds in software implementations, and it does not provide an attractive blueprint for the design of learning algorithms in neuromorphic hardware. We have shown that a replacement of the propagation of error signals backwards in time in favor of a propagation of the local activation histories of synapses – called eligibility traces – forward in time allows us to capture with physically and biologically realistic mechanisms a large portion of the functional benefits of BPTT. We are referring to to this new approach to gradient descent learning in recurrent neural networks as *merge*. In contrast to many other approaches for learning in recurrent networks of spiking neurons it can be based on a rigorous mathematical theory.

We have presented a few variations of *merge* where eligibility traces are combined with different types of top-down learning signals that are generated and transmitted in real-time. In *merge #1* we combine eligibility traces with a variation of broadcast alignment (Samadi et al., 2017) or direct feedback alignment (Nøkland, 2016). We first evaluated the performance of *merge #1* on a task that has become a standard for the evaluation of the FORCE learning method for recurrent networks of spiking neurons (Nicola and Clopath, 2017) and related earlier work on artificial neural networks: supervised learning of generating a temporal pattern. In order to make the task more interesting we considered a task where 3 independent temporal patterns have to be generated simultaneously by the same RSNN. Here it is not

enough to transmit a single error variable to the network, so that broadcasting of errors for different dimensions of the network output to the network becomes less trivial. We found (see Figure 1) that a random weight matrix for the distribution of error signals works well, as in the case of feedforward networks (Samadi et al., 2017), (Nøkland, 2016). But surprisingly, the results were best when this matrix was fixed, or rarely changed, whereas a direct application of broadcast alignment to an unrolled recurrent network suggests that a different random matrix should be used for every time slice.

In order to challenge the capability of *merge #1* to deal also with cases where error signals arise only at the very end of a computation in a recurrent network, we considered a store-recall task, where the network has to learn what information it should store –and maintain until it is needed later. We found (see Figure 2) that this task can also be learnt with *merge #1*, and verified that the eligibility traces of the network were able to bridge the delay. We used here an LSNN (Bellec et al., 2018) that includes a model of a slower process in biological neurons: neuronal adaptation. We also compared the performance of *merge #1* with BPTT for a more demanding task: the speech recognition benchmark task TIMIT. We found that *merge #1* approximates also here the performance of BPTT quite well.

Altogether we have the impression that *merge #1* can solve all learning tasks for RSNNs that the FORCE method can solve, and many more demanding tasks. Since the FORCE method is not argued to be biologically realistic, whereas *merge #1* only relies on biologically realistic mechanisms, this throws new light on the understanding of learning in recurrent networks of neurons in the brain. An additional new twist is that *merge #1* engages also plasticity of synaptic connections within a recurrent network, rather than only synaptic connections to a postulated readout neuron as in the FORCE method. Hence we can now analyze how network configurations and motifs that emerge in recurrent networks through learning in a model relate to experimentally found data.

In the analysis of *merge #2* we moved to a minimal model that captures salient aspects of the organization of learning in the brain, where dedicated brain areas process error signals and generate suitably modified gating signals for plasticity in different populations of neurons. We considered in this minimal model just a single RSNN (termed error module) for generating learning signals. But obviously this minimal model opens the door to the analysis of more complex learning architectures –as they are found in the brain– from a functional perspective. We found that a straightforward application of the Learning-to-Learn (L2L) paradigm, where the error module is optimized on a slower time scale for its task, significantly boosts the learning capability of an RSNN. Concretely, we found that it endows the RSNN with one-shot learning capability (see Figure 3), hence with a characteristic advantage of learning in the human brain (Brea and Gerstner, 2016), (Lake et al., 2017). In addition the model of Figure 3 suggests a new way of thinking about motor learning. It is shown that no separate inverse model is needed to learn motor control. Furthermore in the case that we considered, not even sensory feedback from the environment is needed.

Finally we arrived at an example where biologically inspired ideas and mechanisms, in this case eligibility traces, can enhance state-of-the-art methods in machine learning. Concretely, we have shown in Figure 4 that adding eligibility traces to the synthetic gradient methods of (Jaderberg et al., 2016) and (Czarnecki et al., 2017) for training artificial recurrent neural networks significantly enhances the performance of synthetic gradient algorithms. In fact, the resulting algorithm *merge #3* was found to supercede the performance of full BPTT in one case and rival BPTT with $\Delta t = 20$ in another.

A remarkable feature of *merge* is that the resulting local learning rules (21) are very similar to previously proposed rules for synaptic plasticity that were fitted to experimental data (Clopath et al., 2010). In fact, we have shown in Figure 1d that the theory-derived local plasticity rule for *merge #1* can be replaced by the Clopath rule with little loss in performance. On a more general level, the importance of eligibility traces for network learning that our results suggest provides concrete hypotheses for the functional role of a multitude of processes on the molecular level in neurons and synapses, including metabotropic receptors. Many of these processes are known to store information about multiple aspects of the recent history. The *merge* approach suggests that these processes, in combination with a sufficiently sophisticated production of learning signals by dedicated brain structures, can practically replace the physically impossible backpropagation of error signals backwards in time of theoretically optimal BPTT.

A key challenge for neuromorphic engineering is to design a new generation of computing hardware that enables energy efficient implementations of major types of networks and learning algorithms that have driven recent progress in machine learning and learning-driven AI (see e.g. (Barrett et al., 2018)). Recurrent neural networks are an essential component of many of these networks, and hence learning algorithms are needed for this type of networks that can be efficiently implemented in neuromorphic hardware. In addition, neuromorphic implementations of recurrent neural networks – rather than deep feedforward networks—promise larger efficiency gains because hardware neurons can be re-used during a computation. Recently developed diffusive memristors (Wang et al., 2018) would facilitate an efficient local computation of eligibility traces with new materials. In addition, new 3-terminal memristive synapses (Yang et al., 2017) are likely to support an efficient combination of local eligibility traces with top-down error signals in the hardware. Thus *merge* provides attractive functional goals for novel materials in neuromorphic hardware.

Methods

General network model: Our proposed learning algorithms for recurrent neural networks can be applied to a large class of spiking and non-spiking neural network models. We assume that the state at time t of each neuron j in the network can be described by an internal state vector $\mathbf{s}_j^t \in \mathbb{R}^d$ and an observable state z_j^t. The internal state includes internal variables of the neuron such as its activation or membrane potential. The observable state is given by the output of the neuron (analog output for ANNs and spiking output for SNNs). The dynamics of a neuron's discrete-time state evolution is described by two functions $M(\mathbf{s}, \mathbf{z}, \mathbf{x}, \boldsymbol{\theta})$ and $f(\mathbf{s})$, where $\mathbf{s}$ is an internal state vector, $\mathbf{z}$ is a vector of the observable network state (i.e., outputs of all neurons in the network), $\mathbf{x}$ is the vector of inputs to the network, and $\boldsymbol{\theta}$ denotes the vector of network parameters (i.e., synaptic weights). In particular, for each neuron j the function M maps from the current network state observable to that neuron to its next internal state, and f maps from its internal state to its observable state (neuron output):

$$\mathbf{s}_j^t = M(\mathbf{s}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{\theta}), \tag{10}$$

$$z_j^t = f(\mathbf{s}_j^t), \tag{11}$$

where $\mathbf{z}^t$ ($\mathbf{x}^t$) denotes the vector of observable states of all network (input) neurons at time t. A third function E defines the error of the network within some time interval $0, \ldots, T$. It is assumed to depend only on the observable states $E(\mathbf{z}^0, \ldots, \mathbf{z}^T)$.

We explicitly distinguish betweens partial derivative and total derivatives in our notation. We write $\frac{\partial M}{\partial s}(\mathbf{s}^*, \mathbf{z}^*, \mathbf{x}^*, \boldsymbol{\theta})$ to denote the partial derivative of the function M with respect to $\mathbf{s}$, applied to particular arguments $\mathbf{s}^*, \mathbf{z}^*, \mathbf{x}^*, \boldsymbol{\theta}$. To simplify notation, we define the shortcuts $\frac{\partial \mathbf{s}_j^t}{\partial \mathbf{s}_j^{t-1}} \stackrel{\text{def}}{=} \frac{\partial M}{\partial s}(\mathbf{s}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{\theta})$, $\frac{\partial \mathbf{s}_j^t}{\partial \theta_{ji}} \stackrel{\text{def}}{=} \frac{\partial M}{\partial \theta_{ji}}(\mathbf{s}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{\theta})$, and $\frac{\partial z_j^t}{\partial \mathbf{s}_j^t} \stackrel{\text{def}}{=} \frac{\partial f}{\partial s}(\mathbf{s}_j^{t-1})$.

To emphasize that $\frac{\partial \mathbf{s}_j^t}{\partial \mathbf{s}_j^{t-1}}$ is a matrix of shape $d \times d$, and because it has an important role in the following derivation and in definition of eligibility traces, we also use the further notation $D_j^t = \frac{\partial \mathbf{s}_j^{t+1}}{\partial \mathbf{s}_j^t}$. Note that we write gradients as row vectors and states as column vectors.

