A solution to the learning dilemma for recurrent networks of spiking neurons

Guillaume Bellec[1,◦], Franz Scherr[1,◦], Anand Subramoney[1], Elias Hajek[1], Darjan Salaj[1], Robert Legenstein[1], Wolfgang Maass[1,*]

[1]Institute of Theoretical Computer Science, Graz University of Technology, Inffeldgasse 16b, Graz, Austria

◦ First authors.

* To whom correspondence should be addressed; E-mail: maass@igi.tugraz.at.

Recurrently connected networks of spiking neurons underlie the astounding information processing capabilities of the brain. But in spite of extensive research, it has remained open how learning through synaptic plasticity could be organized in such networks. We argue that two pieces of this puzzle were provided by experimental data from neuroscience. A new mathematical insight tells us how they need to be combined to enable network learning through gradient descent. The resulting learning method – called *e-prop* – approaches the performance of *BPTT* (backpropagation through time), the best known method for training recurrent neural networks in machine learning. But in contrast to *BPTT*, *e-prop* is biologically plausible. In addition, it elucidates how brain-inspired new computer chips – that are drastically more energy efficient – can be enabled to learn.

Introduction

Networks of neurons in the brain differ in at least two essential aspects from deep networks in machine learning: They are recurrently connected by synapses, forming a giant number of loops, and they communicate via asynchronously emitted stereotypical electrical pulses, called spikes, rather than bits or numbers that are produced in a synchronized manner by each layer. Models that capture primary information processing capabilities of spiking neurons in the brain are well known, and we consider the arguably most prominent one: leaky integrate-and-fire (LIF) neurons, where spikes that arrive from other neurons through synaptic connections are multiplied with the corresponding synaptic weight, and are linearly integrated by a leaky membrane potential. The neuron fires – i.e., emits a spike – when the membrane potential reaches a firing threshold.

An important open problem is how recurrent networks of spiking neurons (RSNNs) can learn, i.e., how their synaptic weights can be modified by local rules for synaptic plasticity so that the computational performance of the network improves. In deep learning this problem is solved for feedforward networks through gradient descent for a loss function E that measures imperfections of current network performance (*LeCun et al., 2015*). Gradients of E are propagated backwards through all layers of the feedforward networks to each synapse through a process called backpropagation. Recurrently connected networks can compute more efficiently because each neuron can participate several times in a network computation, and they are able to solve tasks that require integration of information over time and a suitable timing of network outputs according to task demands. But since a synaptic weight can affect the network computation at several time points during a computation, its impact on the loss function (see Fig. 1A) is more indirect, and learning through gradient descent becomes substantially more difficult in a recurrent network. In machine learning one had solved this problem 30 years ago by unrolling

a recurrent network into a virtual feedforward network, see Fig. 1B, and applying the backpropagation algorithm to it (Fig. 1C). This learning method for recurrent neural networks is called backpropagation through time (*BPTT*).

We show that with a careful choice of the pseudo-derivative for handling the discontinuous dynamics of spiking neurons one can apply this learning method also to RSNNs, yielding the by far best performing learning algorithm for such networks (see (*Huh and Sejnowski, 2018*) for related preceding results). But the dilemma is that *BPTT* requires storing the intermediate states of all neurons during a network computation, and to merge these in a subsequent offline process with gradients that are computed backwards in time (see Fig. 1C and Movie S2). This makes it very unlikely that *BPTT* is used by the brain (*Lillicrap and Santoro, 2019*). This dilemma is exacerbated by the fact that neurons in the brain have a repertoire of additional internal dynamic processes on slower time scales that are not reflected in the LIF model, but which are likely to contribute to the superior capabilities of RSNNs in the brain to compute in the temporal domain. In fact, even in machine learning one uses special types of neuron models, called LSTM (Long Short-Term Memory) units, in order to handle such tasks. But any neuron model that has additional internal processes, and hence more hidden variables that capture their current state, makes learning in a recurrent network of such neurons even more difficult.

We present an approach for solving this dilemma: *e-prop* (Fig. 1D and 1E, see Movie S3). It can be applied not only to RSNNs, but also to recurrent networks of LSTM units and most other types of recurrent neural networks. We focus on the application of *e-prop* to RSNNs that have, besides LIF neurons, also a more sophisticated form of LIF neurons, called ALIF neurons. An ALIF neuron has a second hidden variable besides its membrane potential: an adaptive firing threshold. The firing threshold of an ALIF neuron increases through each ot its spikes and decays back to a resting value between spikes. This models firing rate adaptation, a well known feature of a fraction of neurons in the brain (*Allen Institute: Cell Types Database,*

2018) that dampens their firing activity. We refer to an RSNN that contains a fraction of ALIF neurons as a Long short-term memory Spiking Neural Network (LSNN), because we show that ALIF neurons provide a qualitative jump in temporal computing capabilities of RSNNs, allowing RSNNs to approach for the first time the performance of LSTM networks in machine learning for temporal processing tasks.

E-prop is motivated by two streams of experimental data from neuroscience that can be seen as providing hints how the brain solves the learning dilemma for RSNNs:

i) The dynamics of neurons in the brain is enriched by continuously ongoing updates of traces of past activity on the molecular level, for example in the form of calcium ions or activated CaMKII enzymes (*Sanhueza and Lisman, 2013*). These traces in particular record events where the presynaptic neuron fired before the postsynaptic neuron, which is known to induce Hebbian-like STDP (spike timing dependent plasticity) if followed by a top-down learning signal (*Cassenaer and Laurent, 2012, Yagishita et al., 2014, Gerstner et al., 2018*). We refer to local traces of this type as eligibility traces in our learning model.

ii) In the brain there exists an abundance of top-down signals such as dopamine and acetylcholine, to name only a few, that inform local populations of neurons about sub-optimal performance of brain computations. Interestingly some of these signals are of a predictive nature, e.g. they predict upcoming rewards in the case of dopamine or movement errors in the case of the error-related negativity (ERN), see (*MacLean et al., 2015*). Furthermore both dopamine signals (*Engelhard et al., 2019, Roeper, 2013*) and ERN-related neural firing (*Sajad et al., 2019*) are reported to be specific for a target population of neurons, rather than global. We refer to such top-down signals as learning signals in our learning model.

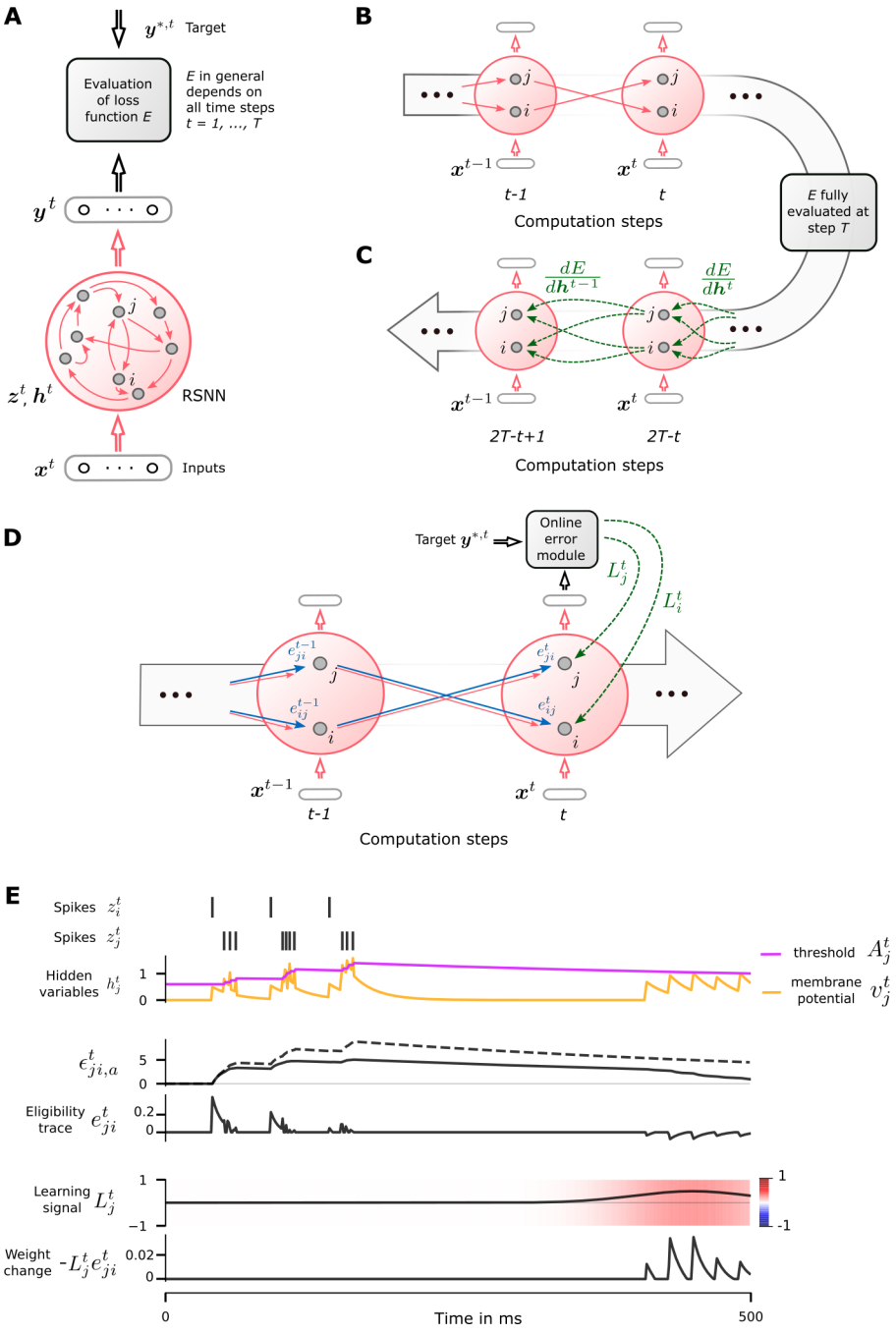

Figure 1: (Caption on the next page.)

Figure 1: **Schemes for RSNNs, *BPTT*, and *e-prop*.** **A**) RSNN with network inputs x, neuron spikes z, and output targets y^*, for each time step t of the RSNN computation. Output neurons y provide a low-pass filter of network spike z. **B**) BPTT computes gradient in the unrolled version of the network. It has a copy of all neurons of the RSNN for each time step t. A synaptic connection from neuron i to neuron j of the RSNN is replaced by an array of feedforward connections, one for each time step t, that goes from the copy of neuron i in the layer for time step t to a copy of neuron j in the layer for time step $t+1$. All synapses in this array have the same weight: the weight of this synaptic connection in the RSNN. **C**) Loss gradients of *BPTT* are propagated backwards in time and retrograde across synapses in an offline manner, long after the forward computation has passed a layer. **D**) Online learning dynamics of *e-prop*. Feedforward computation of eligibility traces is indicated in blue. These are combined with online learning signals according to equ. (1). **E**) Illustration of the dynamics of ALIF neurons and e-prop. Observable variables (spikes) $\boldsymbol{z}^t$ and hidden variables of an ALIF neuron, slow factor $\epsilon^t_{ji,a}$ (equation (22)) of the eligibility trace e^t_{ji} (equation (23)) of the synapse from neuron i to neuron j, as well as a learning signal L^t_j and the resulting online weight change proposed by e-prop. In this case a late activation of a learning signal, such as dopamine in the experiments of (*Yagishita et al., 2014*), it transforms the eligibility trace into the modification of the synaptic weight. The dashed curve above the plot of $\epsilon^t_{ji,a}$ shows an easily computable approximation (see equation (24)) of $\epsilon^t_{ji,a}$ as low-pass filter of STDP-inducing spiking events that can be used for an approximation of *e-prop*.

Our re-analysis of the mathematical basis of gradient descent in recurrent neural networks in equ. (1) tells us how eligibility traces and learning signals need to be combined to produce network learning through gradient descent – without backprogation of signals through time or retrograde through synaptic connections. We will show that the resulting new learning method, *e-prop*, approximates the performance of *BPTT* for RSNNs, thereby providing a solution to the learning dilemma for RSNNs. We demonstrate this on tasks for supervised learning (Fig. 2,3) and reinforcement learning (Fig. 4). None of these tasks were previously known to be solvable by RSNNs.

The previously described learning dilemma for RSNNs also affects the development of new, brain inspired computing hardware, which aims at a drastic reduction in the energy consumption of computing and learning. Resulting new designs of computer chips, such as Intels Loihi (*Davies et al., 2018*), are usually focused on RSNN architectures. On-chip learning capa-

bility for these RSNNs in the hardware is essential. Although it does not matter here whether the learning algorithm is biologically plausible, the excessive storage and offline processing demands of BPTT make this option unappealing for such novel computing hardware also. Hence a corresponding learning dilemma exists also there. *E-prop* does not contain any features that make it unlikely to be implementable on such neuromorphic chips, thereby promising a solution also for this learning dilemma.

Results

Mathematical basis for *e-prop*

Spikes are modeled as binary variables z_j^t that assume value 1 if neuron j fires at time t, otherwise value 0. It is common to let t vary over small discrete time steps, e.g. of 1ms length. The goal of network learning is to find synaptic weights W that minimize a given loss function E. E may depend on all or a subset of the spikes in the network. E measures in the case of regression or classification learning the deviation of the actual output y_k^t of each output neuron k at time t from its given target value $y_k^{*,t}$ (Fig. 1A). In reinforcement learning (RL), the goal is to optimize the behavior of an agent in order to maximize obtained rewards. In this case, E measures deficiencies of the current agent policy to collect rewards.

The gradient $\frac{dE}{dW_{ji}}$ for the weight W_{ji} of the synapse from neuron i to neuron j tells us how this weight should be changed in order to reduce E. The key observation for *e-prop* (see proof in Methods) is that this gradient can be represented as a sum over the time steps t of the RSNN computation: A sum of products of learning signals L_j^t (specific for the post-synaptic neuron j of the corresponding synapse) and synapse-specific eligibility traces e_{ji}^t:

$$\frac{dE}{dW_{ji}} = \sum_t L_j^t \, e_{ji}^t \, . \tag{1}$$

The ideal value of L_j^t is the derivative $\frac{dE}{dz_j^t}$, which tells us how the current spike output z_j^t of neuron j affects E. In contrast, the eligibility trace e_{ji}^t does not depend on E, but on the internal dynamics of neuron j. It tells us how a change of the weight W_{ji} would affect its spike output z_j^t via the temporal evolution of the hidden variables of neuron j, without considering recurrent loops formed with other neurons (see equation (S2) in supplementary materials).

We view (1) as a program for online learning: In order to reduce E, change at each step t of the network computation all synaptic weights W_{ji} proportionally to $-L_j^t e_{ji}^t$ (see Fig. 1E for an illustration). There is no need to explicitly compute or store the sum (1), or to wait for later signals. Hence *e-prop* is an online learning method in a strict sense (see Fig. 1D and Movie S3). In particular, there is no need to unroll the network as for *BPTT*. Furthermore, in contrast to the previously known real time recurrent learning algorithm (RTRL, see (*Williams and Zipser, 1989*) and Methods), which substantially increases the required number of multiplications as function of network size, *e-prop* is – up to a constant factor – not more costly than the RSNN computation itself. This is obviously an important issue both for biological plausibility and neuromorphic implementations.

Since the ideal value $\frac{dE}{dz_j^t}$ of the learning signal L_j^t also captures influences which the current spike output z_j^t of neuron j may have on E via future spikes of other neurons, its precise value is in general not available at time t. We replace it by an approximation that ignores these indirect influences: Only currently arising errors at the output neurons k of the RSNN are taken into account, and are routed with neuron-specific weights B_{jk} to the network neurons j, (see Fig. 2A):

$$L_j^t = \sum_k B_{jk} \underbrace{(y_k^t - y_k^{*,t})}_{\substack{\text{error of output } k \\ \text{at time } t}} . \tag{2}$$

Although this signal L_j^t only captures errors that arise at the current time step t, it is combined in equation (1) with an eligibility trace e_{ji}^t that may reach far back into the past of the target

neuron j (see Fig.1E). In this way *e-prop* alleviates the need to propagate signals backwards in time.

There are several strategies for choosing the weights B_{jk} for this online learning signal. In *symmetric e-prop* we set it equal to the corresponding output weight W^{out}_{kj} from neuron j to output neuron k. This learning signal is closest to the theory, and would be theoretically optimal in the absence of recurrent connections. Biologically more plausible are two variants that avoid weight sharing: If all network neurons j are connected to output neurons k, we let B_{jk} evolve in *adaptive e-prop* through a simple local plasticity rule that mirrors the plasticity rule applied to W^{out}_{kj}. In *random e-prop* the values of the weights B_{jk} are randomly chosen and remain fixed, similar to broadcast alignment for feedforward networks (*Lillicrap et al., 2016, Nøkland, 2016*). Resulting synaptic plasticity rules (see Methods) look very similar to previously proposed plasticity rules (*Gerstner et al., 2018*). In particular they involve postsynaptic depolarization as one of the factors, similarly as the data-based rule in (*Clopath et al., 2010*), see section S6 in the supplement for an analysis.

We finally would like to mention that the Learning-to-Learn approach can be used to train a separate neural network to generate – instead of the previously considered options – tailormade learning signals for a limited range of potential learning tasks. This variation of *e-prop* enables for example one-shot learning of new arm movements (*Bellec et al., 2019*).

Comparing the performance of *e-prop* and *BPTT* on a common benchmark task

The speech recognition task TIMIT (*Garofolo et al., 1993*) is one of the most commonly used benchmarks for temporal processing capabilities of different types of recurrent neural networks and different learning approaches (*Greff et al., 2017*). It comes in two versions. Both use, as input, acoustic speech signals from sentences that are spoken by 630 speakers from 8 dialect

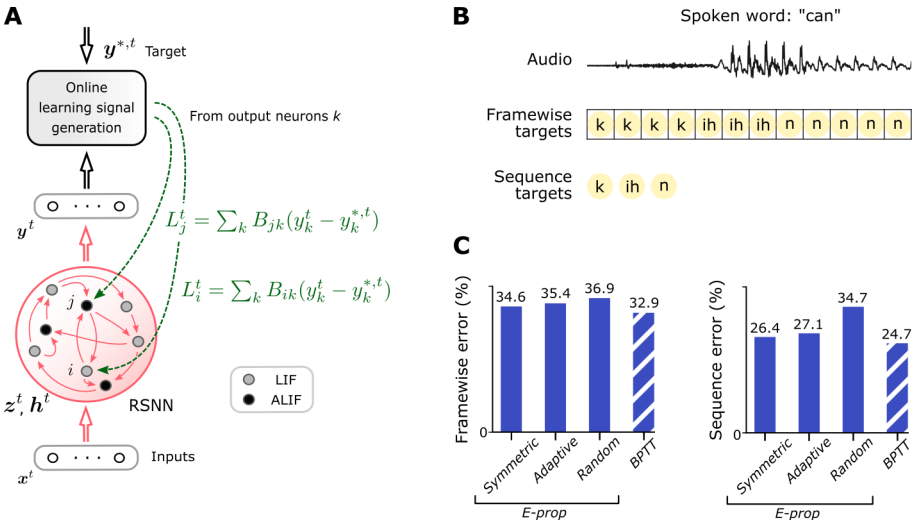

Figure 2: **Comparison of the performance of *BPTT* and *e-prop* on TIMIT. A**) Network architecture for *e-prop*, illustrated for an LSNN consisting of LIF and ALIF neurons. **B**) Input and target output for the two versions of TIMIT. **C**) Performance of *BPTT* and the three versions of *e-prop* for LSNNs consisting of 800 neurons for framewise targets and 2400 for sequence targets.

regions of the USA (see the top of Fig. 2B for a sample segment). In the simpler version, used for example in (*Greff et al., 2017*), the goal is to recognize which of 61 phonemes is spoken in each 10 ms time frame ("frame-wise classification"). In the harder version from (*Graves et al., 2013*), which achieved an essential step toward human-level performance in speech-to-text transcription, the goal is to recognize the sequence of phonemes in the entire spoken sentence independently of their timing ("sequence transcription"). *E-prop* approximates the performance of *BPTT* on LSNNs for both versions of TIMIT very well, as shown in Fig. 2C. For the more difficult version of TIMIT we trained as in (*Graves et al., 2013*) a complex LSNN consisting of a feedforward sequence of three recurrent networks. Our results show that *e-prop* can also handle learning for such more complex network structures very well. In Fig. S2 we show for comparison also the performance of LSTM networks. These data show that for both versions of TIMIT the performance of LSNNs comes rather close to that of LSTM networks.

