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

Guillaume Bellec\textsuperscript{1,\circledast}, Franz Scherr\textsuperscript{1,\circledast}, Anand Subramoney\textsuperscript{1}, Elias Hajek\textsuperscript{1}, Darjan Salaj\textsuperscript{1}, Robert Legenstein\textsuperscript{1}, Wolfgang Maass\textsuperscript{1,*}

\textsuperscript{1}Institute of Theoretical Computer Science, Graz University of Technology, Infeldgasse 16b, Graz, Austria
\circledast First authors.
* To whom correspondence should be addressed; E-mail: maass@igi.tugraz.at.

August 16, 2019

Contents

1. Eligibility traces 3
   1.1 Viewing eligibility traces as derivatives 3
   1.2 Eligibility traces for LSNNs with membrane potential reset 4
   1.3 Eligibility traces for LSNNs with non-uniform synaptic delays 4

2. Optimization and regularization procedures 5
   2.1 Optimization procedure 5
   2.2 Firing rate regularization for LSNNs 6
   2.3 Weight decay regularization 7
   2.4 Optimization with rewiring for sparse network connectivity 7

3. Supervised learning with \textit{e-prop} 8
   3.1 Synaptic plasticity rules for \textit{e-prop} in supervised learning 8
   3.2 Simulation details: speech recognition task (Fig. 2) 10
      3.2.1 Frame-wise phoneme classification 10
      3.2.2 Phoneme sequence recognition with CTC 12
   3.3 Applying \textit{e-prop} to an episodic memory task 15
   3.4 Simulation details: evidence accumulation task (Fig. 3) 17
### S4 Applying supervised learning with e-prop to artificial neural networks (LSTMs)

- **S4.1** Speech recognition with LSTM networks and e-prop ........................................... 19
- **S4.2** Simulation details: speech recognition task with LSTMs (Fig. S2) .......................... 19
  - **S4.2.1** Frame-wise phoneme classification with LSTM networks ............................. 19
  - **S4.2.2** Phoneme sequence recognition with CTC and LSTM networks ...................... 20
- **S4.3** LSTM network model ......................................................................................... 20
- **S4.4** Eligibility traces for LSTM units ........................................................................... 21

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

- **S5.1** Synaptic plasticity rules for reward-based e-prop .................................................. 22
- **S5.2** Simulation details: evidence accumulation task (Fig. S4) ...................................... 27
- **S5.3** Simulation details: delayed arm reaching task (Fig. 4) .......................................... 27

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

- **S6.1** A truncated eligibility trace for LIF neurons ......................................................... 30
- **S6.2** Global broadcast weights ..................................................................................... 31
- **S6.3** Temporally local broadcast weights ...................................................................... 31
- **S6.4** Replacing the eligibility trace by the corresponding term of the Clopath rule ......... 31
- **S6.5** Simulation details: pattern generation task ............................................................ 32
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 $\tilde{\partial} h_t^j / \partial W_{ji}$ that quantifies the influence of an infinitesimal change of $W_{ji}$ on the hidden state $h_t^j$ through the internal processes of neuron $j$. Unlike the partial derivative $\partial h_t^j / \partial W_{ji}$ 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 $d h_t^j / d W_{ji}$ it ignores that a spike of neuron $j$ might influence its future self through the recurrent connections. Defining the derivative $\tilde{\partial} z_t^j / \partial W_{ji}$ according to the same principles, the eligibility traces and eligibility vectors can be defined by:

$$e_{ji}^t = \frac{\tilde{\partial} h_t^j}{\partial W_{ji}}$$  \hspace{1cm} (S1)

$$e_{ji}^t = \frac{\tilde{\partial} z_t^j}{\partial W_{ji}}.$$  \hspace{1cm} (S2)

More formally, $\tilde{\partial} h_t^j / \partial W_{ji}$ is the total derivative computed in the computational graph where the cross neuron dependencies are ignored, i.e. where $\partial h_t^j / \partial z_i^{t-1}$ and $\partial z_i^{t-1} / \partial z_i^{t-1}$ 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 $\tilde{\partial} h_t^j / \partial W_{ji}$ is given by the sum $\sum_{t \leq t'} \frac{\partial h_t^j}{\partial W_{ji}} \frac{\partial h_t^j}{\partial W_{ji}}$, and one recognizes here the eligibility vector given in equation (17). Equation (S2) follows since $e_{ji}^t = \frac{\partial z_t^j}{\partial W_{ji}} = \tilde{\partial} z_t^j / \partial W_{ji}$. 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 \( \frac{\partial E}{\partial W_{ji}} \) 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 \( \frac{\partial h_t^{j+1}}{\partial h_t^j} \) becomes \( \alpha - v_{\text{thr}} \psi_t^j \) instead of \( \alpha \) and, accordingly to equation (19), the eligibility vector can be computed with the recursive formula:

\[
\epsilon_{ji}^{t+1} = (\alpha - \beta \psi_t^j) \epsilon_{ji}^t + z_t^j.
\]