Proof of factorization (equation (1)): We provide here the proof for equation (1), i.e., we show that the total derivative of the error function E with respect to the parameters $\boldsymbol{\theta}$ can be written as a product of learning signals $\mathbf{L}_j^t$ and eligibility traces $\mathbf{e}_{ji}^t$. First, recall that in BPTT the error gradient is decomposed as (see equation (12) in Werbos (1990)):

$$\frac{dE}{d\theta_{ji}} = \sum_t \frac{dE}{d\mathbf{s}_j^t} \cdot \frac{\partial \mathbf{s}_j^t}{\partial \theta_{ji}} \ , \tag{12}$$

where $\frac{dE}{d\mathbf{s}_j^t}$ is the total derivative of the error E with respect to the neuron states $\mathbf{s}_j^t$ at time step t. $\frac{dE}{d\mathbf{s}_j^t}$ can be expressed recursively as a function of the same derivative at the next time step $\frac{dE}{d\mathbf{s}_j^{t+1}}$ by applying the chain rule at the node $\mathbf{s}_j^t$ of the computational graph shown in

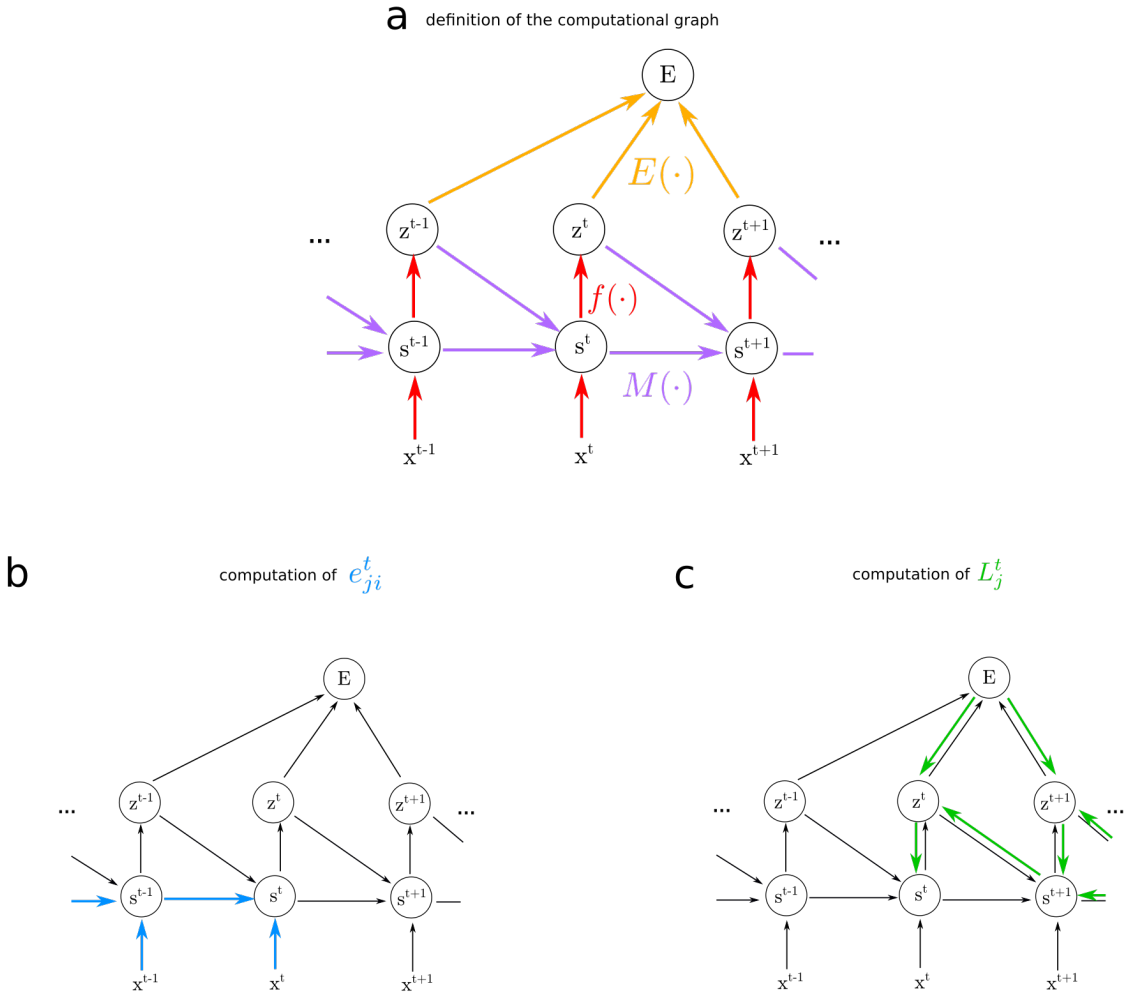

Figure 5: **Computational graph a)** Assumed mathematical dependencies between neuron states $\mathbf{s}_j^t$, neuron outputs $\mathbf{z}^t$, network inputs $\mathbf{x}^t$, and the network error E through the mathematical functions $f(\cdot)$, $M(\cdot)$ and $E(\cdot)$ represented by coloured arrows. **b)** The dependencies involved in the computation of the eligibility traces e_{ji}^t are shown in blue. **c)** The dependencies involved in the computation of the learning signal L_j^t are shown in green.

Figure 5c:

$$\frac{dE}{d\mathbf{s}_j^t} = \frac{dE}{dz_j^t}\frac{\partial z_j^t}{\partial \mathbf{s}_j^t} + \frac{dE}{d\mathbf{s}_j^{t+1}}\frac{\partial \mathbf{s}_j^{t+1}}{\partial \mathbf{s}_j^t} \tag{13}$$

$$= \mathbf{L}_j^t + \frac{dE}{d\mathbf{s}_j^{t+1}} D_j^t, \tag{14}$$

where we defined the *learning signal* for neuron j at time t as $\mathbf{L}_j^t \stackrel{\text{def}}{=} \frac{dE}{dz_j^t}\frac{\partial z_j^t}{\partial \mathbf{s}_j^t}$. The resulting recursive expansion ends at the last time step T, i.e., $\frac{dE}{d\mathbf{s}_j^{T+1}} = 0$. If one substitutes the recursive formula (14) into the definition of the error gradients (12), one gets:

$$\frac{dE}{d\theta_{ji}} = \sum_t \left(\mathbf{L}_j^t + \frac{dE}{d\mathbf{s}_j^{t+1}} D_j^t \right) \cdot \frac{\partial \mathbf{s}_j^t}{\partial \theta_{ji}} = \sum_t \left(\mathbf{L}_j^t + \left(\mathbf{L}_j^{t+1} + (\cdots) D_j^{t+1} \right) D_j^t \right) \cdot \frac{\partial \mathbf{s}_j^t}{\partial \theta_{ji}} \ . \tag{15}$$

The following equation is the main equation for understanding the transformation from BPTT into *merge*. The key idea is to collect all terms which are multiplied with the learning signal $\mathbf{L}_j^{t'}$ at a given time t'. These are only terms that concern events in the computation of neuron j at time t', and these do not depend on future errors or variables. Hence one can collect them conceptually into an internal eligibility trace $\mathbf{e}_{ji}^t$ for neuron j which can be computed autonomously within neuron j in an online manner.

To this end, we write the term in parentheses in equation (15) into a second sum indexed by t' and exchange the summation indices to pull out the learning signal $\mathbf{L}_j^t$. This expresses the error gradient as a sum of learning signals $\mathbf{L}_j^t$ multiplied by some factor indexed by ji, which implicitly defines what we call eligibility traces:

$$\frac{dE}{d\theta_{ji}} = \sum_t \sum_{t' \geq t} \mathbf{L}_j^{t'} D_j^{t'-1} \cdots D_j^t \cdot \frac{\partial \mathbf{s}_j^t}{\partial \theta_{ji}} \tag{16}$$

$$= \sum_{t'} \mathbf{L}_j^{t'} \underbrace{\sum_{t \leq t'} D_j^{t'-1} \cdots D_j^t \cdot \frac{\partial \mathbf{s}_j^t}{\partial \theta_{ji}}}_{\stackrel{\text{def}}{=} \mathbf{e}_{ji}^{t'}} \ . \tag{17}$$

Here, we use the identity matrix for the $D_j^{t-1} \cdots D_j^t$ where $t' - 1 < t$. Finally, it is easy to see that the eligibility traces can also be computed recursively as in equation (2). This proves that the error gradient satisfies the equation (1), given the definition of eligibility traces and learning signals in (2) and (3).