This has previously not been demonstrated for any type of RSNN with any learning method on a real-world benchmark task for temporal processing. The FORCE method of (*Nicola and Clopath, 2017*) is the best performing previously known learning method for RSNNs. However this learning method was not argued to be biologically realistic, since the plasticity rule for each synaptic weight required knowledge of the current values of all other synaptic weights in the RSNNs. It was applied in (*Nicola and Clopath, 2017*) to supervised learning of several pattern generation task. We show in Figs. S1 and S5 that RSNNs can learn such tasks also with *e-prop*, hence without the biologically unrealistic feature of FORCE. We show in Fig S2 that *e-prop* can not only be applied to RSNNs, but also to LSTM networks – and many other types of recurrent networks – that fit under the quite general model discussed in Methods. Furthermore, *e-prop* approximates the performance of BPTT very well for LSTM networks as well (Fig. S2).

E-prop performance for a task where temporal credit assignment is difficult

A hallmark of cognitive computations in the brain is the capability to go beyond a purely reactive mode, to integrate diverse sensory cues over time, and to wait until the right moment arrives for an action. A large number of experiments in neuroscience analyze neural coding after learning for such tasks. But it had remained unknown how one can model the underlying learning processes in RSNNs of the brain. We wondered whether *e-prop* can fill this void. As an example we consider the task that was studied in the experiments of (*Morcos and Harvey, 2016, Engelhard et al., 2019*). There a rodent learnt to run along a linear track in a virtual environment, where it encountered several visual cues on the left and right, see Fig. 3A and Movie S2. Later, when it arrived at a T-junction, it had to decide whether to turn left or right. It was rewarded when it turned to that side from which it had previously received the majority of visual cues. This task is not easy to learn since the subject needs to find out that it does not matter on which side the last cue was, or in which order the cues were presented. Instead,

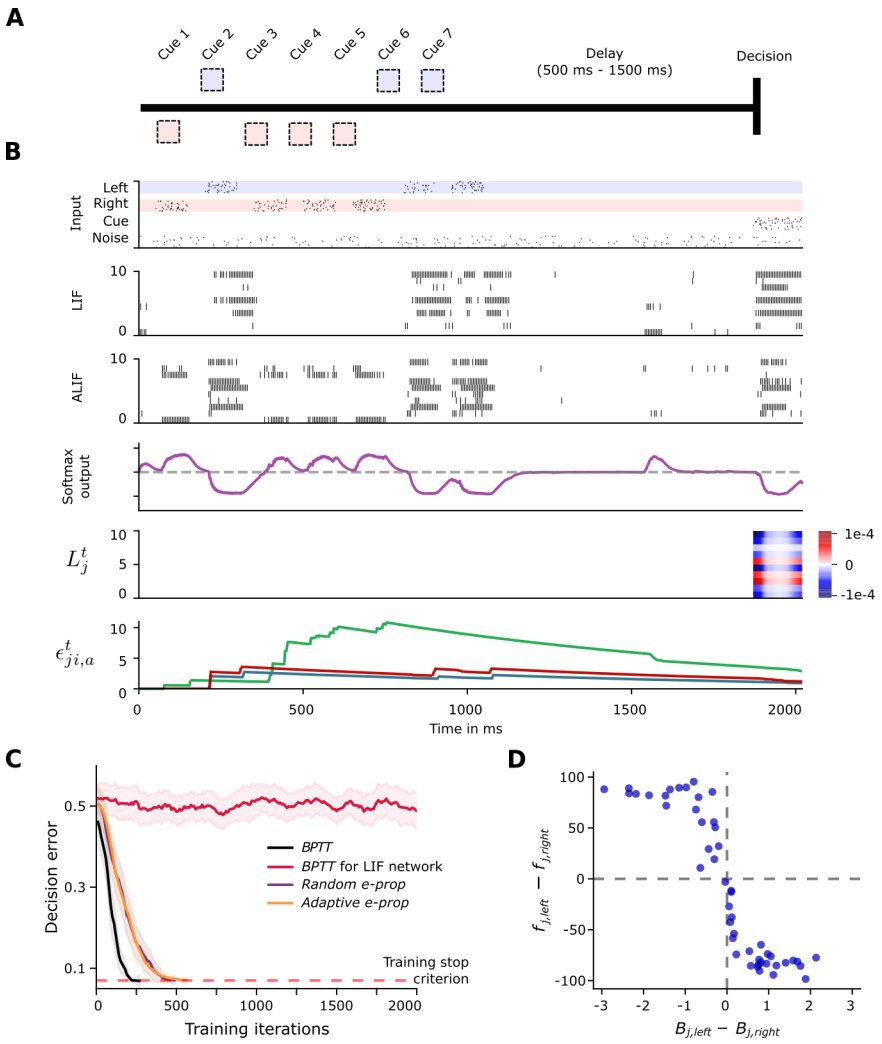

Figure 3: **Solving a task with difficult temporal credit assignment by *e-prop*.** **A**) Setup of corresponding rodent experiments of (*Morcos and Harvey, 2016, Engelhard et al., 2019*), see Movie S2. **B**) Input spikes, internal spiking activity of 10 out of 50 sample LIF neurons and 10 out of 50 sample ALIF neurons, softmax output, sample learning signals and samples of slow components of eligibility traces in the bottom row. **C**) Learning curves for *BPTT* and two *e-prop* versions. **D**) Correlation between the broadcast weights B_{jk} for $k =$ left/right for learning signals in *random e-prop* and sensitivity to "left" and "right" input components after learning. $f_{\mathrm{j,left}}$ ($f_{\mathrm{j,right}}$) is the resulting average firing rate of neuron j during presentation of left (right) cues after learning.

the subject has to learn to count cues separately for each side and to compare the two resulting numbers. Furthermore the cues need to be processed long before a reward is given. We show in Fig. S4 that LSNNs can learn this task through *reward-based e-prop*. But since the LSNNs can alleviate there the temporal credit assignment problem through reward prediction, we wondered whether an LSNN would also be able to learn via *e-prop* a supervised learning variation of this task, where a teacher tells the subject at the end of each trial what would have been the right decision. This yields a really challenging scenario for *e-prop* since non-zero learning signals L_j^t arise only during the last 150ms of a trial (Fig. 3B). Hence all synaptic plasticity of *e-prop* has to take place during these last 150ms, long after the relevant computations on input cues had been carried out. The result of training an LSNN with *BPTT* and *e-prop* for solving this task is shown in Fig. 3C (illustrated in Movies S3 and S4). Whereas this task can not even be solved by *BPTT* with a regular RSNN that has no adapting neurons (red curve), all 3 previously discussed variations of *e-prop* can solve it if the RSNN contains adapting neurons. We also explain in section S2.4 how this task can be solved for sparsely connected LSNNs when biologically inspired stochastic rewiring (*Kappel et al., 2018*) is integrated into *e-prop*.

But how can the neurons in the LSNN learn to record and count the input cues if all the learning signals are identically 0 until the last 150ms (5th row of Fig. 3B)? The solution is indicated in the bottom row of Fig. 3B: The slow component $\epsilon_{ji,a}^t$ (equation (22)) of the eligibility traces e_{ji} of adapting neurons j decays with the long time constant of firing rate adaptation (see equation (27) and Movie S4), that typically lies in the range of seconds. Since these traces stretch from the beginning of the trial into its last phase, they enable assignment of credit to firing events that happened over 1000 ms ago. Fig. 3D provides insight into the functional role of the broadcast weights of *random e-prop* in this context: The difference of these weights determines for each neuron j whether it learns to respond in the first phase of a trial more to cues from the left or right. This observation suggests that neuron-specific learning signals for RSNNs

have the advantage that they can create a variety of feature detectors for task-relevant network inputs. Hence a suitable weighted sum of these feature detectors is able to cancel remaining errors at the network output, similarly as in the case of feedforward networks (*Lillicrap et al., 2016*).

Reward-based e-prop

Deep RL has recently produced really powerful results in machine learning and AI through clever applications of *BPTT* to RL (*Mnih et al., 2016*). We found that one of the arguably most powerful RL methods within the range of deep RL approaches that are not directly biologically implausible, policy gradient in combination with actor-critic, can be implemented with *e-prop*. This yields the biologically plausible RL algorithm *reward-based e-prop*. The LSNN learns through *reward-based e-prop* both an approximation to the value function and a stochastic policy. Neuron-specific learning signals are combined in *reward-based e-prop* with a global signal that transmits reward prediction errors (Fig. S3). In contrast to the supervised case where the learning signals depend on the deviation from an external target signal, the learning signals here are emitted when an action is taken and they express here how much this action deviates from the action mean that is currently proposed by the network. We show in Methods that *reward-based e-prop* yields local reward-based rules for synaptic plasticity that are in many aspects similar to ones that have previously been discussed in the literature (*Gerstner et al., 2018*). But those previously proposed rules estimated gradients of the policy essentially by correlating the noisy output of network neurons with rewards, which is known to be inefficient due to noisy gradient estimates. In contrast, *reward-based e-prop* computes policy- and value-gradients by approximating *BPTT*, which is one of the pillars of modern deep RL.

We tested *reward-based e-prop* on a task that captures the essence of numerous learning experiments in systems neuroscience: A delayed goal-directed movement has to be learnt, con-

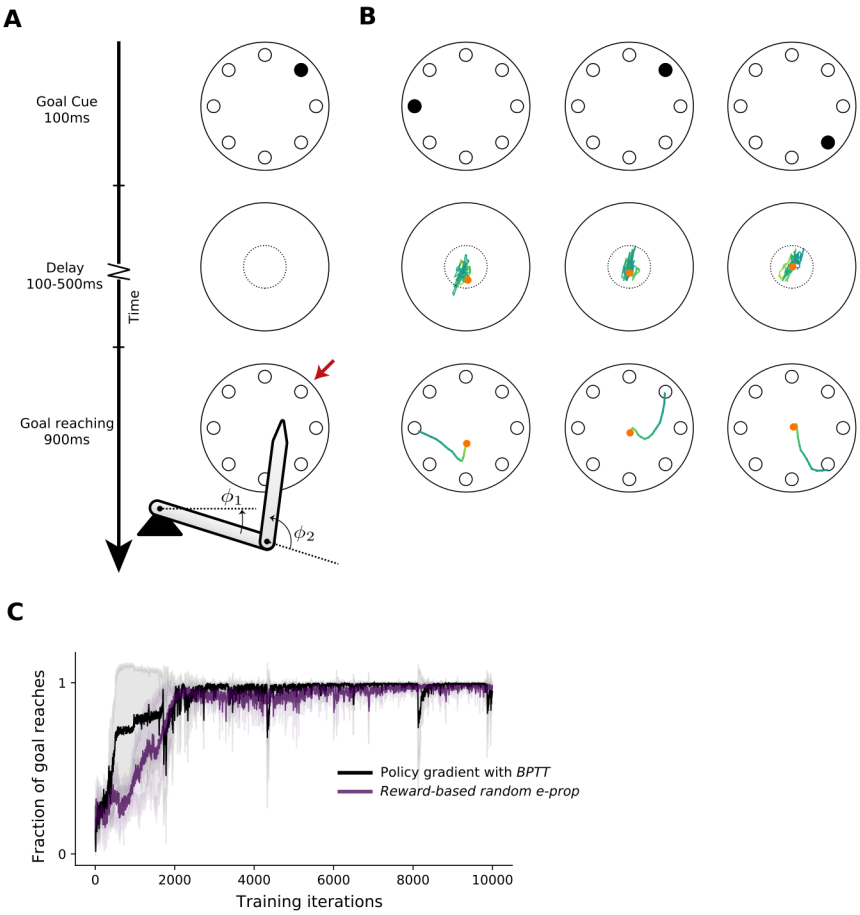

Figure 4: **Application of *e-prop* to RL. A**) Scheme of the delayed arm movement task. The red arrow points to the formerly visible goal. The arm always starts moving from the center of the circle. **B**) Resulting arm movement in three sample trials after learning. The orange dot indicates the position of the tip of the arm at the end of the delay period. **C**) Performance of *reward-based random e-prop* and of a control where *e-prop* is replaced by *BPTT*, both for an LSNN consisting of 350 LIF and 150 ALIF neurons. Solid curves show the mean over 5 different runs, and shaded area indicates 1 standard deviation.

sisting of a sequence of many 2-dimensional continuous motor commands, each of them being only loosely linked to rewards. We chose a setup where the agent first receives a spatial goal cue (Fig. 4A), then has to control the angles of a two-joint arm during a delay so that its tip remains – in spite of motor noise that result from the stochastic policy – within a center region (indicated by a dotted circle) in order to avoid small negative rewards, until it receives a go-cue (see Movie S5). The agent then has to move the tip of the arm to the location of the initial goal cue in order to receive a reward. Note that no forward- or inverse model of the arm was given to the LSNN, it had to learn those implicitly. This task had so far been beyond the reach of biologically plausible learning, for any type of neural network model.

Three sample trials after learning are shown in Fig. 4B (and in Movie S6). Fig. 4C shows that *reward-based e-prop* is able to solve this demanding RL task about as well as policy gradient with biologically implausible *BPTT*. We conjecture that variants of *reward-based e-prop* will be able to solve most RL tasks that can be solved by online actor-critic methods in machine learning.

Discussion

We propose that in order to understand the computational function and neural coding of higher brain areas, one needs to understand the organization of the plasticity mechanisms that install and maintain the computational functions of the underlying RSNNs. So far BPTT was the only candidate for that, since no other learning method provided sufficiently powerful computational function to RSNN models. But since *BPTT* is not viewed to be biologically realistic (*Lillicrap and Santoro, 2019*), it does not help us to understand the organization of synaptic plasticity in RSNNs of the brain. *E-prop* offers a solution to this dilemma, since it does not require biologically unrealistic mechanisms, but still enables RSNNs to learn difficult computational tasks almost as well as *BPTT*. In particular, we have shown in Fig. 3 and 4 that *e-prop* enables

us to model for the first time the learning processes in RSNNs of the brain that underlie the emergence of complex behaviors in key experiments of systems neuroscience.

E-prop relies on two types of signals that are abundandly available in the brain, but whose precise role for learning have not yet been understood: eligibility traces and learning signals. Since *e-prop* is based on a transparent mathematical principle, it provides a normative model for both types of signals, as well as for synaptic plasticity rules. In particular, it suggests a new rule for the organization of eligibility traces: that the time constant of the eligibility trace for a synapse is correlated with the time constant for the history-dependence of the firing activity of the postsynaptic neuron. It also suggests that the experimentally found diverse time constants of the firing activity of populations of neurons in different brain areas (*Runyan et al., 2017*) are correlated with their capability to handle corresponding ranges of delays in temporal credit assignment for learning. Finally, *e-prop* theory suggests that learning signals for different populations of neurons should be diverse, rather than uniform and global (see section S6.2), and should be correlated with the impact which the activity of these neurons has on the quality of the learnt behavior.

Apart from these consequences of *e-prop* for research in neuroscience and cognitive science, *e-prop* also provides an interesting new tool for approaches in machine learning where *BPTT* is replaced by approximations in order to improve computational efficiency. For example, the combination of eligibility traces from *e-prop* with synthetic gradients from (*Jaderberg et al., 2016*) substantially improves performance of LSTM networks for difficult machine learning problems such as the copy-repeat task and the Penn Treebank word prediction task (*Bellec et al., 2019*).

Finally, *E-prop* suggests a viable new approach for on-chip learning of RSNNs on neuromorphic chips. Whereas *BPTT* is not within the reach of current neuromorphic chip designs, an implementation of *e-prop* appears to offer no serious hurdle. Since we have shown in Fig. 2

that *e-prop* enables RSNNs to learn to understand speech, and in Fig. 4 that *e-prop* enables reward-based learning of the control of complex arm movements, *e-prop* promises to support a qualitative jump in on-chip learning capabilities of neuromorphic chips.

Methods

To exhibit the theory around *e-prop* and preceding related work, we structure the methods section in the following way:

- Comparison of *e-prop* with other online learning methods for recurrent neural networks (RNNs)
- Network models
- Conventions
- Mathematical basis for *e-prop*
- Eligibility traces
- Eligibility traces for concrete neuron models
- Derivation of the synaptic plasticity rules resulting from *e-prop*
- *Reward-based e-prop*: application of *e-prop* to policy gradient RL.

Comparison of *e-prop* with other online learning methods for recurrent neural networks (RNNs)

In this section we compare *e-prop* with other learning algorithms implementing gradient descent in RNNs without BPTT. A well-known alternative to *BPTT* is real time recurrent learning

(RTRL). RTRL was derived for networks of rate-based (sigmoidal) neurons in (*Williams and Zipser, 1989*). There, the loss gradients are computed forward in time by multiplying the full Jacobian $\mathbf{J}^t_{kk'} = \frac{d\mathbf{h}^t_k}{d\mathbf{h}^{t-1}_{k'}}$ of the network dynamics with the tensor $\frac{d\mathbf{h}^t_k}{dW_{ji}}$ that computes the dependency of the state variables with respect to the parameters: $\frac{d\mathbf{h}^t_k}{dW_{ji}} = \sum_{k'} \mathbf{J}^t_{kk'} \cdot \frac{d\mathbf{h}^{t-1}_{k'}}{dW_{ji}} + \frac{\partial \mathbf{h}^t_k}{\partial W_{ji}}$ (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. Unbiased Online Recurrent Optimization (*Tallec and Ollivier, 2018*) (UORO) used an unbiased estimator of $\mathbf{J}^t_{kk'}$ of rank one that can be computed online. The authors report that the variance of this estimator increases with the network size and simulations were only carried out for a network size up to 64. Another unbiased estimator of $\mathbf{J}^t_{kk'}$ (*Mujika et al., 2018*) based on Kronecker factors solved this issue and made it possible to approach the performance of *BPTT* on harder tasks. Yet this method requires $O(n^3)$ operations per time step, which is one order more than UORO, *e-prop* or *BPTT*.