#### Eligibility traces for ALIF neurons.

According to the dynamics of the ALIF neurons defined in equations (3)–(7) one coefficient differs in the matrix \( \frac{\partial h_t^{j+1}}{\partial h_t^j} \in \mathbb{R}^{2 \times 2} \) as soon as one takes the reset into account. The coefficient \( \frac{\partial v^t_t}{\partial a^t_j} \) was 0 without reset and becomes now \( v_{\text{thr}} \beta \psi_t^j \). Overall the full derivative \( \frac{\partial h_t^{j+1}}{\partial h_t^j} \) is then equal to:

\[
\frac{\partial h_t^{j+1}}{\partial h_t^j} = \begin{pmatrix} \alpha - v_{\text{thr}} \psi_t^j & v_{\text{thr}} \beta \psi_t^j \\ \psi_t^j & \rho - \beta \psi_t^j \end{pmatrix}.
\]

(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:

\[
v_{j}^{t+1} = \alpha v_{j}^{t} + \sum_{i \neq j} W_{ji}^{\text{rec}} z_{i}^{t+1} - c(j, i) + \sum_{i} W_{ji}^{\text{in}} x_{i}^{t+1} - d(j, i) - z_{j}^{t} v_{th} .
\]

(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:

\[
\epsilon_{ji,v}^{t} = \sum_{t' \leq t - c(j, i)} z_{i}^{t'} = z_{i}^{t} - c(j, i) .
\]

(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

\[
\epsilon_{ji}^{t} = \psi_{j}^{t} \left( z_{i}^{t} - c(j, i) - \beta \epsilon_{ji,a}^{t} \right) .
\]

(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 \( \Delta W_{ji}^{\text{rec}} = -\eta \hat{d}E_{ji}^{\text{rec}}, \) where \( \hat{d}E_{ji}^{\text{rec}} \) is the gradient estimate) in this article. In practice, we used Adam \((\text{Kingma and Ba, 2014})\) to boost stochastic gradient descent. We refer to \((\text{Kingma and Ba, 2014})\) 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 \( E_{\text{reg}} \) to the loss function \( E \). This regularization term had the form:

\[
E_{\text{reg}} = \frac{1}{2} \sum_j \left( f_{av}^j - f_{\text{target}} \right)^2 , \tag{S7}
\]

where \( f_{\text{target}} \) is a target firing rate and \( f_{av}^j = \frac{1}{n_{\text{trials}} T} \sum_t z_t^j \) is the average firing rate of neuron \( j \). Here, the sum runs over the time steps of all the \( n_{\text{trials}} \) 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:

\[
\frac{\partial E_{\text{reg}}}{\partial z_{t}^{j}} = \frac{1}{n_{\text{trials}} T} (f_{av}^{j} - f_{\text{target}}) . \tag{S8}
\]

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

\[
\Delta W_{ji}^{\text{rec}} = \eta C_{\text{reg}} \sum_t \frac{1}{n_{\text{trials}} T} (f_{\text{target}} - f_{av}^{j}) e_{tji}^{j} , \tag{S9}
\]

where \( C_{\text{reg}} \) 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 \( e\text{-prop} \) framework provided by equation (1) with a learning signal \( L_{\text{reg},t}^{j} \) proportional to \( f_{\text{target}} - f_{av}^{j} \) available locally at neuron \( j \). This learning signal \( L_{\text{reg},t}^{j} \) can simply be added to the task-specific learning signal \( L_{t}^{j} \).
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 $C_{\text{decay}}$). 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 broadcast 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 hardware 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 $i \rightarrow j$ 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 $\Delta W_{ji}$ 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 $\frac{dE}{dW_{ji}} \defeq \sum_t \frac{\partial E}{\partial z_j} e_t^{j,i}$. Given the eligibility 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 $\frac{\partial E}{\partial z_j}$ 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 $\frac{\partial E}{\partial z_j}$ takes the form:

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

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

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 $\hat{\frac{dE}{dW_{ji}}} = \frac{\partial E}{\partial z_j^t}$ 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 $\frac{dE}{dW_{ji}}$):

$$\hat{\frac{dE}{dW_{ji}}} = \sum_{t'} \frac{\partial E}{\partial z_j^{t'}} e_j^{t'} \quad (S12)$$

$$= \sum_{k,t'} W_{\text{out}}^{kji} \sum_{t' \geq t} (y_k^t - y_k^{*,t}) \kappa^{t'-t} e_j^{t'} \quad (S13)$$

$$= \sum_{k,t} W_{\text{out}}^{kji} (y_k^t - y_k^{*,t}) \kappa^{t'-t} \sum_{t' \leq t} e_j^{t'}, \quad (S14)$$

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 $e_j^{t'}$. 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 $\eta$, 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 $\frac{dE}{dW_{kj}}$ can be implemented online without relying on the theory of $e$-prop. The plasticity rule resulting from gradient descent is
directly:

\[ \Delta W_{kj}^{\text{out}} = -\eta \sum_t (y_t^k - y_t^{*t}) F_\kappa (z_j^t). \] (S15)

Similarly the update of the bias of the output neurons is \( \Delta b_{\text{out}}^k = -\eta \sum_t (y_t^k - y_t^{*t}) \).

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

\[ E = - \sum_{t,k} \pi_{k,t}^{*t} \log \pi_{k,t}^t, \] (S16)

where the probability for class \( k \) predicted by the network is given as \( \pi_{k,t}^t = \text{softmax}_k (y_1^t, \ldots, y_K^t) = \exp(y_k^t)/\sum_k \exp(y_k^t) \). 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):

\[ \frac{\partial E}{\partial z_j^t} = \sum_k W_{kj}^{\text{out}} \sum_{t' \geq t} (\pi_{k,t'}^{*t'} - \pi_{k,t'}^{*t'}) \kappa^{t'-t}. \] (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 \( y_t^k \) and the target \( y_t^{*t} \) are replaced by \( \pi_t^k \) and \( \pi_t^{*t} \) respectively: \( \Delta W_{kj}^{\text{out}} = -\eta \sum_t (\pi_t^k - \pi_t^{*t}) F_\kappa (z_j^t) \). The update of the bias of the output neurons is \( \Delta b_{\text{out}}^k = -\eta \sum_t (\pi_t^k - \pi_t^{*t}) \).

**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 $v_{th} = 1.6$, 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 $\approx 0.4$ 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 $x^t$ 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 $x^t$ 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 \cite{Kingma2014} except for $\epsilon_{\text{Adam}}$, 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 $10^{-5} \|W\|^2$ 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 \cite{Graves2013}. The essential building blocks of this architecture were also used in \cite{Amodei2016} 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 \cite{Graves2013} 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 $v_{th} = 0.2$, 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 $\frac{\partial E}{\partial z_j}$ 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 $y_k^l$ of the RNN was given as $y_k^l = \sum_{t'} \kappa^{t-t'} \sum_l \sum_i W^\text{out}_l z_j^{(l),t'},$ 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 $y_k^l$. 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 $y_k^l$ that we denote as $\frac{dE}{dy_k^l}$. Using the linear relationship between the visible state $z_j^{(l),t'}$ and the outputs $y_k^l$, 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 $\sum_{t'' \geq t} \kappa^{t''-t} \sum_k \frac{dE}{dy_k^l} B_{jk}^{(l)}$. 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 $x^t$ 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 $x^t$ 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 $N_{\text{clip}} = \| \frac{dE}{dW^m_{ji}} \|^2 + \| \frac{dE}{dW^\text{rec}_{ji}} \|^2 + \| \frac{dE}{dW^\text{out}_{ji}} \|^2$ 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 $v_{th} = 0.62$. 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 $66 \times 28$ 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
In adaptive e-prop we used L2 weight decay (see Section S2.3) for the broadcast weights $B_{jk}$ and the output weights $W_{ji}^{\text{out}}$ 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 $W_{ji}^{\text{out}}$ 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) networks (*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
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 LSTMs