Leaky integrate-and-fire neuron model: We define here the leaky integrate-and-fire (LIF) spiking neuron model, and exhibit the update rules that result from *merge* for this model. We consider LIF neurons simulated in discrete time. In this case the internal state $\mathbf{s}_j^t$ is one dimensional and contains only the membrane voltage v_j^t. The observable state $z_j^t \in \{0, 1\}$ is binary, indicating a spike ($z_j^t = 1$) or no spike ($z_j^t = 0$) at time t. The dynamics of the LIF model is defined by the equations:

$$v_j^{t+1} = \alpha v_j^t + \sum_{i \neq j} \theta_{ji}^{\text{rec}} z_i^t + \sum_i \theta_{ji}^{\text{in}} x_i^t - z_j^t v_{\text{th}} \tag{18}$$

$$z_j^t = H\left(\frac{v_j^t - v_{\text{th}}}{v_{\text{th}}} \right), \tag{19}$$

where $x_i^t = 1$ indicates a spike from the input neuron i at time step t ($x_i^t = 0$ otherwise), θ_{ji}^{rec} (θ_{ji}^{in}) is the synaptic weight from network (input) neuron i to neuron j, and H denotes the Heaviside step function. The decay factor α is given by $e^{-\delta t/\tau_m}$, where δt is the discrete time step (1 ms in our simulations) and τ_m is the membrane time constant. Due to the term $-z_j^t v_{\text{th}}$ in equation (18), the neurons membrane voltage is reset to a lower value after an output spike.

Considering the LIF model defined above, we derive the resulting eligibility traces and error gradients. By definition of the model in equation (18), we have $D_j^t = \frac{\partial v_j^{t+1}}{\partial v_j^t} = \alpha$ and

$\frac{\partial v_j^t}{\partial \theta_{ji}} = z_i^{t-1}$. By definition of eligibility traces in equation (2), the eligibility trace is therefore a simple geometric series and one can write:

$$e_{ji}^{t+1} = \sum_{t' \leq t} \alpha^{t-t'} z_i^{t'} \tag{20}$$

In other words, the eligibility trace corresponds to the exponentially filtered pre-synaptic spike train. Note that this trace is specific to the pre-synaptic neuron and does not depend on the post-synaptic neuron j. To simplify notation, we therefore denote this trace as $\hat{z}_i^{t-1} \stackrel{\text{def}}{=} e_{ji}^t$. To exhibit the full learning rule, we need to compute the learning signal defined by equation (3). This requires to compute the derivative $\frac{\partial z_j^t}{\partial v_j^t}$, which is ill-defined in the case of LIF neurons because it requires the derivative of the discontinuous function H. As shown in (Bellec et al., 2018), we can alleviate this by using a pseudo-derivative p_j^t in place of $\frac{\partial z_j^t}{\partial v_j^t}$, given by $p_j^t = \gamma \max(0, 1 - \frac{v_j^t - v_{\text{th}}}{v_{\text{th}}})$ where $\gamma = 0.3$ is a constant called dampening factor. The gradient of the error with respect to a recurrent weight θ_{ji}^{rec} thus takes on the following form, reminiscent of spike-timing dependent plasticity:

$$\frac{dE}{d\theta_{ji}^{\text{rec}}} = \sum_t \frac{dE}{dz_j^t} p_j^t \hat{z}_i^{t-1} \ . \tag{21}$$

A similar derivation leads to the gradient of the input weights. In fact, one just needs to substitute recurrent spikes z_i^t by input spikes x_i^t.

Leaky integrate-and-fire with threshold adaptation: We derive the learning rule defined by *merge* for a LIF neuron model with an adaptive threshold. For this model, the internal state is given by a two-dimensional vector $\mathbf{s}_j^t := (v_j^t, b_j^t)^T$, where v_j^t denotes the membrane voltage as in the LIF model, and b_j^t is a threshold adaptation variable. As for the LIF model above, the voltage dynamics is defined by equation (18). The spiking threshold B_j^t at time t is given by

$$B_j^t = v_{\text{th}} + \beta b_j^t \ , \tag{22}$$

where v_{th} denotes the baseline-threshold. Output spikes are generated when the membrane voltage crosses the adaptive threshold $z_j^t = H\left(\frac{v_j^t - B_j^t}{v_{\text{th}}}\right)$, and the threshold adaptation evolves according to

$$b_j^{t+1} = \rho b_j^t + H\left(\frac{v_j^t - B_j^t}{v_{\text{th}}}\right). \tag{23}$$

The decay factor ρ is given by $e^{-\delta t/\tau_a}$, where δt is the discrete time step (1 ms in our simulations) and τ_a is the adaptation time constant. In other words, the neuron's threshold is increased with every output spike and decreases exponentially back to the baseline threshold.

Because of the extended state, we obtain two-dimensional eligibility traces per parameter: $\mathbf{e}_{ji}^t := (e_{ji,v}^t, e_{ji,b}^t)^T$ and the matrix D_j^t is a 2×2 matrix. On its diagonal one finds the terms $\frac{\partial v_j^{t+1}}{\partial v_j^t} = \alpha$ and $\frac{\partial b_j^{t+1}}{\partial b_j^t} = \rho - p_j^t \beta$.

Above and below the diagonal, one finds respectively $\frac{\partial v_j^{t+1}}{\partial b_j^t} = 0, \frac{\partial b_j^{t+1}}{\partial v_j^t} = p_j^t$. One can finally compute the eligibility traces using its definition in equation (2). The eligibility trace for the voltage remains the same as in the LIF case: $e_{ji,v}^t = \hat{z}_i^{t-1}$. For the eligibility trace associated with the adaptive threshold we find the following recursive update:

$$e_{ji,b}^{t+1} = p_j^t \hat{z}_i^{t-1} + (\rho - p_j^t \beta) e_{ji,b}^t \ . \tag{24}$$

This results in the following equation for the gradient of recurrent weights:

$$\frac{dE}{d\theta_{ji}^{\text{rec}}} = \sum_t \frac{dE}{dz_j^t} p_j^t \left(\hat{z}_i^{t-1} - \beta e_{ji,b}^t\right) \ . \tag{25}$$

The eligibility trace for input weights is again obtained by replacing recurrent spikes z_i^t with input spikes x_i^t.

Artificial neuron models: The dynamics of recurrent artificial neural networks is usually given by $s_j^t = \alpha s_j^{t-1} + \sum_i \theta_{ji}^{\text{rec}} z_i^{t-1} + \sum_i \theta_{ji}^{\text{in}} x_j^t$ with $z_j^t = \sigma(s_j^t)$, where $\sigma : \mathbb{R} \to \mathbb{R}$ is some activation function (often a sigmoidal function in RNNs). We call the first term the leak term in analogy with LIF models. For $\alpha = 0$ this term disappears, leading to the arguably most basic RNN model. If $\alpha = 1$, it models a recurrent version of residual networks.

For such model, one finds that $D_j^t = \alpha$ and the eligibility traces are of the form $e_{ji}^t = \hat{z}_i^t = \sum_{t' \leq t} z_i^{t'} \alpha^{t-t'}$. The resulting *merge* update is written as follows (with σ' the derivative of the activation function):

$$\frac{dE}{d\theta_{ji}^{\text{rec}}} = \sum_t \frac{dE}{dz_j^t} \sigma'(s_j^t) \hat{z}_i^{t-1}. \tag{26}$$

Although simple, this derivation provides insight in the relation between BPTT and *merge*. If the neuron model does not have neuron specific dynamics $\alpha = 0$, the factorization of *merge* is obsolete in the sense that the eligibility trace does not propagate any information from a time step to the next $D_j^t = 0$. Thus, one sees that *merge* is most beneficial for models with rich internal neural dynamics.

LSTM: For LSTM units, (Hochreiter and Schmidhuber, 1997) the internal state of the unit is the content of the memory cell and is denoted by c_j^t, the observable state is denoted by h_j^t. One defines the network dynamics that involves the usual input, forget and output gates (denoted by $\imath_j^t$, $\mathscr{f}_j^t$, and $\mathscr{o}_j^t$) and the cell state candidate $\widetilde{c}_j^t$ as follows (we ignore biases for simplicity):

$$\imath_j^t = \sigma\big(\sum_i \theta_{ji}^{\text{rec},\imath} h_i^{t-1} + \sum_i \theta_{ji}^{\text{in},\imath} x_i^t\big) \tag{27}$$

$$\mathscr{f}_j^t = \sigma\big(\sum_i \theta_{ji}^{\text{rec},\mathscr{f}} h_i^{t-1} + \sum_i \theta_{ji}^{\text{in},\mathscr{f}} x_i^t\big) \tag{28}$$

$$\mathscr{o}_j^t = \sigma\big(\sum_i \theta_{ji}^{\text{rec},\mathscr{o}} h_i^{t-1} + \sum_i \theta_{ji}^{\text{in},\mathscr{o}} x_i^t\big) \tag{29}$$

$$\widetilde{c}_j^t = \tanh\big(\sum_i \theta_{ji}^{\text{rec},c} h_i^{t-1} + \sum_i \theta_{ji}^{\text{in},c} x_i^t\big). \tag{30}$$

Using those intermediate variables as notation short-cuts, one can now write the update of the states of LSTM units in a form that we can relate to *merge*:

$$c_j^t = M(c_j^{t-1}, \mathbf{h}^{t-1}, \boldsymbol{\theta}) = \mathscr{f}_j^t c_j^{t-1} + \imath_j^t \widetilde{c}_j^t \tag{31}$$

$$h_j^t = f(c_j^t, \mathbf{h}^{t-1}, \boldsymbol{\theta}) = \mathscr{o}_j^t c_j^t. \tag{32}$$