In *e-prop*, the eligibility traces are just $d \times d$ matrices (d being the dimension of $\mathbf{h}^t_j$), since they are restrictions of the full Jacobian $\mathbf{J}^t_{kk'}$ to the internal dynamics of a neuron ($k = k'$). As a consequence, only $O(n^2)$ multiplications are required for the forward propagation of eligibility traces. Hence their computation is not more costly than *BPTT* or the simulation of the RNN.

The learning rule called Superspike (*Zenke and Ganguli, 2018*) was derived by applying RTRL in spiking neural networks without recurrent connections. In the absence of these connections RTRL is practicable and the resulting learning rule uses eligibility traces similar to those arising in *e-prop* with LIF neurons. Two other algorithms, (*Roth et al., 2019*) and (*Murray, 2019*), were introduced to train recurrent neural networks of sigmoidal units by approximating RTRL with another form of eligibility traces. Random Feedback Local Online (RFLO) learning (*Murray, 2019*) is equivalent to *random e-prop* in the particular case of leaky sigmoidal neurons for regression tasks. But the performance of RFLO was not compared to *BPTT* on

published benchmarks for RNNs, or for spiking neurons. In contrast to the eligibility traces in *e-prop*, the eligibility traces in kernel RNN learning (keRNL) (*Roth et al., 2019*) are viewed as components of an estimator of the tensor $\mathbf{J}^t_{kk'}$, and are not related to the specific definition of the neuron model. This approach requires non-local communication within the RNN, which we wanted to avoid in *e-prop*. In contrast to *e-prop*, none of the papers above (*Zenke and Ganguli, 2018, Murray, 2019, Roth et al., 2019*) derived a theory or a definition of eligibility traces that can be applied to neuron models with a non-trivial internal dynamics, such as adaptive neurons or LSTM units, that appear to be essential for solving tasks with demanding temporal credit assignment of errors.

Network models

To exhibit the generality of the *e-prop* approach, we define the dynamics of recurrent neural networks using a general formalism that is applicable to many recurrent neural network models, not only to RSNNs and LSNNs. Also non-spiking models such as LSTM networks fit under this formalism (see Section S4.3 in the Supplement). The network dynamics is summarized by the computational graph in Fig. 5. It uses the function M to define the update of the hidden state: $\mathbf{h}^t_j = M(\mathbf{h}^{t-1}_j, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{W})$, and f to define the update of the observable state: $z^t_j = f(\mathbf{h}^t_j, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{W})$ (f simplifies to $z^t_j = f(\mathbf{h}^t_j)$ for LIF and ALIF neurons).

RSNNs. RSNNs are recurrently connected networks of leaky integrate-and-fire (LIF) neurons. Each LIF neuron has a one dimensional internal state h^t_j that consists only of the membrane potential v^t_j. The observable state $z^t_j \in \{0, 1\}$ is binary, indicating a spike ($z^t_j = 1$) or no

spike ($z_j^t = 0$) at time t. The dynamics of the LIF model is defined by the equations:

$$v_j^{t+1} = \alpha v_j^t + \sum_{i \neq j} W_{ji}^{\text{rec}} z_i^t + \sum_i W_{ji}^{\text{in}} x_i^{t+1} - z_j^t v_{\text{th}} \tag{3}$$

$$z_j^t = H\left(v_j^t - v_{\text{th}}\right), \tag{4}$$

where $x_i^t = 1$ indicates a spike from the input neuron i at time step t ($x_i^t = 0$ otherwise) and W_{ji}^{rec} (W_{ji}^{in}) is the synaptic weight from network (input) neuron i to neuron j. The decay factor α in (3) is given by $e^{-\delta t/\tau_m}$, where δt is the discrete time step size (1 ms in our simulations) and $\tau_m = 20$ ms is the membrane time constant. H denotes the Heaviside step function.

Due to the term $-z_j^t v_{\text{th}}$ in equation (3), the neurons membrane potential is reduced by a constant value after an output spike, which relates our model to the spike response model (*Gerstner et al., 2014*). To introduce a simple model of neuronal refractoriness, we further assume that z_j^t is fixed to 0 after each spike of neuron j for a short refractory period of 2 to 5ms depending on the simulation.

LSNNs. LSNNs are recurrently connected networks that consist of LIF neurons and of adaptive LIF (ALIF) neurons. An ALIF neuron has a time-dependent threshold adaptation a_j^t. As a result, their internal state is a 2 dimensional vector $\mathbf{h}_j^t \overset{\text{def}}{=} [v_j^t, a_j^t]$. Their threshold potential A_j^t increases with every output spike and decreases exponentially back to the baseline threshold v_{th}. This can be described by

$$A_j^t = v_{\text{th}} + \beta a_j^t \,, \tag{5}$$

$$z_j^t = H(v_j^t - A_j^t) \,, \tag{6}$$

with a threshold adaptation according to

$$a_j^{t+1} = \rho a_j^t + z_j^t \,, \tag{7}$$

where the decay factor ρ is given by $e^{-\delta t/\tau_a}$, and τ_a is the adaptation time constant that is typically chosen to be in the range of the time span of the length of the working memory that is a relevant for a given task. This is a very simple model for a neuron with spike frequency adaptation. We refer to (*Gerstner et al., 2014, Pozzorini et al., 2015, Gouwens et al., 2018*) for experimental data and other neuron models.

In relation to the more general formalism represented in the computational graph in Fig. 5, equations (3) and (7) define $M(\mathbf{h}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{W})$, and equations (4) and (6) define $f(\mathbf{h}_j^t)$.

Gradient descent for RSNNs. Gradient descent is problematic for spiking neurons because of the step function H in equation (4). We overcome this issue as in (*Esser et al., 2016, Bellec et al., 2018*): the non-existing derivative $\frac{\partial z_j^t}{\partial v_j^t}$ is replaced in simulations by a simple nonlinear function of the membrane potential that is called the pseudo-derivative. Outside of the refractory period, we choose a pseudo-derivative of the form $\psi_j^t = \frac{1}{v_{\text{th}}}\gamma_{\text{pd}} \max\left(0, 1 - \left|\frac{v_j^t - A_j^t}{v_{\text{th}}}\right|\right)$ where $\gamma_{\text{pd}} = 0.3$. During the refractory period the pseudo derivative is set to 0.

Network output and loss functions. We assume that network outputs y_k^t are real-valued and produced by leaky output neurons (readouts), which are not recurrently connected:

$$y_k^t = \kappa y_k^{t-1} + \sum_j W_{kj}^{\text{out}} z_j^t + b_k^{\text{out}}, \tag{8}$$

where $\kappa \in [0, 1]$ defines the leak and b_k^{out} denotes the output bias. The leak factor κ is given for spiking neurons by $e^{-\delta t/\tau_{out}}$, where τ_{out} is the membrane time constant. Note that for non-spiking neural networks (such as for LSTM networks), temporal smoothing of the network observable state is not necessary. In this case, one can use $\kappa = 0$.

The loss function E quantifies the network performance. We assume that it depends only on the observable states $E(\mathbf{z}^1, \ldots, \mathbf{z}^T)$. For instance, for a regression problem we define E as

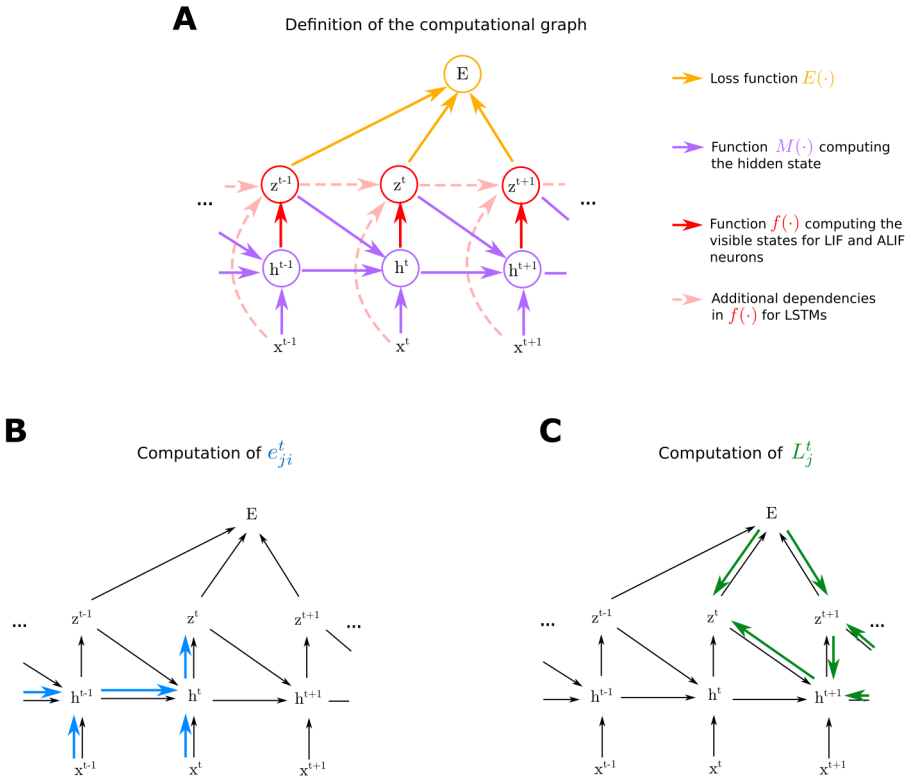

Figure 5: **Computational graph and gradient propagations A)** Assumed mathematical dependencies between hidden neuron states $\mathbf{h}_j^t$, neuron outputs $\mathbf{z}^t$, network inputs $\mathbf{x}^t$, and the loss function E through the mathematical functions $E(\cdot)$, $M(\cdot)$, $f(\cdot)$ are represented by coloured arrows. **B-C)** The gradient computation can be represented in similar graphs, where coloured arrows represent partial derivatives. **B)** Following equation (19), the derivatives involved in the computation of eligibility traces e_{ji}^t are shown in blue in the case where i is an input neuron. **C)** Unlike the eligibility traces, the ideal learning signals required to back-propagate gradients as represented here with green arrows.

the mean square error $E = \frac{1}{2}\sum_{t,k}(y_k^t - y_k^{*,t})^2$ between the network outputs y_k^t and target values $y_k^{*,t}$. For classification or RL tasks the loss function E has to be re-defined accordingly.

Conventions

Notation for derivatives. We distinguish the total derivative $\frac{dE}{d\mathbf{z}^t}(\mathbf{z}^1, \ldots, \mathbf{z}^T)$, which takes into account how E depends on $\mathbf{z}_t$ also indirectly through influence of $\mathbf{z}^t$ on the other variables $\mathbf{z}^{t+1}, \ldots, \mathbf{z}^T$, and the partial derivative $\frac{\partial E}{\partial \mathbf{z}^t}(\mathbf{z}^1, \ldots, \mathbf{z}^T)$ which quantifies only the direct dependence of E on $\mathbf{z}^t$.

Analogously $\frac{\partial M}{\partial h}$ denotes for $\mathbf{h}_j^t = M(\mathbf{h}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{W})$, the partial derivative of M with respect to $\mathbf{h}$. It only quantifies the direct influence of $\mathbf{h}_j^t$ on $\mathbf{h}_j^{t-1}$ and it does not take into account the dependency of $\mathbf{h}_j^t$ on $\mathbf{h}_j^{t-1}$ via the observable states $\mathbf{z}^t$. To improve readability we also use the following abbreviations: $\frac{\partial \mathbf{h}_j^t}{\partial \mathbf{h}_j^{t-1}} \stackrel{\text{def}}{=} \frac{\partial M}{\partial h}(\mathbf{h}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{W})$, $\frac{\partial \mathbf{h}_j^t}{\partial W_{ji}} \stackrel{\text{def}}{=} \frac{\partial M}{\partial W_{ji}}(\mathbf{h}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{W})$, and $\frac{\partial z_j^t}{\partial \mathbf{h}_j^t} \stackrel{\text{def}}{=} \frac{\partial f}{\partial h}(\mathbf{h}_j^t, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{W})$.

Notation for temporal filters. For ease of notation we use the operator $\mathcal{F}_\alpha$ to denote the low-pass filter such that, for any time series x_t:

$$\mathcal{F}_\alpha(x^t) = \alpha \mathcal{F}_\alpha(x^{t-1}) + x^t \,, \tag{9}$$

and $\mathcal{F}_\alpha(x^0) = x^0$. In the specific case of the time series z_j^t and e_{ji}^t, we simplify notation further and write $\bar{z}_j^t$ and $\bar{e}_{ji}^t$ for $\mathcal{F}_\alpha(z_j)^t$ and $\mathcal{F}_\kappa(e_{ji})^t$

Mathematical basis for *e-prop*

We provide here the proof of the fundamental equation (1) for *e-prop*

$$\frac{dE}{dW_{ji}} = \sum_t \frac{dE}{dz_j^t}\, e_{ji}^t \,. \tag{10}$$

This equation shows that the total derivative of the loss function E with respect to the synaptic weights $\boldsymbol{W}$ can be written as a product of learning signals L_j^t and eligibility traces e_{ji}^t for the "ideal" learning signal $L_j^t = \frac{dE}{dz_j}$. The eligibility traces are defined at the end of the proof below.

We start from a factorization of the loss gradient that arises in equation (12) of (*Werbos, 1990*) to describe *BPTT* in recurrent sigmoidal neural networks. Using our notation, this classical factorization of loss gradient can be rewritten as:

$$\frac{dE}{dW_{ji}} = \sum_{t'} \frac{dE}{d\mathbf{h}_j^{t'}} \cdot \frac{\partial \mathbf{h}_j^{t'}}{\partial W_{ji}} \,. \tag{11}$$

We now show how one can derive from this to the new factorization (10) of the loss gradient that underlies *e-prop*. $\frac{dE}{d\mathbf{h}_j^{t'}}$ can be expressed recursively as a function of the same derivative at the next time step $\frac{dE}{d\mathbf{h}_j^{t'+1}}$ by applying the chain rule at the node $\mathbf{h}_j^t$ for $t = t'$ of the computational graph shown in Figure 5C:

$$\begin{aligned} \frac{dE}{d\mathbf{h}_j^{t'}} &= \frac{dE}{dz_j^{t'}} \frac{\partial z_j^{t'}}{\partial \mathbf{h}_j^{t'}} + \frac{dE}{d\mathbf{h}_j^{t'+1}} \frac{\partial \mathbf{h}_j^{t'+1}}{\partial \mathbf{h}_j^{t'}} & (12) \\ &= L_j^{t'} \frac{\partial z_j^{t'}}{\partial \mathbf{h}_j^{t'}} + \frac{dE}{d\mathbf{h}_j^{t'+1}} \frac{\partial \boldsymbol{h}_j^{t'+1}}{\partial \boldsymbol{h}_j^{t'}}, & (13) \end{aligned}$$

where we defined the learning signal $L_j^{t'}$ as $\frac{dE}{dz_j^{t'}}$. The resulting recursive expansion ends at the last time step T of the computation of the RNN, i.e., $\frac{dE}{d\mathbf{h}_j^{T+1}} = 0$. If one substitutes the recursive formula (13) into the definition of the loss gradients (11), one gets:

$$\begin{aligned} \frac{dE}{dW_{ji}} &= \sum_{t'} \left(L_j^{t'} \frac{\partial z_j^{t'}}{\partial \mathbf{h}_j^{t'}} + \frac{dE}{d\mathbf{h}_j^{t'+1}} \frac{\partial \boldsymbol{h}_j^{t'+1}}{\partial \boldsymbol{h}_j^{t'}} \right) \cdot \frac{\partial \mathbf{h}_j^{t'}}{\partial W_{ji}} & (14) \\ &= \sum_{t'} \left(L_j^{t'} \frac{\partial z_j^{t'}}{\partial \mathbf{h}_j^{t'}} + \left(L_j^{t'+1} \frac{\partial z_j^{t'+1}}{\partial \mathbf{h}_j^{t'+1}} + (\cdots) \frac{\partial \boldsymbol{h}_j^{t'+2}}{\partial \boldsymbol{h}_j^{t'+1}} \right) \frac{\partial \boldsymbol{h}_j^{t'+1}}{\partial \boldsymbol{h}_j^{t'}} \right) \cdot \frac{\partial \mathbf{h}_j^{t'}}{\partial W_{ji}} \,. & (15) \end{aligned}$$

The following equation is the main equation for understanding the transformation from *BPTT* into *e-prop*. The key idea is to collect all terms which are multiplied with the learning signal L_j^t at a given time t. These are only terms that concern events in the computation of neuron j

up to time t, and they do not depend on other future losses or variable values. We collect them into an eligibility trace e_{ji}^t for each neuron j and i, which can be computed locally 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 L_j^t. This expresses the loss gradient of E as a sum of learning signals L_j^t multiplied by some factor indexed by ji, which we define as the eligibility trace $e_{ji}^t \in \mathbb{R}$ and eligibility vectors $\boldsymbol{\epsilon}_{ji}^t \in \mathbb{R}^d$, which have the same dimension as the hidden states $\mathbf{h}_{ji}^t$

$$\frac{dE}{dW_{ji}} = \sum_{t'} \sum_{t \geq t'} L_j^t \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} \frac{\partial \boldsymbol{h}_j^t}{\partial \boldsymbol{h}_j^{t-1}} \cdots \frac{\partial \boldsymbol{h}_j^{t'+1}}{\partial \boldsymbol{h}_j^{t'}} \cdot \frac{\partial \mathbf{h}_j^{t'}}{\partial W_{ji}} \tag{16}$$

$$= \sum_t L_j^t \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} \underbrace{\sum_{t' \leq t} \frac{\partial \boldsymbol{h}_j^t}{\partial \boldsymbol{h}_j^{t-1}} \cdots \frac{\partial \boldsymbol{h}_j^{t'+1}}{\partial \boldsymbol{h}_j^{t'}} \cdot \frac{\partial \mathbf{h}_j^{t'}}{\partial W_{ji}}}_{\stackrel{\text{def}}{=} \boldsymbol{\epsilon}_{ji}^t} . \tag{17}$$

Here, we use the identity matrix for $\frac{\partial \boldsymbol{h}_j^t}{\partial \boldsymbol{h}_j^{t-1}} \cdots \frac{\partial \boldsymbol{h}_j^{t'+1}}{\partial \boldsymbol{h}_j^{t'}}$ if $t = t'$. After defining the eligibility vector $\boldsymbol{\epsilon}_{ji}^t$, we also define

$$e_{ji}^t \stackrel{\text{def}}{=} \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} \cdot \boldsymbol{\epsilon}_{ji}^t , \tag{18}$$

so that equation (17) proves the factorization of *e-prop* in (1).