**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 \( \approx 0.4 \) 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 \( c^t_j \), such that \( h^t_j \overset{\text{def}}{=} [c^t_j] \), and \( z^t_j \) 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 \( i^t_j \), \( f^t_j \)
and \( o^t_j \) respectively). The new cell state candidate that may replace the cell state \( c^t_{j-1} \) at each
time step \( t \) is denoted \( \tilde{c}^t_j \). 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:

\[
\begin{align*}
i^t_j &= \sigma \left( \sum_i W^{rec,i}_{ji} z^{t-1}_i + \sum_i W^{in,i}_{ji} x^t_i \right) \\
f^t_j &= \sigma \left( \sum_i W^{rec,f}_{ji} z^{t-1}_i + \sum_i W^{in,f}_{ji} x^t_i \right) \\
o^t_j &= \sigma \left( \sum_i W^{rec,o}_{ji} z^{t-1}_i + \sum_i W^{in,o}_{ji} x^t_i \right) \\
\tilde{c}^t_j &= \tanh \left( \sum_i W^{rec,c}_{ji} z^{t-1}_i + \sum_i W^{in,c}_{ji} x^t_i \right),
\end{align*}
\]

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:

\[
\begin{align*}
c^t_j &= f^t_j c^{t-1}_j + i^t_j \tilde{c}^t_j \\
z^t_j &= o^t_j c^t_j.
\end{align*}
\]

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

### S4.4 Eligibility traces for LSTM units

Eligibility traces for LIF neurons and ALIF neurons were derived in Section “Derivation of eligiblity 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:

\[
\frac{\partial h^{t+1}_j}{\partial h^t_j} = \frac{\partial c^{t+1}_j}{\partial c^t_j} = f^t_j. \text{ For each weight } W^{A,B}_{ji} \text{ with } A \text{ being either “in” or “rec” and } B \text{ being } i, f,
\]

21
or \( c \), we compute a set of eligibility traces. For example, the eligibility vectors for the recurrent weights to the input gate \( W_{ji}^{\text{rec},i} \), are updated according to equation (19), leading to:

\[
\epsilon_{ji}^{i,t} = \sigma_{ji}^{i,t} \epsilon_{ji}^{i,t-1} + \epsilon_{ji}^{i,t} (1 - \sigma_{ji}^{i,t}) z_{i}^{t-1},
\]  
\[
(S24)
\]

resulting in eligibility traces:

\[
\epsilon_{ji}^{i,t} = \sigma_{ji}^{i,t} \epsilon_{ji}^{i,t}.
\]  
\[
(S25)
\]

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 \( W_{ji}^{\text{rec},o} \), the gradient is given by:

\[
\frac{dE}{dW_{ji}^{\text{rec},o}} = \sum_{t} \frac{dE}{dz_{j}^{t}} \frac{\partial z_{i}^{t}}{\partial W_{ji}^{\text{rec},o}} = \sum_{t} \frac{dE}{dz_{j}^{t}} \sigma_{ji}^{o,t} (1 - \sigma_{ji}^{o,t}) z_{i}^{t-1}.
\]  
\[
(S26)
\]

Hence, when applying \( e\text{-prop} \) 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 eligibility traces. For input weights to the output gate \( W_{ji}^{\text{in},o} \), the gradient is obtained by replacing \( z_{i}^{t-1} \) with \( x_{i}^{t} \).

**S5**  
**Reward-based \( e\text{-prop} \): Application of \( e\text{-prop} \) to policy gradient RL**

**S5.1**  
**Synaptic plasticity rules for reward-based \( e\text{-prop} \)**

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 \( E_\pi \) (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 \( E_\pi \) 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 \( a \in \{1, \ldots, K\} \) 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 \( r_T^T \). As a result, the return \( R_T^T \) (denoted here for notational simplicity as \( R \)) is equal to \( r_T^T \). We assume that the agent chooses action \( k \) with probability \( \pi_k = \text{softmax}_k(y_1^T, \ldots, y_K^T) = \exp(y_k^T) / \sum_{k'} \exp(y_{k'}^T) \). Therefore, we can write \( E_\pi \) as:

\[
E_\pi = -R \sum_k 1_{a=k} \log \pi_k .
\]  
(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 \( R \). \( 1_{a=k} \) 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 \( E_\pi \) is reminiscent of the one used for supervised classification, see equation (S16). But it exhibits two differences: firstly, the indicator of the selected action \( 1_{a=k} \) replaces the target label \( \pi_k^* \), 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 $\frac{\partial E}{\partial z_j}$ given in (29), we obtain the synaptic plasticity rule that implements reward-based e-prop in this case:

$$\Delta W_{ji}^{rec} = -\eta \left[ (R - V) \sum_k B_{jk}^\pi (\pi_k - 1_{a=k}) - C_V (R - V) B_j^V \right] \bar{v}_ji^T, \quad (S28)$$

where we denote with $B_{jk}^\pi$ 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: $\Delta W_{kj}^\pi,_{out} = -\eta (R - V) (\pi_k - 1_{a=k}) \mathcal{F}_\pi(z_j^T_k)$, and $\Delta W_j^V = \eta C_V (R - V) \mathcal{F}_V(z_j^T)$. Similarly, the updates of the biases of output neurons are: $\Delta b_{kj}^\pi,_{out} = -\eta (R - V) (\pi_k - 1_{a=k})$, and $\Delta b_j^V = \eta C_V (R - V)$.