There is one difference between LSTMs and the previous neuron models used for *merge*: the function f depends now on the previous observable state $\mathbf{h}^t$ and the parameters through the output gate $\mathscr{o}_j^t$. In fact the derivation of the gradients $\frac{dE}{d\theta_{ji}}$ in the paragraph "Proof of factorization" is still valid for deriving the gradients with respect to the parameters of the input gate, forget gate and cell state candidate, because these parameters do not take part in the definition of the function f. For these parameters we apply the general theory as follows. We compute $D_j^t = \frac{\partial c_j^{t+1}}{\partial c_j^t} = \mathscr{f}_j^t$ and for each variable $\theta_{ji}^{A,B}$ with A being either "in" or "rec" and B being $\imath, \mathscr{f}$, or c, we compute a set of eligibility traces. If we take the example the recurrent weights for the input gate $\theta_{ji}^{\text{rec},\imath}$, the eligibility traces are updated according to:

$$e_{ji}^{\text{rec},\imath,t} = \mathscr{f}_j^{t-1} e_{ji}^{\text{rec},\imath,t-1} + \widetilde{c}_j^t \imath_j^t (1 - \imath_j^t) h_i^t \ , \tag{33}$$

and the gradients are of the form

$$\frac{dE}{d\theta_{ji}^{\text{rec},\imath}} = \sum_t \frac{dE}{dh_j^t} \mathscr{o}_j^t e_{ji}^{\text{rec},\imath,t} \ . \tag{34}$$

For the parameters $\theta_{ji}^{\text{rec},\mathscr{o}}$ of the output gate which take part in the function f, we need to derive the gradients in a different manner. As we still assume that $E(\mathbf{h}^0, \ldots, \mathbf{h}^T)$ depends on the observable state only, we can follow the derivation of BPTT with a formula analogous to (12). This results in a gradient expression involving local terms and the same learning signal

as used for other parameters. Writing $\frac{\partial f}{\partial \theta_{ji}^{\text{rec},o}}$ as $\frac{\partial h_j^t}{\partial \theta_{ji}^{\text{rec},o}}$ the gradient update for the output gate takes the form:

$$\frac{dE}{d\theta_{ji}^{\text{rec},o}} = \sum_t \frac{dE}{dh_j^t} \frac{\partial h_j^t}{\partial \theta_{ji}^{\text{rec},o}} = \sum_t \frac{dE}{dh_j^t} c_j^t o_j^t (1 - o_j^t) h_i^{t-1}. \tag{35}$$

Merge #1

Merge #1 follows the general *merge* framework and applies to all the models above. Its specificity is the choice of learning signal. In this first variant, we make two approximations: future errors are ignored so that one can compute the learning signal in real-time, and learning signals are fed back with random connections. The specific realizations of the two approximations are discussed independently in the following and implementation details are provided.

Ignoring future errors: The first approximation is to focus on the error at the present time t and ignore dependencies on future errors in the computation of the total derivative $\frac{dE}{dz_j^t}$. Using the chain rule, this total derivative expands as $\frac{dE}{dz_j^t} = \frac{\partial E}{\partial z_j^t} + \frac{dE}{d\mathbf{s}_j^{t+1}} \frac{\partial \mathbf{s}_j^{t+1}}{\partial z_j^t}$, and neglecting future errors means that we ignore the second term of this sum. As a result the total derivative $\frac{dE}{dz_j^t}$ is replaced by the partial derivative $\frac{\partial E}{\partial z_j^t}$ in equation (3).

Synaptic weight updates under *merge #1*: Usually, the output of an RNN is given by the output of a set of readout neurons which receive input from network neurons, weighted by synaptic weights θ_{kj}^{out}. In the case of an RSNN, in order to be able to generate non-spiking outputs, readouts are modeled as leaky artificial neurons. More precisely, the output of readout k at time t is given by

$$y_k^t = \xi y_k^{t-1} + \sum_j \theta_{kj}^{\text{out}} z_j^t + b_k^{\text{out}}, \tag{36}$$

where $\xi \geq 0$ defines the leak and b_k^{out} denotes the readout bias. The leak factor ξ is given by $e^{-\delta t/\tau_{out}}$, where δt is the discrete time step and τ_{out} is the membrane time constant). In the following derivation of weight updates under *merge #1*, we assume such readout neurons. Additionally, we assume that the error function is given by the mean squared error $E = \frac{1}{2}\sum_{t,k}(y_k^t - y_k^{*,t})^2$ with $y_k^{*,t}$ being the target output at time t (with minor differences the following analysis is also valid with the cross entropy error $E = -\sum_{t,k} p_k^{*,t} \log \sigma(y_k^t)$ and target target categories provided with the one-hot-encoded vector $p_k^{*,t}$).

In this case, the partial derivative $\frac{\partial E}{\partial z_i^t}$ has the form:

$$\frac{\partial E}{\partial z_i^t} = \theta_{kj}^{\text{out}} \sum_{t' \geq t} (y_k^{t'} - y_k^{*,t'}) \xi^{t'-t}. \tag{37}$$

This seemingly poses a problem for a biologically plausible learning rule, because the partial derivative is a weighted sum over the future. This issue can however easily be solved as we show below. Using equation (1) for the particular case of *merge #1*, we insert $\frac{\partial E}{\partial z_i^t}$ in-place of the total derivative $\frac{dE}{dz_i^t}$ which leads to an estimation $\widehat{\frac{dE}{d\theta_{ji}}}$ of the true gradient given by:

$$\widehat{\frac{dE}{d\theta_{ji}}} = \sum_t \frac{\partial E}{\partial z_i^t} \frac{\partial z_j^t}{\partial \mathbf{s}_j^t} e_{ji}^t \tag{38}$$

$$= \sum_{k,t} \theta_{kj}^{\text{out}} \sum_{t' \geq t} (y_k^t - y_k^{*,t}) \xi^{t'-t} \frac{\partial z_j^t}{\partial \mathbf{s}_j^t} e_{ji}^t \tag{39}$$

$$= \sum_{k,t'} \theta_{kj}^{\text{out}} (y_k^{t'} - y_k^{*,t'}) \sum_{t \leq t'} \xi^{t'-t} \frac{\partial z_j^t}{\partial \mathbf{s}_j^t} e_{ji}^t \ , \tag{40}$$

where we inverted sum indices in the last line. The second sum indexed by t is now over previous events and this gradient computation can be implemented in real time similarly to eligibility traces in equation (2).

Equation (40) holds for any neuron model. In the case of LIF neurons, the eligibility traces are given by equation (21), and one obtains the final expression of the error gradients after substituting these expressions in (40). Implementing weight updates with gradient descent and learning rate η, the updates of the recurrent weights are given by

$$\Delta\theta_{ji}^{\text{rec}} = \eta \sum_t \Big(\sum_k \theta_{kj}^{\text{out}} (y_k^{*,t} - y_k^t) \Big) \sum_{t' \leq t} \xi^{t-t'} p_j^{t'} \hat{z}_i^{t'-1} \ . \tag{41}$$

Random feed-back matrices: According to equations (40) and (41), the error signal from readout k communicated to neuron j has to be weighted with θ_{kj}^{out}. That is, the synaptic efficacies of the feedback synapses have to equal those of the feed-forward synapses. This general property of backpropagation-based algorithms is a problematic assumption for biological circuits. It has been shown however in (Samadi et al., 2017; Nøkland, 2016) that for many tasks, an approximation where the feedback weights are chosen randomly works well. We adopt this approximation in *merge #1*. Therefore we replace θ_{kj}^{out} in equation (41) with fixed random values B_{kj}^{random}, and the learning rule is finally written as

$$\Delta\theta_{ji}^{\text{rec}} = \eta \sum_t \Big(\sum_k B_{kj}^{\text{random}} (y_k^{*,t} - y_k^t) \Big) \sum_{t' \leq t} \xi^{t-t'} p_j^{t'} \hat{z}_i^{t'-1} \ . \tag{42}$$

Details to simulations for *Merge #1*

General simulation and neuron parameters: In all simulations of this article, networks were simulated in discrete time with a simulation time step of 1 ms. Neurons had no refractory period and synaptic delays were 1 ms. Synaptic weights of spiking neural networks were initialized as in (Bellec et al., 2018).

Implementation of the optimization algorithm: A dampening factor of $\gamma = 0.3$ for the pseudo-derivative of the spiking function was used in all simulations of this article. The weights were kept constant for n_{batch} independent trials (specified for individual simulations below), and the gradients were cumulated additively. After collecting the gradients, the weights were updated using the Adam optimizer (Kingma and Ba, 2014). For all simulations of *merge #1*, the gradients were computed according to equation (40).