Eligibility traces

Online computation of eligibility traces. The eligibility vectors as defined in (17) can be computed recursively for efficiency and in order to avoid the back-propagation of signals through time:

$$\boldsymbol{\epsilon}_{ji}^t = \frac{\partial \boldsymbol{h}_j^t}{\partial \boldsymbol{h}_j^{t-1}} \cdot \boldsymbol{\epsilon}_{ji}^{t-1} + \frac{\partial \mathbf{h}_j^t}{\partial W_{ji}} , \tag{19}$$

where $\cdot$ denotes the dot product. The eligibility traces can be computed with their definition in equation (18).

Derivation of eligibility traces for concrete neuron models

The eligibility traces for LSTMs are provided in the supplementary materials. Below we provide the derivation of eligibility traces for spiking neurons.

Eligibility traces for LIF neurons. We compute the eligibility trace of a LIF neuron without adaptive threshold (equation (3)). Here the hidden state $\mathbf{h}_j^t$ consists just of the membrane potential v_j^t and we have $\frac{\partial \boldsymbol{h}_j^{t+1}}{\partial \boldsymbol{h}_j^t} = \frac{\partial v_j^{t+1}}{\partial v_j^t} = \alpha$ and $\frac{\partial v_j^t}{\partial W_{ji}} = z_i^{t-1}$ (for a derivation of the eligibility traces taking the reset into account we refer to section S1.2). Using these derivatives and equation (19), one obtains that the eligibility vector is the low-pass filtered pre-synaptic spike-train,

$$\boldsymbol{\epsilon}_{ji}^{t+1} = \mathcal{F}_\alpha(z_i^t) \stackrel{\text{def}}{=} \bar{z}_i^t \,. \tag{20}$$

and following equation (18), the eligibility trace is:

$$e_{ji}^{t+1} = \psi_j^{t+1} \bar{z}_i^t \,. \tag{21}$$

For LIF neurons as well as for ALIF neurons in the following section the derivation applies to the input connections by substituting the network spikes z_i^{t-1} by the input spikes x_i^t (the time index switches from $t-1$ to t because the hidden state $\mathbf{h}_j^t = M(\mathbf{h}_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{W})$ is defined as a function of the input at time t but the preceding recurrent activity). For simplicity we have focused on the case where transmission delays between neurons in the RSNN are just 1ms. If one uses more realistic length of delays d, this $-d$ appears in equations (21)–(23) instead of -1 as the most relevant time point for pre-synaptic firing (see Section S1.3). This moves resulting synaptic plasticity rules closer to experimentally observed forms of STDP.

Eligibility traces for ALIF neurons. The hidden state of an ALIF neuron $\mathbf{h}_j^t = [v_j^t, a_j^t]$ is a two dimensional vector to capture the state of the adaptive threshold a_j^t besides the membrane potential v_j^t. Hence a two dimensional eligibility vector $\boldsymbol{\epsilon}_{ji}^t \stackrel{\text{def}}{=} [\epsilon_{ji,v}^t, \epsilon_{ji,a}^t]$ is associated with

each weight, and the matrix $\frac{\partial \boldsymbol{h}_j^{t+1}}{\partial \boldsymbol{h}_j^t}$ is a 2×2 matrix. The derivatives $\frac{\partial a_j^{t+1}}{\partial a_j^t}$ and $\frac{\partial a_j^{t+1}}{\partial v_j^t}$ capture the dynamics of the adaptive threshold. Hence to derive the computation of eligibility traces we substitute the spike z_j in equation (7) by its definition given in equation (6). With this convention one finds that the diagonal of the matrix $\frac{\partial \boldsymbol{h}_j^{t+1}}{\partial \boldsymbol{h}_j^t}$ is formed by the terms $\frac{\partial v_j^{t+1}}{\partial v_j^t} = \alpha$ and $\frac{\partial a_j^{t+1}}{\partial a_j^t} = \rho - \psi_j^t \beta$. Above and below the diagonal, one finds respectively $\frac{\partial v_j^{t+1}}{\partial a_j^t} = 0$, $\frac{\partial a_j^{t+1}}{\partial v_j^t} = \psi_j^t$. One can finally compute the eligibility traces using equation (18). The component of the eligibility vector associated with the membrane potential remains the same as in the LIF case and only depends on the presynaptic neuron: $\epsilon_{ji,v}^t = \bar{z}_i^{t-1}$. For the component associated with the adaptive threshold we find the following recursive update:

$$\epsilon_{ji,a}^{t+1} \quad = \quad \psi_j^t \bar{z}_i^{t-1} + (\rho - \psi_j^t \beta) \epsilon_{ji,a}^t \, , \tag{22}$$

and this results in an eligibility trace of the form:

$$e_{ji}^t \quad = \quad \psi_j^t \left(\bar{z}_i^{t-1} - \beta \epsilon_{ji,a}^t \right). \tag{23}$$

Recall that the constant $\rho = \exp(-\frac{\delta t}{\tau_a})$ arises from the adaptation time constant τ_a, which typically lies in the range of hundreds of milliseconds to a few seconds in our experiments, yielding values of ρ between 0.995 and 0.9995. The constant β is typically of the order of 0.07 in our experiments.

To provide a more interpretable form of eligibility trace that fits into the standard form of local terms considered in 3-factor learning rules (*Gerstner et al., 2018*), one may drop the term $-\psi_j^t \beta$ in equation (22). This approximation $\widehat{\epsilon}_{ji,a}^t$ of equation (22) becomes an exponential trace of the post-pre pairings accumulated within a time window as large as the adaptation adaptation time constant:

$$\widehat{\epsilon}_{ji,a}^{t+1} \quad = \quad \mathcal{F}_\rho \left(\psi_j^t \bar{z}_i^{t-1} \right) \, . \tag{24}$$

The eligibility traces are computed with equation (22) in most experiments but the performance obtained with *symmetric e-prop* and this simplification were indistinguishable on the evidence accumulation task of Fig. 3.

Synaptic plasticity rules resulting from *e-prop*

An exact computation of the ideal learning signal $\frac{dE}{dz_j^t}$ in equation (1) requires to back-propagate gradients through time (see Fig. 5C). To compute the loss gradients with *e-prop* we replace it with the partial derivative $\frac{\partial E}{\partial z_j^t}$ which can be computed online. Implementing the weight updates with gradient descent and learning rate η, all the following plasticity rules are derived from the formula

$$\Delta W_{ji}^{\text{rec}} = -\eta \sum_t \frac{\partial E}{\partial z_j^t} e_{ji}^t \,. \tag{25}$$

Note that the weight updates derived for the recurrent weights W_{ji}^{rec} also applies to the inputs weights W_{ji}^{in}. For the output weights and biases the derivation does not rely on the theory of *e-prop*, and the weight updates can be found in the Section S3.1.

Case of regression tasks. In the case of a regression problem with targets $y_k^{*,t}$ and outputs y_k^t defined in equation (8), we define the loss function $E = \frac{1}{2}\sum_{t,k}(y_k^t - y_k^{*,t})^2$ which results in a partial derivative of the form $\frac{\partial E}{\partial z_j^t} = \sum_k W_{kj}^{\text{out}} \sum_{t' \geq t}(y_k^{t'} - y_k^{*,t'})\kappa^{t'-t}$. This seemingly provides an obstacle for online learning, because the partial derivative is a weighted sum over future errors. But this problem can be resolved as one interchange two sum indices in the expression of the weight updates (see section S3.1). It results that the sum over future events transforms into a low-pass filtering of the eligibility traces $\bar{e}_{ji}^t = \mathcal{F}_\kappa(e_{ji}^t)$, and the resulting weight update can be written as

$$\Delta W_{ji}^{\text{rec}} = -\eta \sum_t \underbrace{\Big(\sum_k B_{jk}(y_k^t - y_k^{*,t})\Big)}_{=L_j^t} \bar{e}_{ji}^t \,. \tag{26}$$

Here, B_{jk} denote broadcast weights in analogy to (*Lillicrap et al., 2016*), where we note that $B_{jk} = W^{\text{out}}_{kj}$ as the ideal values.

Case of classification tasks. We assume that K target categories are provided in the form of a one-hot encoded vector $\boldsymbol{\pi}^{*,t}$ with K dimensions. We define the probability for class k predicted by the network as $\pi^t_k = \text{softmax}_k(y^t_1, \ldots, y^t_K) = \exp(y^t_k)/\sum_{k'} \exp(y^t_{k'})$, and the loss function for classification tasks as the cross-entropy error $E = -\sum_{t,k} \pi^{*,t}_k \log \pi^t_k$. The plasticity rule resulting from *e-prop* reads (see derivation in Section S3.1):

$$\Delta W^{\text{rec}}_{ji} = -\eta \sum_t \underbrace{\Big(\sum_k B_{jk}(\pi^t_k - \pi^{*,t}_k)\Big)}_{=L^t_j} \bar{e}^t_{ji} \,. \tag{27}$$

Reward-based e-prop: application of *e-prop* to policy gradient RL

For reinforcement learning, the network interacts with an external environment. Based on the observations $\boldsymbol{x}^t$ that are perceived, the network has to commit to actions $\boldsymbol{a}^{t_0}, \ldots, \boldsymbol{a}^{t_n}, \ldots$ at certain decision times $t_0, \ldots, t_n, \ldots$. Each action $\boldsymbol{a}^{t_n}$ is sampled from a probability distribution $\pi(\boldsymbol{a}^{t_n}; \boldsymbol{y}^{t_n})$ which is also referred to as the policy of the RL agent. The policy is defined as function of the network output $\boldsymbol{y}^{t_n}$, and is chosen here to be a vector of Gaussians with means $\boldsymbol{y}^t$ and variance σ^2 (see section S5.1 for discrete actions). At any time t the environment can provide a positive or negative reward r^t.

The goal of reinforcement learning is to maximize the expected sum of discounted future rewards (also called a return): $R^t = \sum_{t' \geq t} \gamma^{t'-t} r^t$, where $\gamma \leq 1$ is a discount factor. That is, we want to maximize $\mathbb{E}[R^t]$, where the expectation is taken over the agent actions $\boldsymbol{a}^t$ and all stochastic variables of the agent and the environment. We approach this optimization problem using the theory of the actor-critic variant of policy gradient algorithms (*Sutton and Barto, 2018*). It involves the policy π (the actor) and an additional output neuron V^t which predict the

value function $\mathbb{E}[R^t]$ (the critic). The loss function of this algorithm is defined as

$$E = E_\pi + C_V E_V \ , \tag{28}$$

where $E_\pi = -\sum_n R^{t_n} \log \pi(\boldsymbol{a}^{t_n}; \boldsymbol{y}^{t_n})$ measures the performance of the stochastic policy π, and $E_V = \sum_t \frac{1}{2}(R^t - V^t)^2$ measures the accuracy of V^t. Unlike in the supervised learning case, we do not derive the weight update using the derivative $\frac{\partial E_\pi}{\partial z_j^t}$ as in equation (25), because it is known to have a high variance in this setting. Instead, we replace it with the estimator $\widehat{\frac{\partial E}{\partial z_j^t}}$ which has the same value in expectation but a lower variance, as in (*Mnih et al., 2016*):

$$\widehat{\frac{\partial E}{\partial z_j^t}} = -\sum_n (R^{t_n} - V^{t_n}) \frac{\partial \log \pi(\boldsymbol{a}^{t_n}; \boldsymbol{y}^{t_n})}{\partial z_j^t} + C_V \frac{\partial E_V}{\partial z_j^t} \ . \tag{29}$$

We describe below the resulting synaptic plasticity rule in the case of multiple continuous actions as needed to solve the task of Fig. 4. For the case of a single discrete actions as used in Fig. S4 we refer to section S5.1.

Case of continuous actions. This task is more difficult when there is a delay between the action and the reward or, even harder, when a sequence of many actions lead together to a delayed reward. There the loss function E cannot be computed online because the evaluation of R^{t_n} requires knowledge of future rewards. To overcome this, we introduce temporal difference errors $\delta^t = r^t + \gamma V^{t+1} - V^t$ (see Fig. S3), and use the equivalence between the forward and backward view in reinforcement learning (*Sutton and Barto, 2018*) to arrive at the following synaptic plasticity rules for a general actor-critic algorithm with *e-prop* (see Section S5.1):

$$\Delta W_{ji}^{\text{rec}} = -\eta \sum_t \delta^t \mathcal{F}_\gamma\Big(L_j^t \, \bar{e}_{ji}^t\Big) \quad \text{for} \tag{30}$$

$$L_j^t = -C_V B_j^V + \sum_k B_{jk}^a \frac{y_k^t - a_k^t}{\sigma^2} \ , \tag{31}$$

where we define the term $y_k^t - a_k^t$ to have value zero when no action is taken at time t. The combination of reward prediction error and neuron-specific learning signal was also used in

a plasticity rule for feedforward networks inspired by neuroscience (*Roelfsema and Holtmaat, 2018*), here it arises from the approximation of *BPTT* by *e-prop* in RSNNs solving RL problems. Note that the filtering $\mathcal{F}_\gamma$ requires an additional eligibility trace per synapse. This arises from the temporal difference learning in RL (*Sutton and Barto, 2018*). It depends on the learning signal and does not have the same function as the eligibility trace e_{ji}^t.

References

Allen Institute: Cell Types Database, 2018. Allen Institute: Cell Types Database (2018). © 2018 Allen Institute for Brain Science. Allen Cell Types Database, cell feature search. Available from: celltypes.brain-map.org/data.

Bellec et al., 2018. 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 *NeurIPS 32*.

Bellec et al., 2019. Bellec, G., Scherr, F., Hajek, E., Salaj, D., Legenstein, R., and Maass, W. (2019). Biologically inspired alternatives to backpropagation through time for learning in recurrent neural nets. *arXiv:1901.09049 [cs]*. arXiv: 1901.09049.

Cassenaer and Laurent, 2012. Cassenaer, S. and Laurent, G. (2012). Conditional modulation of spike-timing-dependent plasticity for olfactory learning. *Nature*, 482(7383):47.

Clopath et al., 2010. 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):344–52.

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

576 Engelhard et al., 2019. Engelhard, B., Finkelstein, J., Cox, J., Fleming, W., Jang, H. J., Ornelas,
577 S., Koay, S. A., Thiberge, S. Y., Daw, N. D., Tank, D. W., et al. (2019). Specialized coding of
578 sensory, motor and cognitive variables in vta dopamine neurons. *Nature*, page 1.

579 Esser et al., 2016. Esser, S. K., Merolla, P. A., Arthur, J. V., Cassidy, A. S., Appuswamy, R.,
580 Andreopoulos, A., Berg, D. J., McKinstry, J. L., Melano, T., Barch, D. R., di Nolfo, C., Datta,
581 P., Amir, A., Taba, B., Flickner, M. D., and Modha, D. S. (2016). Convolutional networks for
582 fast, energy-efficient neuromorphic computing. *PNAS*, 113(41):11441–11446.

583 Garofolo et al., 1993. Garofolo, J. S., Lamel, L. F., Fisher, W. M., Fiscus, J. G., and Pallett,
584 D. S. (1993). DARPA TIMIT acoustic-phonetic continous speech corpus CD-ROM. NIST
585 speech disc 1-1.1. *NASA STI/Recon Technical Report N*, 93.

586 Gerstner et al., 2014. Gerstner, W., Kistler, W. M., Naud, R., and Paninski, L. (2014). *Neuronal*
587 *dynamics: From single neurons to networks and models of cognition*. Cambridge University
588 Press.

589 Gerstner et al., 2018. Gerstner, W., Lehmann, M., Liakoni, V., Corneil, D., and Brea, J. (2018).
590 Eligibility Traces and Plasticity on Behavioral Time Scales: Experimental Support of Neo-
591 Hebbian Three-Factor Learning Rules. *Frontiers in Neural Circuits*, 12.

592 Gouwens et al., 2018. Gouwens, N. W., Berg, J., Feng, D., Sorensen, S. A., Zeng, H., Hawry-
593 lycz, M. J., Koch, C., and Arkhipov, A. (2018). Systematic generation of biophysically de-
594 tailed models for diverse cortical neuron types. *Nature communications*, 9(1):710.

Graves et al., 2013. Graves, A., Mohamed, A.-R., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *ICASSP*, pages 6645–6649.

Greff et al., 2017. Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2017). Lstm: A search space odyssey. *IEEE TNNLS*, 28(10):2222–2232.

Huh and Sejnowski, 2018. Huh, D. and Sejnowski, T. J. (2018). Gradient descent for spiking neural networks. In *NeurIPS*, pages 1433–1443.

Jaderberg et al., 2016. 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*.

Kappel et al., 2018. Kappel, D., Legenstein, R., Habenschuss, S., Hsieh, M., and Maass, W. (2018). A dynamic connectome supports the emergence of stable computational function of neural circuits through reward-based learning. *eNeuro*.

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

Lillicrap et al., 2016. 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.

Lillicrap and Santoro, 2019. Lillicrap, T. P. and Santoro, A. (2019). Backpropagation through time and the brain. *Current Opinion in Neurobiology*, 55:82–89.

MacLean et al., 2015. 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.

Mnih et al., 2016. Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *ICML*, pages 1928–1937.

Morcos and Harvey, 2016. Morcos, A. S. and Harvey, C. D. (2016). History-dependent variability in population dynamics during evidence accumulation in cortex. *Nature Neuroscience*, 19(12):1672.

Mujika et al., 2018. Mujika, A., Meier, F., and Steger, A. (2018). Approximating real-time recurrent learning with random kronecker factors. In *NeurIPS*, pages 6594–6603.

Murray, 2019. Murray, J. M. (2019). Local online learning in recurrent networks with random feedback. *eLife*, 8:e43299.

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

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

Pozzorini et al., 2015. Pozzorini, C., Mensi, S., Hagens, O., Naud, R., Koch, C., and Gerstner, W. (2015). Automated high-throughput characterization of single neurons by means of simplified spiking models. *PLoS computational biology*, 11(6):e1004275.

Roelfsema and Holtmaat, 2018. Roelfsema, P. R. and Holtmaat, A. (2018). Control of synaptic plasticity in deep cortical networks. *Nature Reviews Neuroscience*, 19(3):166–180.

Roeper, 2013. Roeper, J. (2013). Dissecting the diversity of midbrain dopamine neurons. *Trends in neurosciences*, 36(6):336–342.

638 Roth et al., 2019. Roth, C., Kanitscheider, I., and Fiete, I. (2019). Kernel rnn learning (kernl).
639 *ICLR*.

640 Runyan et al., 2017. Runyan, C. A., Piasini, E., Panzeri, S., and Harvey, C. D. (2017). Distinct
641 timescales of population coding across cortex. *Nature*, 548:92–96.

642 Sajad et al., 2019. Sajad, A., Godlove, D. C., and Schall, J. D. (2019). Cortical microcircuitry
643 of performance monitoring. *Nature Neuroscience*, 22(2):265.

644 Sanhueza and Lisman, 2013. Sanhueza, M. and Lisman, J. (2013). The camkii/nmdar complex
645 as a molecular memory. *Molecular brain*, 6(1):10.