Integration of the "Clopath rule" in *merge #1*: We replaced the presynpatic and postsynaptic factors of equation (21) with the model of long term potentation defined in Clopath et al. (2010) and fitted to data in the same paper. Using our notation the combination of their learning rule with *merge #1* becomes:

$$\frac{dE}{d\theta_{ji}^{\text{rec}}} = \sum_t \frac{dE}{dz^t} p_j^t [\hat{v}_j^t - v_{thr}^-]^+ \hat{z}_i^{t-1} \ , \tag{43}$$

where $\hat{v}_j^t$ is an exponential trace of the post synaptic membrane potential with time constant 10 ms and $[\cdot]^+$ is the rectified linear function. The time constant of $\hat{v}_j^t$ was chosen to match their data and the threshold v_{th}^- is chosen to be somewhere between resting potential $v_j = 0$ and the firing threshold v_{th}, we took $v_{\text{th}}/5$. Actually the authors had used a different term of the form $[v_j^t - v_{thr}^+]^+$ in place of p_j^t. But we argue that our term p_j^t is in practice equivalent to theirs because assuming $v_{\text{th}}^+ = 0$, p_j^t is linear function of the voltage for v_j^t between 0 and v_{thr} and 0 for lower voltages.

Pattern generation task #1.1: The three target sequences had a duration of 1000 ms and were given by the sum of four sinusoids for each sequence with fixed frequencies of 1 Hz, 2 Hz, 3 Hz, and 5 Hz. The amplitude of each sinusoidal component was drawn from a uniform distribution over the interval $[0.5, 2]$. Each component was also randomly phase-shifted with a phase sampled uniformly in the interval $[0, 2\pi)$.
The network consisted of 600 all-to-all recurrently connected LIF neurons (no adaptive thresholds). The membrane time constant τ_m was sampled from a uniform distribution over the interval $[10, 50]$ ms independently for each neuron. The firing threshold was set to $v_{\text{th}} = 0.02$. The network outputs were provided by the membrane potential of three readout neurons with

a time constant $\tau_{out} = 20$ ms. The network received input from 20 input neurons, divided into 5 groups, which indicated the current phase of the target sequence similar to (Nicola and Clopath, 2017). Neurons in group $i \in \{0, 4\}$ produced 100 Hz regular spike trains during the time interval $[200 \cdot i, 200 \cdot i + 200)$ ms and were silent at other times.
A single learning trial consisted of a simulation of the network for 1000 ms, i.e., the time to produce the target pattern at the output. The input, recurrent, and output weights of the network were trained for 1000 iterations with a learning rate of 0.003 and the Adam optimizer with $\epsilon = 10^{-8}$ and exponential decay rate for the first and second moment estimates 0.9 and 0.999 respectively. After 100 iterations, the learning rate was decayed with a multiplicative factor of 0.7. A batch size of a single trial was used for training. To avoid an implausibly high firing rate, a regularization term was added to the loss function, that keeps the neurons closer to a target firing rate of 10 Hz. The regularization loss was given by the mean squared error (mse) between the mean firing rate of all neurons over a batch and the target rate. This loss was multiplied with the factor 0.5 and added with the target-mse to obtain the total loss to be optimized.

The comparison algorithms in Figure 1d,e were implemented as follows. For the performance of the global error signal, *merge #1* was used, but the random feedback matrix was replaced by a matrix where all entries had the value $1/\sqrt{n}$, where n is the number of network neurons. Second, a network without recurrent connections was used, trained with *merge #1* ("No rec. conn." in panel d). We further considered *merge #1* where the feedback matrix was drawn afresh every 1 ms and every 20 ms ("1 ms" and "10 ms" in panels d and e). Note that here, the same sequence feedback matrices had to be used in every learning trial as otherwise optimization did not work. We also compared to BPTT, where the ADAM optimizer was used with the same meta-parameters as used for *merge #1*.

Store-recall task #1.2: The store-recall task is described in Results. Each learning trial consisted of a 2400 ms network simulation. We used a recurrent LSNN network consisting of 10 standard LIF neurons and 10 LIF neurons with adaptive thresholds. All neurons had a membrane time constant of $\tau_m = 20$ ms and a baseline threshold of $v_{\text{th}} = 0.01$. Adaptive neurons had a threshold increase constant of $\beta = 1.7$ and a threshold adaptation time constant of $\tau_a = 1200$ ms.
The input, recurrent and output weights of the network were trained for 300 iterations with a learning rate of 0.01 and the Adam optimizer with $\epsilon = 10^{-8}$ and exponential decay rate for the first and second moment estimates 0.9 and 0.999 respectively. Training was stopped prematurely when a stopping criterion of a misclassification rate below 0.05 was reached before. After 100 iterations, the learning rate was decayed with a multiplicative factor of 0.3. A batch size of 128 trials was used.

In Figure 2b, we quantified the information content of eligibility traces at training iteration 25, 75, and 200 in this task. After the predefined number of training iterations, we performed test simulations where we provided only a store command to the network and simulated the network up to 6000 ms after this store. A linear classifier (one for a time window of 100 ms, at every multiple of 50 ms) was then trained to predict the stored bit from the value of the eligibility traces at that time. For this purpose we used logistic regression with a squared regularizer on the weights. We used 150 different simulations to train the classifiers and evaluated the decoding accuracy, as shown in Figure 2b, on 50 separate simulations.

Speech recognition task #1.3: The TIMIT dataset was split according to Halberstadt (Glass et al., 1999) into a training, validation, and test set with 3696, 400, and 192 sequences respectively. The networks received preprocessed audio at the input. Preprocessing of audio input consisted of the following steps: computation of 13 Mel Frequency Cepstral Coefficients (MFCCs) with frame size 10 ms on input window of 25 ms, computation of the first and the second derivatives of MFCCs, concatenation of all computed factors to 39 input channels. Input channels were mapped to the range $[0, 1]$ according to the minumum/maximum values in the training set. These continuous values were used directly as inputs x_i^t in equation (18).

To tackle this rather demanding benchmark task, we used a bi-directional network architecture (Graves and Schmidhuber, 2005), that is, the standard LSNN network was appended by a second network which recieved the input sequence in reverse order. A bi-directional LSNN (300 LIF neurons and 100 adaptive LIF neurons per direction) was trained with dif-

ferent training algorithms. Every input step which represents the 10 ms preprocessed audio frame is fed to the LSNN network for 5 consecutive 1 ms steps. All neurons had a membrane time constant of $\tau_m = 20$ ms and a baseline threshold of $v_{\text{th}} = 0.01$. Adaptive neurons had $\beta = 1.8$ and an adaptation time constant of $\tau_a = 200$ ms. We used 61 readout neurons, one for each class of the TIMIT dataset. A softmax was applied to their output, which was used to compute the cross entropy error against the target label. Networks were trained using Adam with $\epsilon = 10^{-5}$, and exponential decay rate for the first and the second moment estimates 0.9 and 0.999 respectively. The learning rate was fixed to 0.01 during training. We used a batch size of 32 and the membrane time constant of the output neurons was 3 ms. Regularizaion of the network firing activity was applied as in Task #1.1.

Merge #2

In *merge #2*, the learning signals are computed in a separate error module. In order to distinguish the error module from the main network, we define a separate internal state vector for each neuron j in the error module $\boldsymbol{\sigma}_j^t$ and network dynamics $\boldsymbol{\sigma}_j^t = M_e(\boldsymbol{\sigma}_j^{t-1}, \boldsymbol{\zeta}^{t-1}, \boldsymbol{\xi}^t, \boldsymbol{\Psi})$ for it. Here, $\boldsymbol{\zeta}^{t-1}$ is the vector of neuron outputs in the error module at time t, and synaptic weights are denoted by $\boldsymbol{\Psi}$. The inputs to the error module are written as: $\boldsymbol{\xi}^t = (\boldsymbol{x}^t, \boldsymbol{z}^t, \boldsymbol{y}^{*,t})$ with $\boldsymbol{y}^{*,t}$ denoting the target signal for the network at time t. Note that the target signal is not necessarily the target output of the network, but can be more generally a target state vector of some controlled system. For example, the target signal in task #2.1 is the target position of the tip of an arm at time t, while the outputs of the network define the angular velocities of arm joints.

The error module produces at each time t a learning signal ℓ_j^t for each neuron j of the network, which were computed according to:

$$\ell_j^t = \alpha_e \ell_j^{t-1} + \sum_i \Psi j i^{\text{out}} \zeta_i^t, \tag{44}$$

where the constant α_e defines the decay of the resulting learning signal, e.g. the concentration of a neuromodulator.

Learning signal in LIF neurons: The task of the error module ℓ_j^t is to compute approximation of the gradient $\frac{dE}{dz_j^t}$ used in the definition of the learning signal in equation (3). Since we considered LIF neurons for all our experiments with *merge #2*, we identify the second factor with the pseudo-derivative p_j^t. Hence, in comparison to equation (21), this leads to an estimation of the true gradients $\frac{dE}{d\theta_{ji}^{\text{rec}}}$ given by: $\widehat{\frac{dE}{d\theta_{ji}^{\text{rec}}}} = \sum_t \ell_j^t p_j^t \hat{z}_i^{t-1}$.

Target movement task #2.1: In this task, the two network outputs are interpreted as angular velocities $\dot{\phi}_1$ and $\dot{\phi}_2$ and are applied to the joints of a simple arm model. The configuration of the arm model at time t is described by the angles ϕ_1^t and ϕ_2^t of the two joints measured against the horizontal and the first leg of the arm respectively, see Figure 3c. For given angles, the position $\boldsymbol{y}^t = (x^t, y^t)$ of the tip of the arm in Euclidean space is given by $x^t = l\cos(\phi_1^t) + l\cos(\phi_1^t + \phi_2^t)$ and $y^t = l\sin(\phi_1^t) + l\sin(\phi_1^t + \phi_2^t)$. Angles were computed by discrete integration over time: $\phi_i^t = \sum_{t' \le t} \dot{\phi}_i^{t'} \cdot 1\,\text{ms} + \phi_i^0$. The initial values were set to $\phi_1^0 = 0$ and $\phi_2^0 = \frac{\pi}{2}$.

Feasible target movements $\boldsymbol{y}^{*,t}$ of duration 500 ms were generated randomly by sampling the generating angular velocities $\dot{\Phi}^{*,t} = (\dot{\phi}_1^{*,t}, \dot{\phi}_2^{*,t})$. Each of the target angular velocities exhibitted a common form

$$\dot{\phi}_i^{*,t} = \sum_k A_k \sin\left(2\pi\omega_i \frac{t}{T} + \delta_i\right) \stackrel{\text{def}}{=} \sum_k q_k^t, \tag{45}$$

where the number of components k was set to 5, A_k was sampled uniformly in $[0, 30]$, ω_i was sampled uniformly in $[0.3, 1]$ and δ_i was sampled uniformly in $[0, 2\pi]$. After this sampling, every component q_k^t in $\dot{\phi}_2^{*,t}$ was rescaled to satisfy $\max_t(q_k^t) - \min_t(q_k^t) = 20$. In addition, we considered constraints on the angles of the joints: $\phi_1 \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ and $\phi_2 \in [0, \pi]$. If violated, the respective motor commands $\dot{\phi}_i^{*,t}$ were rescaled to match the constraints.

A clock-like input signal was implemented as in task #1.1 by 20 input neurons, that fired in groups in 5 successive time steps with a length of 100 ms at a rate of 100 Hz.

Outer loop optimization: The procedure described above defines an infinitely large familiy of tasks, each task of the family being one particular target movement. We optimized the parameters of the error module as well as the initial parameters of the learning network in an outer-loop optimization procedure. The learning cost $\mathcal{L}_C$ for tasks C in the above defined family of tasks was defined as $\mathcal{L}_C(\boldsymbol{\theta}_{\text{test},C}) = \sum_t \left(\left(\boldsymbol{y}^t(\boldsymbol{\theta}_{\text{test},C}) - \boldsymbol{y}^{*,t}\right)^2 + \left(\dot{\Phi}^t(\boldsymbol{\theta}_{\text{test},C}) - \dot{\Phi}^{*,t}\right)^2 \right)$ to measure how well the target movement was reproduced. We then optimized the expected cost over the family of learning task using BPTT. Gradients were computed over batches of 200 different tasks to estimate the expected cost $\mathbb{E}_{C\sim\mathcal{F}}\left[\mathcal{L}_C(\boldsymbol{\theta}_{\text{test},C})\right] \approx \frac{1}{200}\sum_{i=1}^{200} \mathcal{L}_{C_i}(\boldsymbol{\theta}_{\text{test},C_i})$. We used the Adam optimizer (Kingma and Ba, 2014) with a learning rate of 0.001. The learning rate decayed after every 1000 steps by a factor of 0.95. Regularization of both network and error module activity was applied as in task #1.1.

Parameters: The learning network consisted of 400 LIF neurons according to the model stated in equation (18) and (19), with a membrane time constant of 20 ms and a threshold of $v_{\text{th}} = 0.01$. The motor commands $\dot{\phi}_j^t$ predicted by the network were given by the output of readout neurons with a membrane time constant of 20 ms.

The error module was implemented as a recurrently connected network of 300 LIF neurons, which had the same membrane decay as the learning network. The neurons in the error module were set to have a threshold of $v_{\text{th}} = 0.03$. Readout neurons of the error module had a membrane time constant of 20 ms. Finally, the weight update with *merge* according to equation (6) used a learning rate of $\eta = 4 \cdot 10^{-4}$.

Linear error module: The alternative implementation of a linear error module was implemented as a linear mapping of inputs formerly received by the spiking implementation of the error module. Prior to the linear mapping, we applied a filter to the spiking quantities $\boldsymbol{x}^t$, $\boldsymbol{z}^t$ such that $\hat{\boldsymbol{x}}^t = \sum_{t'\leq t} \alpha_e^{t-t'} \boldsymbol{x}^t$ and similarly for $\hat{\boldsymbol{z}}^t$. Then, the learning signal from the linear error module was given as: $\ell_j^t = \sum_i \Phi_{ji}^x \hat{x}_i^t + \sum_i \Phi_{ji}^z \hat{z}_i^t + \sum_i \Phi_{ji}^y y_i^{*,t}$

Merge #3

We first describe *merge #3* in theoretical terms when the simulation duration is split into intervals of length Δt and show two mathematical properties of the algorithm: first, it computes the correct error gradients if the synthetic gradients are ideal; and second, when the synthetic gradients are imperfect, the estimated gradients are a better approximation of the true error gradient in comparison to BPTT. In subsequent paragraphs we discuss details of the implementation of *merge #3*, the computation of the synthetic gradients and hyper parameters used in tasks #3.1 and #3.2.

Notation and review of truncated BPTT: We consider the true error gradient $\frac{dE}{d\theta_{ji}}$ to be the error gradient computed over the full simulation ranging from time $t = 1$ to time T. Truncated BPTT computes an approximation of this gradient. In this paragraph, we identify the approximations induced by truncated BPTT.

In truncated BPTT, the network simulation is divided into K successive intervals of length Δt each. For simplicity we assume that T is a multiple of Δt, such that $K = T/\Delta t$ is an integer. Using the shorthand notation $t_k = k\Delta t$, the simulation intervals are thus $\{1, \ldots, t_1\}, \{t_1 + 1, \ldots, t_2\}, \ldots, \{t_{K-1} + 1, \ldots, t_K\}$. To simplify the theory we assume that updates are implemented after the processing of all these intervals (i.e., after time T).

For each interval $\{t_{k-1} + 1, \ldots, t_k\}$, the simulation is initialized with the network state $\mathbf{s}^{t_{k-1}}$. Then, the observable states $\mathbf{z}^{t'}$ and hidden states $\mathbf{s}^{t'}$ are computed for $t' \in \{t_{k-1} + 1, \ldots, t_k\}$. It is common to use for the overall error $E(\mathbf{z}_1, \ldots, \mathbf{z}_T)$ an error function that is given by the sum of errors in each individual time step. Hence the error can be written as a sum of errors $E_k(\mathbf{z}^{t_{k-1}+1}, \ldots, \mathbf{z}^{t_k})$ in the intervals:

$$E(\mathbf{z}^1, \ldots, \mathbf{z}^T) = \sum_{k=1}^{K} E_k(\mathbf{z}^{t_{k-1}+1}, \ldots, \mathbf{z}^{t_k}) \ . \tag{46}$$

For each such interval, after network simulation until t_k (the forward pass), the gradients $\frac{dE_k}{d\mathbf{s}_j^{t'}}$

are propagated backward from $t' = t_k$ to $t' = t_{k-1} + 1$ (the backward pass). The contribution to the error gradient for some paramter θ_{ji} in the interval is then given by (compare to equation (12))

$$g^{\text{trunc}}_{k,ji} = \sum_{t'=t_{k-1}+1}^{t_k} \frac{dE_k}{d\mathbf{s}_j^{t'}} \cdot \frac{\partial \mathbf{s}_j^{t'}}{\partial \theta_{ji}} \ . \tag{47}$$

The overall gradient $\frac{dE}{d\theta_{ji}}$ is then approximated by the sum of the gradients in the intervals: $g^{\text{trunc}}_{1,ji} + g^{\text{trunc}}_{2,ji} + \cdots + g^{\text{trunc}}_{K,ji}$.

This approximation is in general not equal to the true error gradient $\frac{dE}{d\theta_{ji}}$, as it disregards the contributions of network outputs within one interval on errors that occur in a later interval.

Synthetic gradients: To correct for the truncated gradient, one can provide a suitable boundary condition at the end of each interval that supplements the missing gradient. The optimal boundary condition cannot be computed in an online manner since it depends on future activities and future errors that are not yet available. In truncated BPTT, one chooses $\frac{dE}{d\mathbf{s}_j^{t_k+1}} = 0$ at the end of an interval $\{t_{k-1}+1, \ldots, t_k\}$, which is exact only if the simulation terminates at time t_k or if future errors do not depend on network states of this interval. The role of synthetic gradients is to correct this approximation by providing a black box boundary condition $\text{SG}_j(\mathbf{z}^{t_k}, \Psi)$, where SG_j is a parameterized function of the network output with parameters Ψ. SG_j should approximate the optimal boundary condition, i.e., $\text{SG}_j(\mathbf{z}^{t_k}, \Psi) \approx \frac{dE}{d\mathbf{s}_j^{t_k+1}}$.