646 Sutton and Barto, 2018. Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An*
647 *Introduction*. MIT press.

648 Tallec and Ollivier, 2018. Tallec, C. and Ollivier, Y. (2018). Unbiased online recurrent opti-
649 mization. *ICLR*.

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

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

654 Yagishita et al., 2014. Yagishita, S., Hayashi-Takagi, A., Ellis-Davies, G. C., Urakubo, H.,
655 Ishii, S., and Kasai, H. (2014). A critical time window for dopamine actions on the struc-
656 tural plasticity of dendritic spines. *Science*, 345(6204):1616–1620.

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

Acknowledgments

This research/project was supported by the Human Brain Project (Grand Agreement number 785907) and the SYNCH project (Grand Agreement number 824162) of the European Union. We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Quadro P6000 GPU used for this research. Computations were carried out on the Human Brain Project PCP Pilot Systems at the Juelich Supercomputing Centre, which received co-funding from the European Union (Grand Agreement number 604102) and on the Vienna Scientific Cluster (VSC).

We thank Thomas Bohnstingl, Wulfram Gerstner, Christopher Harvey, Martin Vinck, Jason MacLean, Adam Santoro, Christopher Summerfield, and Yuqing Zhu for helpful comments on an earlier version of the manuscript.

Supplementary materials

Supplementary Text

Figs. S1 to S5

Movies S1 to S6

Supplementary materials for: *A solution to the learning dilemma for recurrent networks of spiking neurons*

Guillaume Bellec[1,°], Franz Scherr[1,°], Anand Subramoney[1], Elias Hajek[1], Darjan Salaj[1], Robert Legenstein[1], Wolfgang Maass[1,*]

[1]Institute of Theoretical Computer Science, Graz University of Technology, Infeldgasse 16b, Graz, Austria

° First authors.

* To whom correspondence should be addressed; E-mail: maass@igi.tugraz.at.

August 16, 2019

Contents

S1 Eligibility traces

Eligibility traces have been introduced in Section "Mathematical basis for e-prop" in Results. Here, we provide further information on eligibility traces. In Section S1.1, we discuss an alternative view on eligibility traces as derivatives. Second, we extend in Section S1.3 our treatment of eligibility traces for LSNNs in Methods to include non-uniform synaptic delays.

S1.1 Viewing eligibility traces as derivatives

There exists an alternative definition of the eligibility traces that is perhaps more intuitive than the recursive equation in (19). For this we need to define a notion of derivative $\frac{\widetilde{\partial}\mathbf{h}_j^t}{\partial W_{ji}}$ that quantifies the influence of an infinitesimal change of W_{ji} on the hidden state $\mathbf{h}_j^t$ through the internal processes of neuron j. Unlike the partial derivative $\frac{\partial \mathbf{h}_j^t}{\partial W_{ji}}$ it takes the full neuron history into account and not only the update of the hidden state at time step t. In comparison to the total derivative $\frac{d\mathbf{h}_j^t}{dW_{ji}}$ it ignores that a spike of neuron j might influence its future self through the recurrent connections. Defining the derivative $\frac{\widetilde{\partial} z_j^t}{\partial W_{ji}}$ according to the same principles, the eligibility traces and eligibility vectors can be defined by:

$$\boldsymbol{\epsilon}_{ji}^t = \frac{\widetilde{\partial}\boldsymbol{h}_j^t}{\partial W_{ji}} \tag{S1}$$

$$e_{ji}^t = \frac{\widetilde{\partial} z_j^t}{\partial W_{ji}} \,. \tag{S2}$$

More formally, $\frac{\widetilde{\partial}\mathbf{h}_j^t}{\partial W_{ji}}$ is the total derivative computed in the computational graph where the cross neuron dependencies are ignored, i.e. where $\frac{\partial \mathbf{h}_j^t}{\partial z_i^{t-1}}$ and $\frac{\partial z_j^t}{\partial z_i^{t-1}}$ are assumed to be zero for all i, j and t. This definition is equivalent to the previous one because, when inter neuron dependencies are ignored, the gradient $\frac{\widetilde{\partial}\mathbf{h}_j^{t'}}{\partial W_{ji}}$ is given by the sum $\sum_{t \leq t'} \frac{\widetilde{\partial}\mathbf{h}_j^{t'}}{\partial \mathbf{h}_j^t} \frac{\partial h_j^t}{\partial W_{ji}}$ and one recognizes here the eligibility vector given in equation (17). Equation (S2) follows since $e_{ji}^t = \frac{\partial z_j^t}{\partial \mathbf{h}_j^t} \cdot \epsilon_{ji}^t = \frac{\widetilde{\partial} z_j^t}{\partial W_{ji}}$. By extension of this notation of derivative to other quantities one can summarize *symmetric e-prop*

as the replacement of $\frac{dE}{dW_{ji}}$ by $\frac{\widetilde{\partial} E}{\partial W_{ji}}$ in stochastic gradient descent.

S1.2 Eligibility traces for LSNNs with membrane potential reset

The eligibility traces derived in the methods do not take the reset term into account. We derive here the eligibility traces that can correct for this. Note however that we did not observe an improvement when using this more complex model on the speech recognition and evidence accumulation tasks.

Eligibility traces for LIF neurons. When taking into account the reset, the partial derivative $\frac{\partial \boldsymbol{h}_j^{t+1}}{\partial \boldsymbol{h}_j^t}$ becomes $\alpha - v_{\text{thr}}\psi_j^t$ instead of α and, accordingly to equation (19), the eligibility vector can be computed with the recursive formula: $\boldsymbol{\epsilon}_{ji}^{t+1} = (\alpha - \beta\psi_j^t)\boldsymbol{\epsilon}_{ji}^t + z_j^t$.

Eligibility traces for ALIF neurons. According to the dynamics of the ALIF neurons defined in equations (3)–(7) one coefficient differs in the matrix $\frac{\partial \boldsymbol{h}_j^{t+1}}{\partial \boldsymbol{h}_j^t} \in \mathbb{R}^{2\times 2}$ as soon as one takes the reset into account. The coefficient $\frac{\partial v_j^t}{\partial a_j^t}$ was 0 without reset and becomes now $v_{\text{thr}}\beta\psi_j^t$. Overall the full derivative $\frac{\partial \boldsymbol{h}_j^{t+1}}{\partial \boldsymbol{h}_j^t}$ is then equal to:

$$\frac{\partial \boldsymbol{h}_j^{t+1}}{\partial \boldsymbol{h}_j^t} = \begin{pmatrix} \alpha - v_{\text{thr}}\psi_j^t & v_{\text{thr}}\beta\psi_j^t \\ \psi_j^t & \rho - \beta\psi_j^t \end{pmatrix}. \tag{S3}$$

Even-though this algorithm in still practicable, the recursive propagation of the eligibility vector in equation (19) cannot be written in the form of two separable equations as done in equations (22) and (23). We preferred to ignore the reset in Methods to provide more interpretable equations for eligibility traces.

S1.3 Eligibility traces for LSNNs with non-uniform synaptic delays

In our derivation of eligibility traces for LSNNs, we used uniform synaptic delays to ease notation. Here, we detail how *e-prop* can be extended to non-uniform delays. Resulting rules

for synaptic plasticity favor then corresponding larger delays of several ms between pre- and post-synaptic firing. Let the delay of a synapse from neuron i to j be denoted by $c(j,i) > 0$. Similarly, let $d(j,i) \geq 0$ be the delay of a synapse that connects an input neuron i with neuron j. Using this definition, the dynamics of the membrane potential, see equation (3), is written as:

$$v_j^{t+1} = \alpha v_j^t + \sum_{i \neq j} W_{ji}^{\text{rec}} z_i^{t+1-c(j,i)} + \sum_i W_{ji}^{\text{in}} x_i^{t+1-d(j,i)} - z_j^t v_{\text{th}} \,. \tag{S4}$$

Like in the uniform delay case, we obtain $\frac{\partial v_j^{t+1}}{\partial v_j^t} = \alpha$. The difference for arbitrary delays becomes visible in $\frac{\partial v_j^t}{\partial W_{ji}^{\text{rec}}} = z_i^{t-c(j,i)}$ and in $\frac{\partial v_j^t}{\partial W_{ji}^{\text{in}}} = x_i^{t-d(j,i)}$. For recurrent weights, the component of the eligibility vector associated to the membrane potential is hence:

$$\epsilon_{ji,v}^t = \sum_{t' \leq t-c(j,i)} z_i^{t'} = \bar{z}_i^{t-c(j,i)} \,. \tag{S5}$$

As the dynamics of the threshold adaptation is unchanged, the update of $\epsilon_{ji,a}^t$ remains as given in equation (22). We obtain an eligibility trace

$$e_{ji}^t = \psi_j^t \left(\bar{z}_i^{t-c(j,i)} - \beta \epsilon_{ji,a}^t \right) \,. \tag{S6}$$

Analogously, we obtain the corresponding eligibility trace for input synapses by replacing z_i^t and $c(j,i)$ with x_i^t and $d(j,i)$ respectively.

S2 Optimization and regularization procedures

Here, we discuss how optimization of networks was implemented and techniques that were used to regularize networks.

S2.1 Optimization procedure

For *e-prop* and for *BPTT*, the weights were updated once after a batch of training trials. For simplicity, all the weight updates $\Delta W_{ji}^{\text{rec}}$ are written for the most basic version of stochastic

gradient descent ($\Delta W_{ji}^{\rm rec} = -\eta \widehat{\frac{dE}{dW_{ji}^{\rm rec}}}$, where $\widehat{\frac{dE}{dW_{ji}^{\rm rec}}}$ is the gradient estimate) in this article. In practice, we used Adam (*Kingma and Ba, 2014*) to boost stochastic gradient descent. We refer to (*Kingma and Ba, 2014*) for the computation of the weight updates that result from the gradient estimates.

S2.2 Firing rate regularization for LSNNs

To ensure a low firing rate in LSNNs, we added a regularization term $E_{\rm reg}$ to the loss function E. This regularization term had the form:

$$E_{\rm reg} = \frac{1}{2} \sum_j \left(f_j^{\rm av} - f^{\rm target} \right)^2 \,, \tag{S7}$$

where $f^{\rm target}$ is a target firing rate and $f_j^{\rm av} = \frac{1}{n_{\rm trials} T} \sum_t z_j^t$ is the average firing rate of neuron j. Here, the sum runs over the time steps of all the $n_{\rm trials}$ trials between two weight updates. To derive the plasticity rule that implements this regularization, we follow equation (25) in Methods. The partial derivative of the regularization loss has the form:

$$\frac{\partial E_{\rm reg}}{\partial z_j^t} = \frac{1}{n_{\rm trials} T} \left(f_j^{\rm av} - f^{\rm target} \right) . \tag{S8}$$

Inserting this expression into equation (25), we obtain the plasticity rule that implements the regularization:

$$\Delta W_{ji}^{\rm rec} = \eta \, C_{\rm reg} \sum_t \frac{1}{n_{\rm trials} T} \left(f^{\rm target} - f_j^{\rm av} \right) e_{ji}^t \,, \tag{S9}$$

where $C_{\rm reg}$ is a positive coefficient that controls the strength of the regularization. This plasticity rule is applied simultaneously together with the plasticity rule that minimizes the loss E. Note that this weight update fits the *e-prop* framework provided by equation (1) with a learning signal $L_j^{{\rm reg},t}$ proportional to $f^{\rm target} - f_j^{\rm av}$ available locally at neuron j. This learning signal $L_j^{{\rm reg},t}$ can simply be added to the task-specific learning signal L_j^t.

S2.3 Weight decay regularization

When using *adaptive e-prop*, readout and broadcast weights were regularized using L2 norm weight decay regularization. This was implemented by subtracting $C_{\text{decay}} \cdot W$ from each weight W that was regularized at each weight update, where $C_{\text{decay}} > 0$ is the regularization factor (see specific experiments for the value of C_{decay}). This weight decay in combination with the mirroring of the weight updates has the effect that, despite different initialization, the output weights and the adaptive boradcast weights converge to similar values. The remaining difference of performance between *symmetric* and *adaptive e-prop* reported in Fig. 2 and Fig. S2 may be explained by the different initializations.

S2.4 Optimization with rewiring for sparse network connectivity

Due to limited resources, neural networks in the brain and in neuromorphic harware are sparsely connected. In addition, the connectivity structure of brain networks is dynamic, with synaptic connections being added and deleted on the time scale of hours or days, which was shown to help the network to use the limited connectivity resources in an optimal manner (*Kappel et al., 2018*). In order to test whether *e-prop* is compatible with synaptic rewiring, we combined it with DEEP R (*Bellec et al., 2018*). DEEP R is based on a model for synaptic rewiring in the brain (*Kappel et al., 2018*) and allows to rewire sparse neural network models during training with gradients descent. The algorithm minimizes the loss function E subject to a constraint on the total number of connected synapses. To do so, each synaptic weight W_{ji} is assigned a fixed sign s_{ji} (it is defined to be excitatory or inhibitory) and an amplitude w_{ji}. Each potential synaptic connection can either be "active", i.e., the synaptic connection is realized, or "dormant", i.e., this potential connection is not realized.

For a dormant synaptic connection, the weight W_{ji} is set to be zero and the gradients and weight updates of the connection $i \to j$ are not computed. It means in *e-prop* that dor-

mant synapses do not require eligibility traces. For an active connection, the weight is defined as $W_{ji} = s_{ji} w_{ji}$ and the weight amplitude is updated according to the update $\Delta w_{ji} = s_{ji} \Delta W_{ji} - \eta C_{L1}$ where ΔW_{ji} is the weight update given here by *e-prop* and $C_{L1} = 0.01$ is an $L1$ regularization coefficient. To update the network structure such that the set of active connections is optimized along side their synaptic weights, DEEP R proceeds as follows after each weight update:

- every active connection for which the amplitude becomes negative is set to be dormant,
- and some dormant connections are selected randomly and set to be active with $w_{ji} = 0$ such that the total number of active connection remains constant.

We define the synapse signs s_{ji} such that 80% of the neurons are excitatory and 20% are inhibitory. Despite the constraint on the neuron signs and the constraint that 90% of the synapses should remain dormant throughout the learning process, *e-prop* and rewiring solve the evidence accumulation task of Fig. 3.

S3 Supervised learning with *e-prop*

S3.1 Synaptic plasticity rules for *e-prop* in supervised learning

Here, we derive synaptic plasticity rules that result from *e-prop* for supervised learning. We consider two cases: First, we derive plasticity rules for regression tasks, and second, for classification tasks.

We follow the scheme described by equation (25) in Methods. Hence the loss gradients $\frac{dE}{dW_{ji}}$ are estimated using the approximation $\widehat{\frac{dE}{dW_{ji}}} \overset{\text{def}}{=} \sum_t \frac{\partial E}{\partial z_j^t} e_{ji}^t$. Given the eligiblity traces that are derived in Methods and Section S4.4, what remains to be derived for each task is the expression of the relevant derivative $\frac{\partial E}{\partial z_j^t}$ and show that it can be computed online.

Regression tasks: Consider a regression problem with loss function $E = \frac{1}{2}\sum_{t,k}(y_k^t - y_k^{*,t})^2$, targets $y_k^{*,t}$ and outputs y_k^t as defined in equation (8). The partial derivative $\frac{\partial E}{\partial z_j^t}$ takes the form:

$$E = \frac{1}{2}\sum_{t,k}(y_k^t - y_k^{*,t})^2 \tag{S10}$$

$$\frac{\partial E}{\partial z_j^t} = \sum_k W_{kj}^{\text{out}} \sum_{t' \geq t} (y_k^{t'} - y_k^{*,t'})\kappa^{t'-t} \,. \tag{S11}$$

This seemingly provides an obstacle for online learning, because the partial derivative is a weighted sum over future errors. But this problem can be resolved. Following equation (1), the approximation $\widehat{\frac{dE}{dW_{ji}}}$ of the loss gradient is computed with *e-prop* as follows (we insert $\frac{\partial E}{\partial z_j^t}$ in place of the total derivative $\frac{dE}{dz_j^t}$):

$$\widehat{\frac{dE}{dW_{ji}}} = \sum_{t'} \frac{\partial E}{\partial z_j^{t'}} e_{ji}^{t'} \tag{S12}$$

$$= \sum_{k,t'} W_{kj}^{\text{out}} \sum_{t \geq t'} (y_k^t - y_k^{*,t})\kappa^{t-t'} e_{ji}^{t'} \tag{S13}$$

$$= \sum_{k,t} W_{kj}^{\text{out}} (y_k^t - y_k^{*,t}) \underbrace{\sum_{t' \leq t} \kappa^{t-t'} e_{ji}^{t'}}_{\stackrel{\text{def}}{=} \bar{e}_{ji}^t} \,, \tag{S14}$$

where we changed the order of summations in the last line. The second sum indexed by t' is now over previous events that can be computed online. It is just a low-pass filtered version of the eligibility trace e_{ji}^t. With this additional filtering of the eligibility trace with a time constant equal to that of the leak of output neurons, we see that *e-prop* takes into account the latency between an event at time t' and its impact on later errors at time t within the integration time window of the output neuron. Hence, implementing weight updates with gradient descent and learning rate η, the plasticity rule resulting from *e-prop* is given by the equation (26). The gradient of the loss function with respect to the output weights $\frac{dE}{dW_{kj}^{\text{out}}}$ can be implemented online without relying on the theory of *e-prop*. The plasticity rule resulting from gradient descent is

directly:

$$\Delta W_{kj}^{\text{out}} = -\eta \sum_t (y_k^t - y_k^{*,t}) \mathcal{F}_\kappa(z_j^t) \; . \tag{S15}$$

Similarly the update of the bias of the output neurons is $\Delta b_k^{\text{out}} = -\eta \sum_t (y_k^t - y_k^{*,t})$.

Classification tasks: We assume that K target categories are provided in the form of a K-dimensional one-hot encoded vector $\boldsymbol{\pi}^{*,t}$. To train recurrent networks in this setup, we replace the mean squared error by the cross entropy loss:

$$E = -\sum_{t,k} \pi_k^{*,t} \log \pi_k^t \; , \tag{S16}$$

where the probability for class k predicted by the network is given as $\pi_k^t = \text{softmax}_k(y_1^t, \ldots, y_K^t)$ $= \exp(y_k^t) / \sum_{k'} \exp(y_{k'}^t)$. To derive the modified learning rule that results from this loss function E, we replace $\frac{\partial E}{\partial z_j^t}$ of equation (S11) with the corresponding one resulting from (S16):

$$\frac{\partial E}{\partial z_j^t} = \sum_k W_{kj}^{\text{out}} \sum_{t' \geq t} (\pi_k^{t'} - \pi_k^{*,t'}) \kappa^{t'-t}. \tag{S17}$$

Following otherwise the same derivation as in equations (S12)-(S14), the plasticity rule in the case of classification tasks is given by equation (27).

Similarly, one obtains the plasticity rule for the output connections, where the only difference between the cases of regression and of classification is that the output y_k^t and the target $y_k^{*,t}$ are replaced by π_k^t and $\pi_k^{*,t}$ respectively: $\Delta W_{kj}^{\text{out}} = -\eta \sum_t (\pi_k^t - \pi_k^{*,t}) \mathcal{F}_\kappa(z_j^t)$. The update of the bias of the output neurons is $\Delta b_k^{\text{out}} = -\eta \sum_t (\pi_k^t - \pi_k^{*,t})$.