We denote the approximate gradient that is computed by synthetic gradient by $\frac{d\overline{E}}{d\mathbf{s}_j^t}$. This gradient is given by

$$\frac{d\overline{E}_k}{d\boldsymbol{s}_j^t} = \frac{dE_k}{d\boldsymbol{s}_j^t} + \eta_{SG} \sum_m \text{SG}_m(\boldsymbol{z}^{t_k}, \Psi) \frac{d\boldsymbol{s}_m^{t_k+1}}{d\boldsymbol{s}_j^t}. \tag{48}$$

We will continue our theoretical analysis with a factor $\eta_{SG} = 1$ (as suggested in Jaderberg et al. (2016), we set η_{SG} to 0.1 in simulations to stabilize learning). We define $g^{\text{SG}}_{k,ji}$ as the corrected version of $g^{\text{trunc}}_{k,ji}$ that incorporates the new boundary condition. We finally define the estimator of the error gradient with synthetic gradients as:

$$\widehat{\frac{dE}{d\theta_{ji}}}^{\text{SG}} = g^{\text{SG}}_{1,ji} + g^{\text{SG}}_{2.ji} + \cdots + g^{\text{SG}}_{K,ji}. \tag{49}$$

The synthetic gradient approximator $SG_j(\mathbf{z}^{t_k}, \Psi)$ is optimized by minimizing the following synthetic gradient loss, which is the sum squared error between the synthetic gradient approximation and the gradient $\widehat{\frac{dE}{d\mathbf{s}_j^{t_k+1}}}$ computed in the next interval:

$$E_{SG}\left(\mathbf{z}^{t_k}, \widehat{\frac{dE}{d\mathbf{s}_j^{t_k+1}}}, \Psi\right) = \sum_j \frac{1}{2} \left\| SG_j(\mathbf{z}^{t_k}, \Psi) - \widehat{\frac{dE}{d\mathbf{s}_j^{t_k+1}}} \right\|^2 . \tag{50}$$

Correctness of synthetic gradients: We consider Ψ^* to be optimal synthetic gradient parameters if the synthetic gradient loss in equation (50) is always zero. In this case, all synthetic gradients $\text{SG}(\mathbf{z}^{t_k}, \Psi^*)$ exactly match $\frac{dE}{d\mathbf{s}_j^{t_k}}$, and the computed approximation exactly matches the true gradient. This analysis assumes the existence of the optimal parameters Ψ^* and the convergence of the optimization algorithm to the optimal parameters. This is not necessarily true in practice. For an analysis of the convergence of the optimization of the synthetic gradient loss we refer to (Czarnecki et al., 2017).

Proof of correctness of *merge #3* with truncated time intervals: Similarly to the justification above for synthetic gradients, we show now that the error gradients $\frac{dE}{d\theta_{ji}}$ can be estimated with *merge #3* when the gradients are computed over truncated intervals.

Instead of using the factorization of the error gradients as in BPTT (equation (12)), *merge #3* uses equation (1). The approximate gradient that is computed by *merge #3* with respect

to neuron outputs is given analogously to equation (48)

$$\frac{d\overline{E}_k}{dz_j^t} = \frac{dE_k}{dz_j^t} + \sum_m \mathrm{SG}_m(\boldsymbol{z}^{t_k}, \Psi) \frac{d\boldsymbol{s}_m^{t_k+1}}{dz_j^t}. \tag{51}$$

We are defining the learning signal as in equation (3), but now using the enhanced estimate of the derivative of the interval error:

$$\overline{\boldsymbol{L}}_{k,j}^t = \frac{d\overline{E}_k}{dz_j^t} \frac{dz_j^t}{d\mathbf{s}_j^t}. \tag{52}$$

This learning signal is computed recursively using equation (13) within an interval. At the upper boundaries t_k of the intervals, the boundary condition is computed via synthetic gradients.

Analogous to $g_{k,ji}^{\mathrm{SG}}$, we define the gradient approximation of *merge #3* $g_{k,ji}^{\mathrm{merge}}$ as the corrected version of $g_{k,ji}^{\mathrm{trunc}}$ that incorporates the boundary condition for interval $\{t_{k-1}+1, \ldots, t_k\}$ via synthetic gradients. This gradient approximation is given by

$$g_{k,ji}^{\mathrm{merge}} = \sum_{t=t_{k-1}+1}^{t_k} \overline{\mathbf{L}}_{k,j}^t \cdot \mathbf{e}_{ji}^t \ . \tag{53}$$

Considering the sum of terms $g_{k,ji}^{\mathrm{merge}}$ associated with each interval, we write the estimator of the true error gradient computed with *merge #3* as:

$$\widehat{\frac{dE}{d\theta_{ji}}}^{\mathrm{merge}} = g_{1,ji}^{\mathrm{merge}} + g_{2,ji}^{\mathrm{merge}} + \cdots + g_{K,ji}^{\mathrm{merge}} \ . \tag{54}$$

Assuming now that this boundary condition is provided by an error module computing the synthetic gradients SG with optimal parameters Ψ^*. As explained above, it follows that all $\mathrm{SG}_m(\mathbf{z}^{t_k}, \Psi^*)$ compute exactly $\frac{dE}{d\mathbf{s}_m^{t_k+1}}$ which is true independently of the usage of BPTT or *merge #3*. In the later case, it follows that $\frac{d\overline{E}_k}{dz_j^t}$ is correctly computing $\frac{dE}{dz_j^t}$ and hence, $\overline{\boldsymbol{L}}_{k,j}^t$ is equal to the true learning signal $\boldsymbol{L}_j^t$. Looking back at equation (1), it follows that the estimator defined at equation (54) is equal to the true gradient if the parameters of the error module are optimal.

Optimization of the synthetic gradient parameters Ψ: We define here the algorithm used to optimize the synthetic gradients parameters Ψ and the network parameters $\boldsymbol{\theta}$. Using the same truncation scheme as described previously, we recall that the loss function $\overline{E}_k$ formalizes the loss function on interval k denoted E_k with the modification that it takes in to account the boundary condition defined by the synthetic gradients. We then consider the loss E' as the sum the term $\overline{E}_k$ and the synthetic gradient loss E_{SG}. The final algorithm is summarized by the pseudo-code given in Algorithm 1. Note that this algorithm is slightly different from the one used originally by Jaderberg et al. (2016). Our version requires one extra pair of forward and backward passes on each truncated interval but we found it easier to implement.

Copy-repeat task #3.1: Each sequence of the input of the copy repeat task consists of the "8-bit" pattern of length n_{pattern} encoded by 8 binary inputs, a stop character encoded by a 9th binary input channel, and a number of repetitions $n_{\mathrm{repetitions}}$ encoded using a one hot encoding over the 9 input channels. While the input is provided, no output target is defined. After that the input becomes silent and the output target is defined by the $n_{\mathrm{repetitions}}$ copies of the input followed by a stop character. As for the input, the output pattern is encoded with the first 8 output channels and the 9-th channel is used for the stop character. Denoting the target output $b_k^{*,t}$ of the channel k at time t and defining $\sigma(y_k^t)$ as the output of the network with y_k^t a weighted sum of the observable states z_j^t and σ the sigmoid function, the loss function is defined by the binary cross-entropy loss: $E = -\sum_{t,k}(1-b_k^{*,t}) \log_2 \sigma(y_k^t) + b_k^{*,t} \log_2 \left(1 - \sigma(y_j^t)\right)$. The sum is running over the time steps where the output is specified.

We follow the curriculum of Jaderberg et al. (2016) to increase gradually the complexity of the task: when the error E averaged over a batch of 256 sequences is below 0.15 bits per

for $t \in \{\Delta t, 2\Delta t, \cdots T\}$ **do**
  Simulate the network over the interval $[t - \Delta t, t]$ to compute the network states $\mathbf{s}_j^t$
  Apply back-propagation on the interval $[t, t + \Delta t]$ to compute the baked-up gradient $\widehat{\frac{d\overline{E}_{k+1}}{d\mathbf{s}_j^t}}$ that sets the boundary condition at time $t + \Delta t$,
  Apply back-propagation on the interval $[t - \Delta t, t]$ to compute $\frac{d\overline{E}_k}{d\boldsymbol{\theta}}$, $\frac{d\overline{E}_k}{d\Psi}$ and the variables necessary to continue to the next interval: $\mathbf{s}_j^t$ and $\mathbf{e}_{ji}^t$.
  Update the parameters Ψ and $\boldsymbol{\theta}$ using $\frac{dE'}{d\boldsymbol{\theta}}$, $\frac{dE'}{d\Psi}$ and any variant of stochastic gradient descent
end

Algorithm 1: Pseudo code to describe the algorithm used to trained simultaneously the network parameters $\boldsymbol{\theta}$ and the synthetic gradients Ψ in both *merge #3* and BPTT with synthetic gradients.

sequences, n_{pattern} or $n_{\text{repetitions}}$ are incremented by one. When the experiments begins, we initialize n_{pattern} and $n_{\text{repetitions}}$ to one. After the first threshold crossing n_{pattern} is incremented, then the increments are alternating between n_{pattern} and $n_{\text{repetitions}}$.

For each batch of 256 sequences, the parameters are updated every $\Delta t = 4$ time steps when the simulation duration in truncated as in BPTT. The parameter updates are applied with Adam, using learning rate 0.0001 and the default hyper-parameters suggested by Kingma and Ba (2014).

Word prediction task #3.2: Training was performed for 20 epochs, where one epoch denotes a single pass through the complete dataset. All learning rules used gradient descent to minimize loss with initial learning rate of 1 which was decayed after every epoch with factor 0.5, starting with epoch 5. Mini-batch consisted of 20 sequences of length Δt. Sequence of sentences in Penn Treebank dataset are connected and coherent, so the network state was reset only after every epoch. Equally the eligibility traces are set to zero at the beginning of every epoch.

Acknowledgments

This research/project was supported by the HBP Joint Platform, funded from the European Union's Horizon 2020 Framework Programme for Research and Innovation under the Specific Grant Agreement No. 785907 (Human Brain Project SGA2). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro P6000 GPU used for this research. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC) on the GCS Supercomputer JUWELS at Jülich Supercomputing Centre (JSC). The computational results presented have been achieved in part using the Vienna Scientific Cluster (VSC). We gratefully acknowledge Arjun Rao for his contribution to the software used in our experiments. We also want to thank him, Anand Subramoney, Michael Müller and Christoph Stoeckl for comments on earlier version of the manuscript.

References

Barrett, D. G., Hill, F., Santoro, A., Morcos, A. S., and Lillicrap, T. (2018). Measuring abstract reasoning in neural networks. *arXiv preprint arXiv:1807.04225*.

Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., and Maass, W. (2018). Long short-term memory and learning-to-learn in networks of spiking neurons. In *32nd Conference on Neural Information Processing Systems (NeurIPS 2018), arXiv preprint arXiv:1803.09574*.

Brea, J. and Gerstner, W. (2016). Does computational neuroscience need new synaptic learning paradigms? *Current Opinion in Behavioral Sciences*, (11):61–66.

Buzsaki, G. (2006). *Rhythms of the Brain*. Oxford University Press.

Buzzell, G. A., Richards, J. E., White, L. K., Barker, T. V., Pine, D. S., and Fox, N. A. (2017). Development of the error-monitoring system from ages 9–35: Unique insight provided by mri-constrained source localization of eeg. *Neuroimage*, 157:13–26.

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

Czarnecki, W. M., Świrszcz, G., Jaderberg, M., Osindero, S., Vinyals, O., and Kavukcuoglu, K. (2017). Understanding synthetic gradients and decoupled neural interfaces. *arXiv preprint arXiv:1703.00522*.

D'Angelo, E., Mapelli, L., Casellato, C., Garrido, J. A., Luque, N., Monaco, J., Prestori, F., Pedrocchi, A., and Ros, E. (2016). Distributed circuit plasticity: new clues for the cerebellar mechanisms of learning. *The Cerebellum*, 15(2):139–151.

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

Duan, Y., Schulman, J., Chen, X., Bartlett, P. L., Sutskever, I., and Abbeel, P. (2016). Rl2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*.

Engelhard, B., Finkelstein, J., Cox, J., Fleming, W., Jang, H. J., Ornelas, S., Koay, S. A., Thiberge, S., Daw, N., Tank, D., et al. (2018). Specialized and spatially organized coding of sensory, motor, and cognitive variables in midbrain dopamine neurons. *bioRxiv*, page 456194.

Frémaux, N. and Gerstner, W. (2016). Neuromodulated spike-timing-dependent plasticity, and theory of three-factor learning rules. *Frontiers in neural circuits*, 9:85.

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

Gehring, W. J., Goss, B., Coles, M. G., Meyer, D. E., and Donchin, E. (1993). A neural system for error detection and compensation. *Psychological science*, 4(6):385–390.

Gerstner, W., Lehmann, M., Liakoni, V., Corneil, D., and Brea, J. (2018). Eligibility traces and plasticity on behavioral time scales: Experimental support of neohebbian three-factor learning rules. *arXiv preprint arXiv:1801.05219*.

Glass, J., Smith, A., and K. Halberstadt, A. (1999). Heterogeneous acoustic measurements and multiple classifiers for speech recognition.

Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610.

Graves, A., Wayne, G., and Danihelka, I. (2014). Neural turing machines. *arXiv preprint arXiv:1410.5401*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

Hosp, J. A., Pekanovic, A. Rioult-Pedotti, M. S., and Luft, A. R. (2011). Dopaminergic projections from midbrain to primary motor cortex mediate motor skill learning. *The Journal of Neuroscience*, 31(7):2481–24887.

Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., and Kavukcuoglu, K. (2016). Decoupled neural interfaces using synthetic gradients. *arXiv preprint arXiv:1608.05343*.

Kaiser, J., Mostafa, H., and Neftci, E. (2018). Synaptic plasticity dynamics for deep continuous local learning. *arXiv preprint arXiv:1811.10766*.

Kandel, E. R., Schwartz, J. H., Jessell, T. M., of Biochemistry, D., Jessell, M. B. T., Siegelbaum, S., and Hudspeth, A. (2000). *Principles of neural science*, volume 4. McGraw-hill New York.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lake, B. M., Ullman, T. D., B., T. J., and Gershman, S. J. (2017). Building machines that learn and think like peole. *Behavioral and Brain Sciences*, 40.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *nature*, 521(7553):436.

Lillicrap, T. P., Cownden, D., Tweed, D. B., and Akerman, C. J. (2016). Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7:13276.

Lorente de Nó, R. (1938). Architectonics and structure of the cerebral cortex. *Physiology of the nervous system*, pages 291–330.

MacLean, S. J., Hassall, C. D., Ishigami, Y., Krigolson, O. E., and Eskes, G. A. (2015). Using brain potentials to understand prism adaptation: the error-related negativity and the p300. *Frontiers in human neuroscience*, 9:335.

Nayebi, A., Bear, D., Kubilius, J., Kar, K., Ganguli, S., Sussillo, D., DiCarlo, J. J., and Yamins, D. L. (2018). Task-driven convolutional recurrent models of the visual system. In *Advances in Neural Information Processing Systems*, pages 5291–5302.

Nevian, T. and Sakmann, B. (2006). Spine ca2+ signaling in spike-timing-dependent plasticity. *Journal of Neuroscience*, 26(43):11001–11013.

Ngezahayo, A., Schachner, M., and Artola, A. (2000). Synaptic activity modulates the induction of bidirectional synaptic changes in adult mouse hippocampus. *Journal of Neuroscience*, 20(7):2451–2458.

Nicola, W. and Clopath, C. (2017). Supervised learning in spiking neural networks with force training. *Nature communications*, 8(1):2208.

Nøkland, A. (2016). Direct feedback alignment provides learning in deep neural networks. In *Advances in neural information processing systems*, pages 1037–1045.

Pi, H., Hangya, B., Kvitsiani, D., Sanders, J. I., Huang, Z. J., and Kepecs, A. (2013). Cortical interneurons that specialize in disinhibitory control. *Nature*, 503(7477):521–524.

Samadi, A., Lillicrap, T. P., and Tweed, D. B. (2017). Deep learning with dynamic spiking neurons and fixed feedback weights. *Neural computation*, 29(3):578–602.

Schemmel, J., Briiderle, D., Griibl, A., Hock, M., Meier, K., and Millner, S. (2010). A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Circuits and systems (ISCAS), proceedings of 2010 IEEE international symposium on*, pages 1947–1950. IEEE.

Sjöström, P. J., Turrigiano, G. G., and Nelson, S. B. (2001). Rate, timing, and cooperativity jointly determine cortical synaptic plasticity. *Neuron*, 32(6):1149–1164.

Sugihara, H., Chen, N., and Sur, M. (2016). Cell-specific modulation of plasticity and cortical state by cholinergic inputs to the visual cortex. *Journal of Physiology*, 110(1-2):37–43.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. (2016). Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*.

Wang, Z., Joshi, S., Savel'ev, S., Song, W., Midya, R., Li, Y., Rao, M., Yan, P., Asapu, S., Zhuo, Y., et al. (2018). Fully memristive neural networks for pattern classification with unsupervised learning. *Nature Electronics*, 1(2):137.

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

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

Yang, Y., Yin, M., Yu, Z., Wang, Z., Zhang, T., Cai, Y., Lu, W. D., and Huang, R. (2017). Multifunctional nanoionic devices enabling simultaneous heterosynaptic plasticity and efficient in-memory boolean logic. *Advanced Electronic Materials*, 3(7):1700032.

Zenke, F. and Ganguli, S. (2018). Superspike: Supervised learning in multilayer spiking neural networks. *Neural computation*, 30(6):1514–1541.