S3.2 Simulation details: speech recognition task (Fig. 2)

S3.2.1 Frame-wise phoneme classification

The goal of the frame-wise setup of the task is to classify audio-frames into phoneme classes. Every input sequence of audio-frames has a corresponding sequence of class labels of the same

length, hence the model does not need to align the input sequence to the target sequence. This task has been widely adopted as a speech recognition benchmark for recurrent neural networks (RNNs).

Details of the network model: We used a bi-directional network architecture (*Graves and Schmidhuber, 2005*), where the output of an LSNN was augmented by the output a second LSNN that received the input sequence in reverse time order. Each of the two networks consisted of 300 LIF neurons and 100 ALIF neurons. The neurons in the LSNNs had a membrane time constant of $\tau_m = 20$ ms, an adaptation time constant of $\tau_a = 200$ ms, an adaptation strength of $\beta = 0.184$, a baseline threshold $v_{\text{th}} = 1.6$, and a refractory period of 2 ms.

We used 61 output neurons in total, one for each class of the TIMIT dataset. The membrane time constant of the output neurons was $\tau_{out} = 3$ ms. A softmax was applied to their output, resulting in the corresponding class probabilities. The network model had ≈ 0.4 million weights.

Details of the dataset preparation and of the input preprocessing: We followed the same task setup as in (*Greff et al., 2017, Graves and Schmidhuber, 2005*). 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 input $\boldsymbol{x}^t$ was given as preprocessed audio that was obtained by the following procedure: computation of 13 Mel Frequency Cepstral Coefficients (MFCCs) with a frame size of 10 ms on an input window of length 25 ms, computation of the first and the second derivatives of MFCCs, concatenation of all computed factors. The 39 input channels were mapped to the range $[0, 1]$ according to the minimum/maximum values in the training set.

In order to map the inputs into the temporal time domain of LSNNs, each preprocessed audio frame was fed as inputs $\boldsymbol{x}^t$ to the LSNN for 5 consecutive 1 ms steps.

Details of the learning procedure: All networks were trained for a maximum of 80 epochs, where we used early stopping to report the test error at the point of the lowest error on the validation set. Weight updates were implemented using Adam with default hyperparameters (*Kingma and Ba, 2014*) except for ϵ_{Adam}, which was set to 10^{-5}. Gradients were computed using batches of size 32. We used L2 regularization in all networks by adding the term $10^{-5} \cdot \|W\|^2$ to the loss function, where W denotes all weights in the network. The learning rate was initialized to 0.01 and fixed during training. For *random e-prop* and *adaptive e-prop*, broadcast weights B_{jk} were initialized using a Gaussian distribution with a mean of 0 and a variance of 1 and $1/n$ respectively. In *adaptive e-prop*, we used in addition to the weight decay described above L2 weight decay on readout and broadcast weights according to S2.3 using a factor of $C_{\text{decay}} = 10^{-2}$. Firing rate regularization, as described in Section S2.2, was applied with $C_{\text{reg}} = 50$.

S3.2.2 Phoneme sequence recognition with CTC

We compared *e-prop* and *BPTT* on the task and the network architecture used in (*Graves et al., 2013*). The essential building blocks of this architecture were also used in (*Amodei et al., 2016*) for developing commercial software for speech-to-text transcriptions. In this architecture Connectionist Temporal Classification (CTC) is employed. This enabled us to train networks on unaligned sequence labeling tasks end-to-end. We considered the results of (*Graves et al., 2013*) that were obtained with three layers of bi-directional LSTMs, CTC, and *BPTT* as a reference. We are aware that this configuration cannot be adapted to an online implementation easily, due to the usage of a bi-directional LSTM and the CTC loss function. However, we believe that this task is still relevant to compare *BPTT* and *e-prop* because it is a well established benchmark for RNNs.

Details of the network model: The neurons were structured into 3 layers. The network was recurrently connected within a layer and had feedforward connections across layers. Each layer consisted of 80 LIF neurons and 720 ALIF neurons (9.1 million weights). The neurons in LSNNs had a membrane time constant of $\tau_m = 20$ ms, an adaptation time constant of $\tau_a = 500$ ms, an adaptation strength of $\beta = 0.074$, a baseline threshold $v_{\text{th}} = 0.2$, and a refractory period of 2 ms. Synaptic delays were randomly chosen from $\{1, 2\}$ ms with equal probability. The membrane time constant of output neurons was $\tau_{out} = 3$ ms.

***E-prop* with many layers of recurrent neurons:** If one naively applies *e-prop* in such a configuration, the partial derivative $\frac{\partial E}{\partial z_j^t}$ is non-zero only if j belongs to the last layer, whereas earlier layers would not receive any learning signal. To avoid this, we connected all neurons in all layers of the RNN to the output neurons. Therefore, the outputs y_k^t of the RNN was given as $y_k^t = \sum_{t' \leq t} \kappa^{t-t'} \sum_l \sum_i W_{kj}^{\text{out},(l)} z_j^{(l),t'}$, where $z_j^{(l),t'}$ denotes the visible state of a neuron j within the layer l. As a result, the learning signals in the case of *e-prop* were non-zero for neurons in every layer.

***E-prop* with the CTC loss function:** E_{CTC} is defined based on the log-likelihood of obtaining the sequence of labeled phonemes given the network outputs y_k^t. We refer to (*Graves et al., 2006*) for the formal definition of the probabilistic model. Equation (7.27) in (*Graves, 2012*) shows the gradient of the loss function E_{CTC} with respect to the activity of the outputs y_k^t that we denote as $\frac{dE}{dy_k^t}$. Using the linear relationship between the visible state $z_j^{(l),t}$ and the outputs y_k^t, we obtain that the partial derivative $\frac{\partial E_{CTC}}{\partial z_j^{(l),t}}$ that we need in order to find the learning signals used in *e-prop* are defined as $\sum_{t' \geq t} \kappa^{t'-t} \sum_k \frac{dE}{dy_k^{t'}} B_{jk}^{(l)}$. Here, $B_{jk}^{(l)}$ denote the broadcast weights to the layer l.

Details of the dataset preparation and of the input preprocessing: The TIMIT dataset was split in the same manner as in (*Graves et al., 2013*) and in the frame-wise version of the task. The raw audio was preprocessed before it was provided as an input $\boldsymbol{x}^t$ to the network. This included the following steps: computation of a Fourier-transform based filter-bank with 40 coefficients and an additional channel for the signal energy (with step size 10 ms and window size 25 ms), computation of the first and the second derivatives, concatenation of all computed factors, which totals to 123 input channels. Normalization over the training set was done in the same manner as in the frame-wise version of the task.

In order to map the inputs into the temporal time domain of LSNNs, each preprocessed audio frame was fed as inputs $\boldsymbol{x}^t$ to the LSNN for 5 consecutive 1 ms steps.

Details of the learning procedure: All models were trained for a total of 60 epochs, where gradients were computed using batches of 8 sequences. The learning rate was initialized to 10^{-3} and decayed every 15 epochs by a factor of 0.3. We used early stopping to report the test error, as in the previous task. Dropout was applied during training between the hidden layers and at the output neurons with a dropout probability of 0.3. As in the frame-wise setup, the weight updates were implemented using Adam with the default hyperparameters (*Kingma and Ba, 2014*) except for $\epsilon_{\text{Adam}} = 10^{-5}$. For *random e-prop* and *adaptive e-prop*, broadcast weights B_{jk} were initialized using a Gaussian distribution with a mean of 0 and a variance of 1 and $1/n$ respectively. In *adaptive e-prop*, we used L2 weight decay on readout and broadcast weights according to S2.3 using a factor of $C_{\text{decay}} = 10^{-4}$. When the global norm of gradients $N_{\text{clip}} = \|\widehat{\frac{dE}{dW_{ji}^{\text{in}}}}\|^2 + \|\widehat{\frac{dE}{dW_{ji}^{\text{rec}}}}\|^2 + \|\widehat{\frac{dE}{dW_{ji}^{\text{out}}}}\|^2$ was larger then 1, we scaled the gradients by a factor of $\frac{1}{N_{\text{clip}}}$. We used beam search decoding with a beam width of 100. As in (*Graves et al., 2013*), the networks were trained on all 61 phoneme labels but were then mapped to a reduced phoneme set (39 classes) for testing.

S3.3 Applying *e-prop* to an episodic memory task

The FORCE training method (*Nicola and Clopath, 2017*) arguably defines the state-of-the-art for training methods for RSNNs that do not need to backpropagate gradients through time. FORCE learning uses a synaptic plasticity rule that required knowledge of the values of all synaptic weights in the network. This rule was not argued to be biologically plausible, but no other method for training an RSNN to solve the task described below was known so far.

In order to compare *e-prop* to FORCE learning, we tested *e-prop* on the task to replay a movie segment that had been repeatedly presented to the network (*Nicola and Clopath, 2017*). Specifically, it had to generate at each time step the values of all pixels that described the video frame of the movie at that time step. This episodic memory task was arguably the most difficult task for which an RSNN was previously trained in (*Nicola and Clopath, 2017*),

Here, we considered an extension to this task: the RNN had to replay 1 out of 3 possible movies, where the desired movie index was provided as a cue to the network, see Fig. S1A. As in (*Nicola and Clopath, 2017*), the RNN received also a clock-like input signal to indicate the current position in the movie. We show in Fig. S1B that an LSNN can be trained to solve this task by either one of the *e-prop* versions (see Movie S1), and that *e-prop* performs almost as well as *BPTT*.

Details of the network model: We used an LSNN that consisted of 700 LIF neurons and 300 ALIF neurons. Each neuron had a membrane time constant of $\tau_m = 20$ ms and a refractory period of 5 ms. ALIF neurons had a threshold adaptation time constant of 500 ms, and a threshold adaptation strength of $\beta = 0.07$. All neurons had a baseline threshold of $v_{\text{th}} = 0.62$. All 5544 output neurons had a membrane time constant of $\tau_{out} = 4$ ms.

Details of the dataset preparation and of the input scheme: We manually chose three movie clips from the Hollywood 2 dataset (*Marszałek et al., 2009*), which contained between 0 and 2 scene cuts*, see Movie S1. The movie clips were clipped to a length of 5 seconds and spatially subsampled to a resolution of 66×28 pixels. Since our simulations used 1 ms as a discrete time step, we linearly interpolated between the frames of the original movie clips, which had a framerate of 25 frames per second. In total, we obtained a target signal with $66 \times 26 \times 3 = 5544$ dimensions, whose values were divided by a constant of 255, such that they fit in the range of $[0, 1]$.

The network received input from 115 input neurons, divided into 23 groups of 5 neurons. The first 20 groups indicated the current phase of the target sequence, similar to (*Nicola and Clopath, 2017*). Neurons in group $i \in \{0, 19\}$ produced regular spike trains with a firing rate of 50 Hz during the time interval $[250 \cdot i, 250 \cdot i + 250)$ ms and were silent at other times. The remaining 3 groups encoded which movie had to be replayed, where each group was assigned to one of the three movies. To indicate a desired replay of one specific movie, each neuron in the corresponding group produced a Poisson spike train with a rate of 50 Hz and was silent otherwise.

Details of the learning procedure: For learning, we carried out 5 second simulations, where the network produced a 5544 dimensional output pattern. Gradients were accumulated for 8 successive trials, after which weight updates were applied using Adam with a learning rate of $2 \cdot 10^{-3}$ and default hyperparameters (*Kingma and Ba, 2014*). The movie to be replayed in each trial was selected with uniform probability. After every 100 weight updates (iterations), the learning rate was decayed by a factor of 0.95. For *random e-prop*, we used random broadcast weights B_{jk} that were sampled from a Gaussian distribution with a mean of 0 and a variance

*sceneclipautoautotrain00019.avi, sceneclipautoautotrain00061.avi, sceneclipautoautotrain00071.avi

of 1. In *adaptive e-prop* we used L2 weight decay (see Section S2.3) for the broadcast weights B_{jk} and the output weights W_{ji}^{out} with a factor of $C_{\text{decay}} = 0.001$. To avoid an excessively high firing rate, regularization, as described in Section S2.2, was applied with $C_{\text{reg}} = 0.1$ and a target firing rate of $f^{\text{target}} = 10$ Hz.

S3.4 Simulation details: evidence accumulation task (Fig. 3)

This task was inspired by the task performed by mice in (*Morcos and Harvey, 2016*). Each trial was split into three periods: the cue period, the delay period, and the decision period. During the cue period, the agent was stimulated with 7 successive binary cues ("left" or "right"), and had to take a corresponding binary decision ("left" or "right") during the decision period. The trial was considered a success if the decision matched the side that was most often indicated by the 7 cues. No action was required during the delay period. Each cue lasted for 100 ms and the cues were separated by 50 ms. The duration of the delay was distributed uniformly between 500 ms and 1500 ms, and the decision period lasted for 150 ms.

Details of the network model and input scheme: We used an LSNN that consisted of 50 LIF neurons and 50 ALIF neurons. All neurons had a membrane time constant of $\tau_m = 20$ ms, a baseline threshold of $v_{\text{th}} = 0.6$, and a refractory period of 5 ms. The time constants of the threshold adaptation was set to $\tau_a = 2000$ ms, and its impact on the threshold was given as $\beta = 1.74 \cdot 10^{-2}$.

Input to this network was provided by 4 populations of 10 neurons each. The first two input populations encoded the cues as follows: when a cue indicated the "left" side (resp. the "right" side), all the neurons within the first (resp. the second) population produced Poisson spike trains with a firing rate of 40 Hz. The third input population spiked randomly throughout the decision period with a firing rate of 40 Hz and was silent otherwise. All the neurons in the last input

population produced stationary Poisson spike trains of 10 Hz throughout the trial, which was useful in particular to avoid that the network becomes quiescent during the delay.

Details of the learning procedure: For learning, we used *e-prop* for classification tasks, see Section S3.1. The target label $\pi_k^{*,t}$ was given as the correct output during the decision period at the end of a trial. To help the network solving the task, we used a curriculum with an increasing number of cues. We first trained with a single cue, and increased the number of cues to 3, 5 and finally 7. The number of cues increased each time the network achieved less than 8% error on 512 validation trials. The same criterion is used to stop training once 7 cues are reached.

Independent of the learning algorithm that was used (*BPTT*, *e-prop*), a weight update was applied once every 64 trials and the gradients were accumulated during those trials additively. All weight updates were implemented using Adam with default parameters (*Kingma and Ba, 2014*) and a learning rate of $5 \cdot 10^{-3}$. In the cases of *random e-prop* and *adaptive e-prop*, broadcast weights B_{jk} were initialized using a Gaussian distribution with mean 0 and variance 1. In *adaptive e-prop* we used L2 weight decay (see Section S2.3) for the broadcast weights B_{jk} and the output weights W_{ji}^{out} with a factor of $C_{\text{decay}} = 0.001$. In addition, firing rate regularization, as described in Section S2.2, was applied with $C_{\text{reg}} = 1.$ and a target firing rate of $f^{\text{target}} = 10$ Hz.

S4 Applying supervised learning with *e-prop* to artificial neural networks (LSTMs)

Here we show that *e-prop* can also be applied to artificial neural networks. We chose long short-term memory (LSTM) neworks (*Hochreiter and Schmidhuber, 1997*) for this demonstration, whose performance defines the standard for RNNs in machine learning. We demonstrate in Section S4.1 that LSTM networks can achieve competitive results on TIMIT when trained with

e-prop, followed by details on these simulations (Section S4.2). In the following sections, we provide details on the LSTM model used (Section S4.3) and on eligibility traces for LSTM units (Section S4.4).

S4.1 Speech recognition with LSTM networks and *e-prop*

In Results, we have used *e-prop* to train LSNNs on the speech recognition task TIMIT (see Fig. 2). To test whether *e-prop* is effective also for artificial neural networks, we applied it to LSTM network on the very same task in its two flavors of frame-wise classification and sequence transcription.

Supplementary figure S2 shows that *E-prop* approximates the performance of *BPTT* in both versions of TIMIT also for LSTM networks very well. As for LSNNs, we trained as in (*Graves et al., 2013*) an LSTM network consisting of a feedforward sequence of three recurrent networks in the more difficult version of TIMIT involving sequence transcription.

S4.2 Simulation details: speech recognition task with LSTMs (Fig. S2)

The data preparation in the two setups (frame-wise phoneme classification and phoneme sequence recognition) were identical to the LSNN case. They are described in Section S3.2. The details on the network models and training procedures are described next for the two task setups separately.

S4.2.1 Frame-wise phoneme classification with LSTM networks

Details of the network model: We used a bi-directional network architecture (*Graves and Schmidhuber, 2005*), where the output of an LSTM network was augmented by the output a second LSTM network that received the input sequence in reverse time order. Each of the two networks consisted of 200 LSTM units.

We used a 61-fold softmax output, one for each class of the TIMIT dataset. The LSTM had ≈ 0.4 million weights, which matched the number of weights in the LSNN for the same task.

Details of the learning procedure: LSTM networks were trained in the same way as LSNNs, see Section S3.2, except for the following differences in training hyper parameters: We decayed the learning rate after every 500 weight updates by a factor of 0.3. For L2 weight decay on readout and broadcast weights according to S2.3 we used a factor of $C_{\text{decay}} = 10^{-3}$ for LSTMs. As LSTM units are not spiking, we did not use firing rate regularization.

S4.2.2 Phoneme sequence recognition with CTC and LSTM networks

We compared *e-prop* and *BPTT* on the task and the network architecture used in (*Graves et al., 2013*). As for LSNNs, we employed Connectionist Temporal Classification (CTC) to achieve phoneme sequence recognition (see Section "Phoneme sequence recognition with CTC" in Section S3.2). This enabled us to train networks on unaligned sequence labeling tasks end-to-end.

Details of the network model: The neurons of were structured into 3 recurrent layers. In each layer there were 250 LSTM units. All neurons in all layers of the RNN were connected to the output layer (see "*E-prop* with many layers of recurrent neurons" in Section S3.2).

Details of the learning procedure: LSTM networks were trained in the same way as LSNNs, see Section S3.2. In the case of *BPTT*, we also used the peephole feature in the LSTM model.

S4.3 LSTM network model

We use a standard model for LSTM units (*Hochreiter and Schmidhuber, 1997*), for which the hidden state at time step t is a one dimensional vector containing only the content of the memory cell c_j^t, such that $\mathbf{h}_j^t \stackrel{\text{def}}{=} [c_j^t]$, and z_j^t is the value of its output. The memory cell can be viewed as a

register which supports writing, updating, deleting and reading. These operations are controlled independently for each cell j at each time t by input, forget and output gates (denoted by $\imath_j^t$, $\jmath_j^t$ and o_j^t respectively). The new cell state candidate that may replace the cell state c_j^{t-1} at each time step t is denoted $\tilde{c}_j^t$. The input, forget, and output sigmoidal gates as well as the cell state candidate of an LSTM unit j are defined by the following equations:

$$\imath_j^t = \sigma\Big(\sum_i W_{ji}^{\mathrm{rec},\imath} z_i^{t-1} + \sum_i W_{ji}^{\mathrm{in},\imath} x_i^t\Big) \tag{S18}$$

$$\jmath_j^t = \sigma\Big(\sum_i W_{ji}^{\mathrm{rec},\jmath} z_i^{t-1} + \sum_i W_{ji}^{\mathrm{in},\jmath} x_i^t\Big) \tag{S19}$$

$$o_j^t = \sigma\Big(\sum_i W_{ji}^{\mathrm{rec},o} z_i^{t-1} + \sum_i W_{ji}^{\mathrm{in},o} x_i^t\Big) \tag{S20}$$

$$\tilde{c}_j^t = \tanh\Big(\sum_i W_{ji}^{\mathrm{rec},c} z_i^{t-1} + \sum_i W_{ji}^{\mathrm{in},c} x_i^t\Big), \tag{S21}$$

where all the weights used here are parameters of the model (we also used biases that were omitted for readability). Using these notations, one can now write the update of the state of an LSTM unit j in a form that we can relate to our general formalism:

$$c_j^t = \jmath_j^t c_j^{t-1} + \imath_j^t \tilde{c}_j^t \tag{S22}$$

$$z_j^t = o_j^t c_j^t \,. \tag{S23}$$

In terms of the computational graph in Fig. 5 equation (S22) defines $M(c_j^{t-1}, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{W})$ and (S23) defines $f(c_j^t, \mathbf{z}^{t-1}, \mathbf{x}^t, \boldsymbol{W})$.

S4.4 Eligibility traces for LSTM units

Eligibility traces for LIF neurons and ALIF neurons were derived in Section "Derivation of eligibility traces for concrete neuron models" in Methods. Here, we derive eligibility traces for the weights of LSTM units.

To obtain the eligibility traces, we note that the state dynamics of an LSTM unit is given by: $\frac{\partial \boldsymbol{h}_j^{t+1}}{\partial \boldsymbol{h}_j^t} = \frac{\partial c_j^{t+1}}{\partial c_j^t} = \jmath_j^t$. For each weight $W_{ji}^{A,B}$ with A being either "in" or "rec" and B being $\imath$, $\jmath$,

or c, we compute a set of eligibility traces. For example, the eligibility vectors for the recurrent weights to the input gate $W_{ji}^{\text{rec},\imath}$, are updated according to equation (19), leading to:

$$\boldsymbol{\epsilon}_{ji}^{\imath,t} = \not{\imath}_j^{t-1} \boldsymbol{\epsilon}_{ji}^{\imath,t-1} + \widetilde{c}_j^t \imath_j^t (1 - \imath_j^t) z_i^{t-1} \,, \tag{S24}$$

resulting in eligibility traces:

$$e_{ji}^{\imath,t} = \not{o}_j^t \boldsymbol{\epsilon}_{ji}^{\imath,t} \,. \tag{S25}$$

Similarly, the eligibility traces for the input weights to the input gate are obtained by replacing z_i^{t-1} with x_i^t.

Output gates: The gradients with respect to the parameters of the output gate do not require additional eligibility traces. This is because the output gate contributes to the observable state but not to hidden state, see equations S22 and S23. Therefore, one can use the standard factorization of the error gradient as used in *BPTT*. For the recurrent weights to the output gates $W_{ji}^{\text{rec},\not{o}}$, the gradient is given by:

$$\frac{dE}{dW_{ji}^{\text{rec},\not{o}}} = \sum_t \frac{dE}{dz_j^t} \frac{\partial z_j^t}{\partial W_{ji}^{\text{rec},\not{o}}} = \sum_t \frac{dE}{dz_j^t} c_j^t \not{o}_j^t (1 - \not{o}_j^t) z_i^{t-1}. \tag{S26}$$

Hence, when applying *e-prop* to LSTM units, we use the same approximation of the ideal learning signal $\frac{dE}{dz_j^t}$ as for other parameters and the remaining term is local, depends only on t and $t-1$ and does not require eligiblity traces. For input weights to the output gate $W_{ji}^{\text{in},\not{o}}$, the gradient is obtained by replacing z_i^{t-1} with x_i^t.

S5 *Reward-based e-prop*: Application of *e-prop* to policy gradient RL

S5.1 Synaptic plasticity rules for *reward-based e-prop*

Here, we derive the synaptic plasticity rules that result from gradients of the loss function E, as given in equation (28), see Fig. S3 for the network architecture. As a result of the general actor-

critic framework with policy gradient, this loss function additively combines the loss function for the policy E_π (actor) and the value function E_V (critic).

We consider two cases: First, a simplified case where in each trial, one out of K discrete actions is taken at a single time point. In particular this action is taken at the end of the trial. This is the setup of the reward-based version of the evidence accumulation task of Fig. 3, see Fig. S4 for performance results. Second, we analyse a more general case where continuous actions are taken throughout the trial. This is the setup of the delayed arm reaching task (Fig. 4). For both cases, we derive the gradients for the parts E_π and E_V of the loss function E, and express the plasticity rules resulting from these gradients.

Task setup with a discrete action at the end of the trial (Fig. 3): In this setup, a discrete action $a \in \{1, \dots, K\}$ from a set of K possibilities needs to be taken at the last time step T of a trial, leading to a binary-valued reward r^T. As a result, the return R^T (denoted here for notational simplicity as R) is equal to r^T. We assume that the agent chooses action k with probability $\pi_k = \mathrm{softmax}_k(y_1^T, \dots, y_K^T) = \exp(y_k^T)/\sum_{k'} \exp(y_{k'}^T)$. Therefore, we can write E_π as:

$$E_\pi \quad = \quad -R \sum_k \mathbb{1}_{a=k} \log \pi_k \; . \tag{S27}$$

Here and in subsequent equations, we suppress the dependence of the term on the left hand side on the stochastic action a that is actually chosen and the resulting reward R. $\mathbb{1}_{a=k}$ is the one-hot encoded action and assumes a value of 1 only if $a = k$ and is 0 otherwise. Hence, although we sum over all possible actions, only the term corresponding to the action a that was taken is non zero. Interestingly, in the discrete action case, the loss function E_π is reminiscent of the one used for supervised classification, see equation (S16). But it exhibits two differences: firstly, the indicator of the selected action $\mathbb{1}_{a=k}$ replaces the target label π_k^*, and secondly, the loss is multiplied by the reward R.

In order to optimize E, as given in (28), we also need to consider $E_V = \frac{1}{2}(R - V)^2$, for which we can reuse the result for regression (S14). By application of gradient descent using equation (1), and using the estimator $\widehat{\frac{\partial E}{\partial z_j^t}}$ given in (29), we obtain the synaptic plasticity rule that implements *reward-based e-prop* in this case:

$$\Delta W_{ji}^{\text{rec}} = -\eta \underbrace{\left[(R - V) \sum_k B_{jk}^{\pi}(\pi_k - \mathbb{1}_{a=k}) - C_V(R - V)B_j^V\right]}_{L_j} \bar{e}_{ji}^T \,, \tag{S28}$$

where we denote with B_{jk}^{π} the broadcast weights from output neurons y_k, and with B_j^V the broadcast weights from the output neuron that produces the value prediction V. The choice of these broadcast weights then defines which variant of *reward-based e-prop* is employed (*reward-based symmetric e-prop*, *reward-based adaptive e-prop*, or *reward-based random e-prop*).

For the synaptic connections of output neurons, the loss gradient can be computed directly from the loss function (28). We also subtract the value prediction to reduce variance of the gradient estimate as in (29), and obtain for the update rules: $\Delta W_{kj}^{\pi,\text{out}} = -\eta(R - V)(\pi_k - \mathbb{1}_{a=k})\mathcal{F}_\kappa(z_j^T)$, and $\Delta W_j^V = \eta C_V(R - V)\mathcal{F}_\kappa(z_j^T)$. Similarly, the updates of the biases of output neurons are: $\Delta b_k^{\pi,\text{out}} = -\eta(R - V)(\pi_k - \mathbb{1}_{a=k})$, and $\Delta b^V = \eta C_V(R - V)$.

Continuous actions throughout the trial (Fig. 4A-C): In this setup, we assume that the agent can take at certain decision times $t_0, \ldots, t_n, \ldots$ real-valued actions $\boldsymbol{a}$. We also assume that each component k of this action vector follows independent Gaussian distributions, with a mean given by the output y_k and a fixed variance σ^2.

We consider first the regression problem defined by the loss function E_V, and note that a major difference to the previous case is that the return R^t integrates future rewards arrive long after an action was taken. We begin with the result for regression from equation (S14).

Substituting the relevant variables, we obtain an estimation of the loss gradient:

$$\widehat{\frac{dE_V}{dW_{ji}^{\text{rec}}}} = -\sum_{t'}(R^{t'} - V^{t'})W_j^{V,\text{out}}\,\bar{e}_{ji}^{t'}\,, \tag{S29}$$

where $W_j^{V,\text{out}}$ are the weights of the output neuron V_j^t predicting the value function $\mathbb{E}[R^t]$. In order to overcome the obstacle that an evaluation of the return $R^{t'}$ requires to know future rewards, we introduce temporal difference errors $\delta^t = r^t + \gamma V^{t+1} - V^t$, and use that $R^{t'} - V^{t'}$ is equal to the sum $\sum_{t\geq t'}\gamma^{t-t'}\delta^t$. We then reorganize the two sums over t and t' (note that the interchange of the summation order amounts to the equivalence between forward and backward view of RL (*Sutton and Barto, 2018*)):

$$\begin{aligned}
\widehat{\frac{dE_V}{dW_{ji}^{\text{rec}}}} &= -\sum_{t'}\Big(\sum_{t\geq t'}\gamma^{t-t'}\delta^t\Big)W_j^{V,\text{out}}\,\bar{e}_{ji}^{t'} &\text{(S30)}\\
&= -\sum_{t}\delta^t\sum_{t'\leq t}\gamma^{t-t'}W_j^{V,\text{out}}\,\bar{e}_{ji}^{t'} &\text{(S31)}\\
&= -\sum_{t}\delta^t\,\mathcal{F}_\gamma\big(W_j^{V,\text{out}}\,\bar{e}_{ji}^{t}\big)\,. &\text{(S32)}
\end{aligned}$$

For the other part E_π in the loss function E, we consider the estimator $\widehat{\frac{\partial E}{\partial z_j^t}}$ given in (29), and use our previous definition that each component k of the action follows an independent Gaussian, which has a mean given by the output y_k and a fixed variance σ^2. The estimator then becomes:

$$\widehat{\frac{\partial E_\pi}{\partial z_j^t}} = -\sum_k W_{kj}^{\pi,\text{out}}\sum_{\{n\,|\,t_n\geq t\}}\kappa^{t_n-t}(R^{t_n} - V^{t_n})\frac{a_k^{t_n} - y_k^{t_n}}{\sigma^2}\,, \tag{S33}$$

where $W_{kj}^{\pi,\text{out}}$ are the weights onto the output neurons y_k^t defining the policy π, and κ is the constant of the low-pass filtering of the output neurons. Following a derivation similar to equa-

tions (S12) to (S14), we arrive at an estimation of the loss gradient of the form:

$$\widehat{\frac{dE_\pi}{dW_{ji}^{\mathrm{rec}}}} = \sum_t \widehat{\frac{\partial E_\pi}{\partial z_j^t}} e_{ji}^t \tag{S34}$$

$$= -\sum_{t,k} W_{kj}^{\pi,\mathrm{out}} \sum_{\{n \,|\, t_n \geq t\}} (R^{t_n} - V^{t_n}) \frac{a_k^{t_n} - y_k^{t_n}}{\sigma^2} \kappa^{t_n - t} e_{ji}^t \tag{S35}$$

$$= -\sum_{n,k} (R^{t_n} - V^{t_n}) W_{kj}^{\pi,\mathrm{out}} \frac{a_k^{t_n} - y_k^{t_n}}{\sigma^2} \underbrace{\sum_{t \leq t_n} \kappa^{t_n - t} e_{ji}^t}_{\bar{e}_{ji}^{t_n}} . \tag{S36}$$

Like in the derivation of the gradient of E_V, this formula hides a sum over future rewards in R^{t_n} that cannot be computed online. It is resolved by introducing the backward view as in equation (S32). We arrive at the loss gradient:

$$\widehat{\frac{dE_\pi}{dW_{ji}^{\mathrm{rec}}}} = -\sum_t \delta^t \mathcal{F}_\gamma \Big(\sum_k W_{kj}^{\pi,\mathrm{out}} \frac{a_k^t - y_k^t}{\sigma^2} \bar{e}_{ji}^t \Big) . \tag{S37}$$

Importantly, an action is only taken at times $t_0, \ldots, t_n, \ldots$, hence for all other times, we set the term $(a_k^t - y_k^t)$ to zero.

Finally, the gradient of the loss function E is the sum of the gradients of E_π and E_V, equations (S32) and (S37) respectively. Application of stochastic gradient descent with a learning rate of η yields the synaptic plasticity rule given in the equations (30) and (31).

The gradient of E with respect to the output weights can be computed directly from equation (28) without the theory of *e-prop*. However, it also needs to account for the sum over future rewards that is present in the term $R^t - V^t$. Using a similar derivation as in equations (S30)-(S32) the plasticity rule for these weights becomes:

$$\Delta W_{kj}^{\pi,\mathrm{out}} = -\eta \sum_t \delta^t \mathcal{F}_\gamma \Big(\frac{y_k^t - a_k^t}{\sigma^2} \mathcal{F}_\kappa(z_j^t) \Big) \tag{S38}$$

$$\Delta W_j^{\mathrm{V,out}} = \eta C_V \sum_t \delta^t \mathcal{F}_\gamma \left(\mathcal{F}_\kappa(z_j^t) \right) . \tag{S39}$$

Similarly, we also obtain for the update rules of the biases of the output neurons: $\Delta b_k^{\pi,\mathrm{out}} = -\eta \sum_t \delta^t \mathcal{F}_\gamma \Big(\frac{y_k^t - a_k^t}{\sigma^2} \Big)$, and $\Delta b^{\mathrm{V,out}} = \eta C_V \sum_t \delta^t$.

S5.2 Simulation details: evidence accumulation task (Fig. S4)

The task considered in this experiment was the same as in Section S3.4, but while the task was there formulated as a supervised learning, the network is trained here using a reinforcement learning setup. In this setup, the agent had to choose a side at the end of the trial, which represented the two discrete action possibilities. A reward of 1 was given at the end of the trial if the agent selected the side on which more cues than on the other had previously been given, otherwise no reward was given. The network model remained the same as in the supervised setup. The result is shown in Fig. S4: The task can be learnt by *reward-based e-prop*.

Details of the decision process: In the reinforcement learning setup of the task, one binary action formalizes the decision of the agent ("left" of "right") at the end of the trial. This decision was sampled according to probabilities π_k that are computed from the network output using a softmax operation, see "Case of a discrete action at the end of a trial" in Section S5.1.

Details of the learning procedure: For learning, we simulated batches of 64 trials, and applied weight changes at the end of each batch. Independent of the learning method, we used Adam to implement the weight update, using gradients that were accumulated in 64 trials using a learning rate of $5 \cdot 10^{-3}$ and default hyperparameters (*Kingma and Ba, 2014*). For *random e-prop*, we sampled broadcast weights B_{jk} from a Gaussian distribution with a mean of 0 and a variance of 1. To avoid an excessively high firing rate, regularization, as described in Section S2.2, was applied with $C_{\text{reg}} = 0.1$ and a target firing rate of $f^{\text{target}} = 10$ Hz.

S5.3 Simulation details: delayed arm reaching task (Fig. 4)

Details of the arm model: The arm consisted of two links, with one link connected to the other link by a joint, which is itself connected by a joint to a fixed position in space. The

configuration of this arm model at time t can be described by the angles ϕ_1^t and ϕ_2^t of the two joints measured against the horizontal and the first link of the arm respectively, see Fig. 4A. 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: $\phi_i^t = \sum_{t' \leq t} \dot{\phi}_i^{t'} \delta t + \phi_i^0$ using $\delta t = 1\,\mathrm{ms}$.

Details of the delayed arm reaching task and of the input scheme: The agent could control the arm by setting the angular velocities of the two joints to a different value at every ms. There was a total of 8 possible goal locations, which were evenly distributed on a circle with a radius of 0.8. The arm was initially positioned so that its tip was equidistant from all the goals. In each trial, one of the 8 goals was chosen randomly, and indicated as the desired goal location in the first 100 ms of the trial. Each possible goal location was associated with a separate input channel, consisting of 20 neurons. They produced a Poisson spike train with a rate of 500 Hz while the corresponding goal location was indicated. After this cue was provided, a delay period of a randomly chosen length between $100 - 500$ ms started, during which the subject was penalized with a negative reward of -0.1 if it moved outside a central region of radius 0.3. After this delay period, a go cue instructed the subject to move towards the goal location. This cue was provided in a separate input channel of 20 neurons, which produced a Poisson spike train with a rate of 500 Hz for 100 ms. Once the tip of the arm had moved closer than a distance of 0.1 to the goal location, a positive reward of 1 was given to signal a success. A negative reward of -0.01 was given for every ms after the go cue started while the arm did not yet reach the goal, in order to encourage an efficient movement. Going far off the region of interest – a circle of radius 1 – was penalized with a negative reward of -0.1 at each ms. One trial lasted for a total of 1.5 seconds – i.e. the subject had 900 ms from the start of the go cue to reach the goal.

The agent also received its current configuration (angles of the arms ϕ_1 and ϕ_2, see Fig. 4A) as input at each time step in the following way: each one of the angles was encoded by a population of 30 neurons, where each neuron had a Gaussian tuning curve centered on values distributed evenly between 0 and 2π, with a firing rate peak of 100 Hz. The tuning curve had a standard deviation of $\frac{4}{30}$.

In addition, if the goal position was successfully reached, the network received this information using a separate input channel consisting of 20 neurons that produced a Poisson spike train with a rate of 500 Hz.

Details of the network model: The network consisted of 350 LIF neurons and 150 ALIF neurons. The membrane time constant of all neurons was $\tau_m = 20$ ms, with a baseline threshold $v_{\text{th}} = 0.6$ and a refractory period of 3 ms. All synaptic delays were 1 ms. The adaptation time constant of ALIF neurons was set to $\tau_a = 500$ ms, and the adaptation strength was $\beta_j = 0.07$. The membrane time constant of output neurons was given by $\tau_{out} = 20$ ms.

Actions (angular velocities for the 2 joints) were sampled from a Gaussian distribution with a mean of y_k^t, and a standard deviation of $\sigma = 0.1$, which was exponentially decayed over iterations so that it reached $\sigma = 0.01$ at the end.

Details of the learning procedure: The network was trained for a total of 16000 weight updates (iterations). In each iteration, a batch of 200 trials was simulated, and we applied weight changes at the end of each batch. Independent of the learning method, we used Adam to implement the weight update, with a learning rate of 10^{-3} and default hyperparameters (*Kingma and Ba, 2014*). For training with *BPTT*, gradients were computed for the loss function given in equation (28) (using the variance reduction that is made explicit in equation (29)). In the case of *e-prop*, we used equations (30) and (31). For *random e-prop*, the broadcast weights B_{jk} were initialized using a Gaussian distribution with mean 0 and variance 1. To avoid an excessively

high firing rate, regularization, as described in Section S2.2, was applied with $C_{\text{reg}} = 100$ and a target firing rate of $f^{\text{target}} = 10$ Hz.

S6 Evaluation of four variations of *e-prop* (Fig. S5)

We evaluate here the performance of four variations of *random e-prop*. In these variations, we used

- truncated eligibility traces for LIF neurons,
- global broadcast weights,
- temporally local broadcast weights, and
- a replacement of the eligibility trace by the corresponding term of the Clopath rule,

respectively. The considered task, whose implementation details are described in Section S6.5, is an extension of the task used in (*Nicola and Clopath, 2017*). In this task, an RSNN was trained to autonomously generate a 3 dimensional target signal for 1 second. Each dimension of the target signal was 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 indicated the current phase of the pattern.

In Fig. S5A, we show the spiking activity of a randomly chosen subset of 20 out of the 600 neurons in the RSNN along with the output of the three output neurons after application of *random e-prop* for 1, 100, and 500 seconds, respectively. In this representative example, the network achieved a very good fit to the target signal (normalized mean squared error 0.01).

S6.1 A truncated eligibility trace for LIF neurons

A replacement of the term $\bar{z}_i^t$ with z_i^t in equation (21) yields a performance that is reported in panel B of Fig. S5 as "Trunc. e-trace". Its performance is for the considered task only slightly

worse than that of *random e-prop*.

S6.2 Global broadcast weights

Since 3-factor rules have primarily been studied so far with a global 3rd factor, we asked how the performance of *e-prop* would change if the same broadcast weight would be used for broadcast connections between all output neurons k and network neurons j. We set this global broadcast weight equal to $\frac{1}{\sqrt{n}}$. Fig. S5B shows that the performance for the considered task is much worse than that of *random e-prop*. We have also tested this on TIMIT with LSNNs and found there an increase of the frame-wise error rate from 36.9% to 52% when replacing the broadcast weights of *random e-prop* with a global one. On the harder version of same task, the error rate at the sequence level increased from 34.7% to 60%.

S6.3 Temporally local broadcast weights

One can train RNNs also by applying the broadcast alignment method of (*Lillicrap et al., 2016*) and (*Nøkland, 2016*) for feedforward networks to the unrolled version (see Fig. 1B) of the RNN. In contrast to *e-prop*, this approach suggests to draw new random broadcast weights for each layer of the unrolled network, i.e., for each time step of the RNN. Fig. S5C shows that this variation of *random e-prop* performs much worse. However an intermediate version where the random broadcast weights are redrawn every 20 ms performs about equally well as *random e-prop* for the considered task.

S6.4 Replacing the eligibility trace by the corresponding term of the Clopath rule

The dependence of the synaptic plasticity rules from *e-prop* on the postsynaptic membrane potential through the pseudo-derivative in the eligibility traces yields some similarity to some previously proposed rules for synaptic plasticity, such as that of (*Clopath et al., 2010*), which

were motivated by experimental data on the dependence of synaptic plasticity on the postsynaptic membrane potential. We therefore tested the performance of *random e-prop*, where the eligibility trace was replaced by the corresponding term from the "Clopath rule":

$$[v_j^t - v_{\text{th}}^+]^+ [\overline{v}_j^t - v_{\text{th}}^-]^+ \overline{z}_i^{t-1} \ , \tag{S40}$$

where $\overline{v}_j^t$ is an exponential trace of the post synaptic membrane potential, with a time constant of 10 ms chosen to match their data. $[\cdot]^+$ is the rectified linear function. The thresholds v_{th}^- and v_{th}^+ were $\frac{v_{\text{th}}}{4}$ and 0 respectively. Fig. S5B shows that the resulting synaptic plasticity rule performed quite well.

S6.5 Simulation details: pattern generation task

The performance in this task is reported as a normalized mean squared error (nmse) that we defined for this task as: $\text{nmse} = \frac{\sum_{t,k}(y_k^t - y_k^{*,t})^2}{\sum_{t,k}(y_k^{*,t} - \overline{y}_k^*)^2}$, where we set $\overline{y}_k^* = \frac{1}{T}\sum_t y_k^{*,t}$.

Details of the network model and of the input scheme: We used a network that consisted of 600 LIF neurons. Each neuron had a membrane time constant of $\tau_m = 20$ ms and a refractory period of 3 ms. The firing threshold was set to $v_{\text{th}} = 0.41$. Output neurons used a membrane time constant of $\tau_{out} = 20$ 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.

Details of the target pattern: The target signal had a duration of 1000 ms and each component was given by the sum of four sinusoids, with fixed frequencies of 1 Hz, 2 Hz, 3 Hz, and 5 Hz. At the start of learning, the amplitude and phase of each sinusoid in each component

was drawn uniformly in the range $[0.5, 2]$ and $[0, 2\pi]$ respectively. This signal was not changed afterwards.

Details of the learning procedure: For learning, we computed gradients after every 1 second of simulation, and carried out the weight update using Adam (*Kingma and Ba, 2014*) with a learning rate of $3 \cdot 10^{-3}$ and default hyperparameters. After every 100 iterations, the learning rate was decayed by a factor of 0.7. For *random e-prop*, the broadcast weights B_{jk} were sampled from a Gaussian distribution with a mean of 0 and a variance of $\frac{1}{n}$, where n is the number of network neurons.

Firing rate regularization, as described in Section S2.2, was applied with $C_{\text{reg}} = 0.5$ and a target firing rate of $f^{\text{target}} = 10$ Hz.

References

Amodei et al., 2016. Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G., Chen, J., Chen, J., Chen, Z., Chrzanowski, M., Coates, A., Diamos, G., Ding, K., Du, N., Elsen, E., Engel, J., Fang, W., Fan, L., Fougner, C., Gao, L., Gong, C., Hannun, A., Han, T., Johannes, L., Jiang, B., Ju, C., Jun, B., LeGresley, P., Lin, L., Liu, J., Liu, Y., Li, W., Li, X., Ma, D., Narang, S., Ng, A., Ozair, S., Peng, Y., Prenger, R., Qian, S., Quan, Z., Raiman, J., Rao, V., Satheesh, S., Seetapun, D., Sengupta, S., Srinet, K., Sriram, A., Tang, H., Tang, L., Wang, C., Wang, J., Wang, K., Wang, Y., Wang, Z., Wang, Z., Wu, S., Wei, L., Xiao, B., Xie, W., Xie, Y., Yogatama, D., Yuan, B., Zhan, J., and Zhu, Z. (2016). Deep speech 2 : End-to-end speech recognition in english and mandarin. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 173–182, New York, New York, USA. PMLR.

Bellec et al., 2018. Bellec, G., Kappel, D., Maass, W., and Legenstein, R. (2018). Deep rewiring: Training very sparse deep networks. *International Conference on Learning Representations*.

Clopath et al., 2010. 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.

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

Graves, 2012. Graves, A. (2012). Supervised sequence labelling. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 5–13. Springer.

Graves et al., 2006. Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 369–376, New York, NY, USA. ACM. event-place: Pittsburgh, Pennsylvania, USA.

Graves et al., 2013. Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE.

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

Greff et al., 2017. Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2017). LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232.

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

Kappel et al., 2018. Kappel, D., Legenstein, R., Habenschuss, S., Hsieh, M., and Maass, W. (2018). A dynamic connectome supports the emergence of stable computational function of neural circuits through reward-based learning. *eNeuro*, 5(2):ENEURO–0301.

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

Lillicrap et al., 2016. 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.

Marszałek et al., 2009. Marszałek, M., Laptev, I., and Schmid, C. (2009). Actions in context. In *IEEE Conference on Computer Vision & Pattern Recognition*.

Morcos and Harvey, 2016. Morcos, A. S. and Harvey, C. D. (2016). History-dependent variability in population dynamics during evidence accumulation in cortex. *Nature Neuroscience*, 19(12):1672.

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

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

Sutton and Barto, 2018. Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT press.

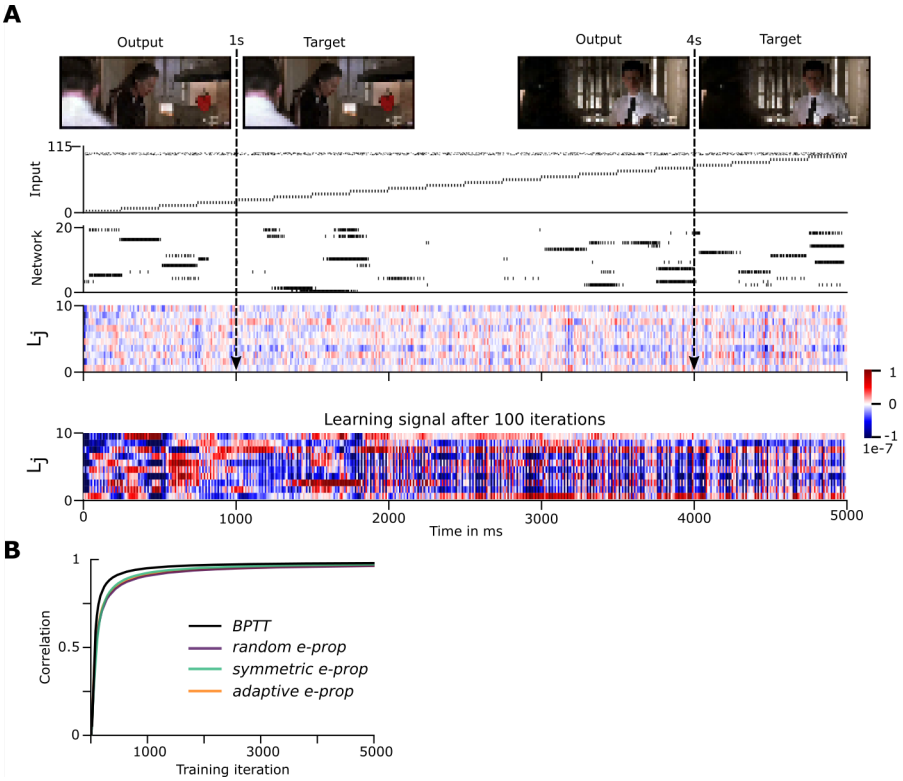

Figure S1: **Performance comparison of *BPTT* and *e-prop* on the episodic memory task from (*Nicola and Clopath, 2017*).** **A**) Input spikes, network activity (for 20 sample neurons), learning signals, and network outputs (at 1s and 4s, shown at the top) of an LSNN after 1000 training iterations. For comparison we also show learning signals after just 100 iterations, where their amplitude is still large. **B**) Performance of *BPTT* and *e-prop*.

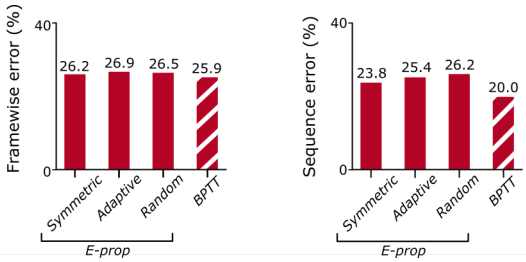

Figure S2: **LSTM trained with *BPTT* and *e-prop* on the TIMIT task.** Performance of *BPTT* and the three versions of *e-prop* frame-wise phoneme classification (left) and for phoneme sequence recognition (right).

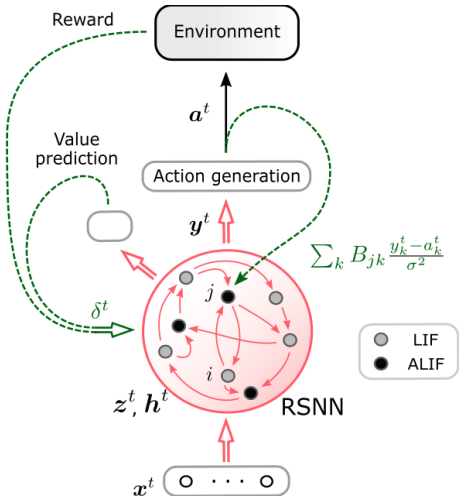

Figure S3: **Learning architecture for *reward-based e-prop***: The network input $\mathbf{x}^t$ consists of the current joint angles and input cues. The network produces output $\mathbf{y}^t$ which is used to stochastically generate the actions $\mathbf{a}^t$. In addition, the network produces the value prediction, which, along with the reward from the environment, is used to calculate the TD-error δ^t, The learning signals and the TD-errors are used to calculate the weight update, as denoted by the green dotted lines.

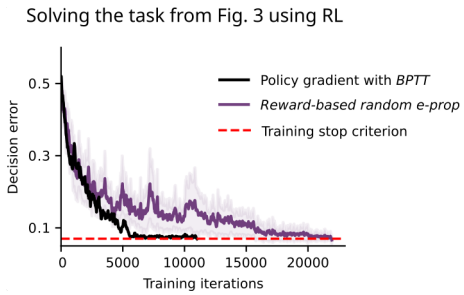

Figure S4: Performance of *reward based random e-prop* and *BPTT* for the RL version of the task from Fig. 3, applied to an LSNN consisting of 50 LIF and 50 ALIF neurons.

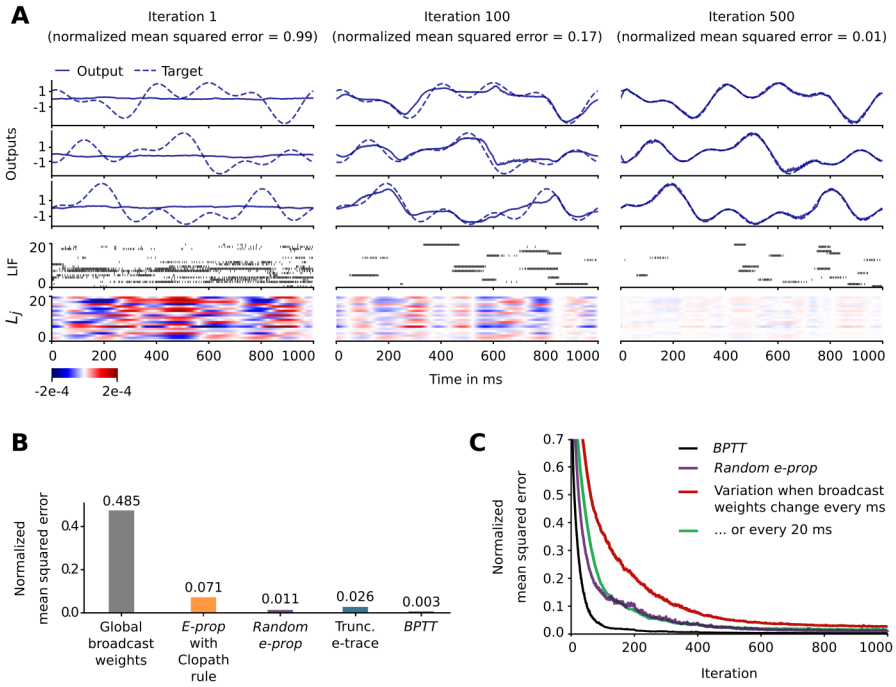

Figure S5: **Evaluation of several variants of *random e-prop*** **A**) The task is a classical benchmark task for learning in recurrent SNNs: learning to generate a target pattern, extended here to the challenge to simultaneously learn to generate 3 different patterns, which makes credit assignment for errors more difficult. Learning performance with *random e-prop* is shown after training for 1, 100, 500 s. **B**) Normalized mean squared error of several learning algorithms for this task after 500 s of training. "Clopath rule" denotes a replacement of the eligibility trace of *random e-prop* by a corresponding term proposed in (*Clopath et al., 2010*) based on experimental data. **C**) Learning curves for variations of *random e-prop* with temporally local broadcast weights.

730 **Movie S1**

731 Rodent task from (1, 2) that requires long-term credit assignment for learning: a rodent has
732 to learn to run along a linear track in a virtual environment, where it encounters several cues
733 on the left and the right side along the way. It then has to run through a corridor without cues
734 (giving rise to delays of varying lengths). At the end of the corridor, the rodent has to turn to
735 either the left or the right side of a T-junction, depending on which side exhibited more cues
736 along the way.

737 **Movie S2**

738 Dynamics of (BPTT) for the evidence accumulation task: First, a simulation of the network
739 has to be carried out in order to produce the network state of all neurons for all time steps.
740 After that the loss function E can be evaluated. Then the simulated network activity is replayed
741 backwards in time to assign credit to particular spikes that occurred before the loss function
742 became non-zero. One sees that the slow time constants that are present in the dynamics of
743 adapting thresholds of ALIF neurons result in slowly decaying non-vanishing gradients during
744 the backpropagation through time. In contrast, for LIF neurons the backpropagated gradients
745 vanish rather quickly.

746 **Movie S3**

747 The computation of the LSNN is accompanied by the computation of synapse specific eligi-
748 bility traces. An error in the computation only becomes apparent during the so-called decision
749 period at the end of a trial. In this last phase, a learning signal (L_j) that transmits deficiencies
750 of the network output is provided separately to each neuron. As can be seen from the video
751 that synapses that project to neurons with adapting thresholds (ALIF neurons) still have non-
752 vanishing eligibility traces during the last phase, and hence can be combined with the learning
753 signals at that time to implement long-term credit assignment.

754 **Movie S4**

755 Episodic memory task from (25) trained with random e-prop. The top row presents the
756 actual movie clip, and the output produced by the trained LSNN. The middle row shows the
757 input that is presented to the network: a channel that indicates which of the three learned clips
758 had to be replayed, and an array of input neurons that indicate the current timing in the clip.
759 The bottom row shows the spiking activity of a subset of the neurons in the LSNN (20 neurons
760 out of 1000). As can be seen, the network learned via e-prop to distinguish well between the
761 different clips and also, the LSNN was able to deal with scene cuts, which require the network
762 to change its output abruptly.

763 **Movie S5**

764 Illustration of the delayed arm-reaching task from Fig. 4: The agent gets the position of the
765 goal as the GOAL CUE during the first 100ms of a trial. This is followed by a delay period of
766 variable length during which the arm receives a negative reward for moving outside the area in
767 the center denoted by the dotted line. Noisy arm movements arise from the stochastic action
768 selection of policy gradient, and the arm needs to be actively steered back into the circle to
769 avoid further negative penalties. After the delay period, the agent gets a GO cue (the screen
770 turns yellow), after which no further negative rewards occur. The agent gets a large positive
771 reward if it reaches the small circle that was initially marked by the GOAL CUE.

772 **Movie S6**

773 A trial of the delayed arm-reaching task after training with random e-prop: One sees that
774 the arm moves to the goal immediately after the GO cue is received. The spike encoding of all
775 the inputs including the position of the arm (top), the GOAL CUE (bottom left), and the GO cue
776 (middle right) is shown in the middle panel of the video. The instantaneous rewards are shown
777 in the bottom panel of the video.