**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 $a$. 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 $\sigma^2$.

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 $R'$ 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:

\[
\hat{dE_V} = - \sum_{t'} (R_{t'} - V_{t'}) W_{ij}^{V,\text{out}} \bar{e}_{ji}^{t'},
\]

where \( W_{ij}^{V,\text{out}} \) are the weights of the output neuron \( V_j \) 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)):

\[
\hat{dE_V} = - \sum_{t'} \left( \sum_{t \geq t'} \gamma^{t-t'} \right) W_{ij}^{V,\text{out}} \bar{e}_{ji}^{t'}
\]

\[
= - \sum_t \delta_t \left( \sum_{t' \leq t} \gamma^{t-t'} W_{ij}^{V,\text{out}} \bar{e}_{ji}^{t'} \right)
\]

\[
= - \sum_t \delta_t \mathcal{F}_\gamma (W_{ij}^{V,\text{out}} \bar{e}_{ji}^{t'}).
\]

For the other part \( E_\pi \) in the loss function \( E \), we consider the estimator \( \hat{\partial E_\pi} \) 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 \( \sigma^2 \). The estimator then becomes:

\[
\hat{\partial E_\pi} = - \sum_k W_{kj}^{\pi,\text{out}} \sum_{\{n | t_n \geq t\}} \kappa^{t_n - t} (R_{t_n} - V_{t_n}) \frac{a_{kn}^{t_n} - y_k^{t_n}}{\sigma^2},
\]

where \( W_{kj}^{\pi,\text{out}} \) are the weights onto the output neurons \( y_k \) defining the policy \( \pi \), and \( \kappa \) 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:

\[
\frac{\hat{dE}_\pi}{dW_{ji}^{\text{rec}}} = \sum_t \frac{\partial \hat{E}_\pi}{\partial z_j^t} e_{ji}^t
\]  

(S34)

\[
= - \sum_{t,k} W_{kj}^{\pi,\text{out}} \sum_{\{n \mid t_n \geq t\}} (R^{t_n} - V^{t_n}) \frac{a_{kn}^{t_n} - y_{kn}^{t_n}}{\sigma^2} \sum_{\substack{t \leq t_n \leq t_n \geq t}} \kappa_{t_n - t} e_{ji}^{t_n}.
\]

(S35)

\[
= - \sum_{n,k} (R^{t_n} - V^{t_n}) W_{kj}^{\pi,\text{out}} \frac{a_{kn}^{t_n} - y_{kn}^{t_n}}{\sigma^2} \sum_{\substack{t \leq t_n \leq t_n \geq t}} \kappa_{t_n - t} e_{ji}^{t_n}.
\]

(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:

\[
\frac{\hat{dE}_\pi}{dW_{ji}^{\text{rec}}} = - \sum_t \delta^t F_\gamma \left( \sum_k W_{kj}^{\pi,\text{out}} \frac{a_{kn}^t - y_{kn}^t}{\sigma^2} e_{ji}^t \right).
\]

(S37)

Importantly, an action is only taken at times \(t_0, \ldots, t_n, \ldots\), hence for all other times, we set the term \((a_{kn}^t - y_{kn}^t)\) to zero.

Finally, the gradient of the loss function \(E\) is the sum of the gradients of \(E_\pi\) and \(E_V\), equations (S32) and (S37) respectively. Application of stochastic gradient descent with a learning rate of \(\eta\) 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:

\[
\Delta W_{kj}^{\pi,\text{out}} = -\eta \sum_t \delta^t F_\gamma \left( \frac{y_k^t - a_k^t}{\sigma^2} F_\kappa(z_j^t) \right)
\]

(S38)

\[
\Delta W_j^{V,\text{out}} = \eta C_V \sum_t \delta^t F_\gamma \left( F_\kappa(z_j^t) \right).
\]

(S39)

Similarly, we also obtain for the update rules of the biases of the output neurons:

\[
\Delta b_k^{\pi,\text{out}} = -\eta \sum_t \delta^t F_\gamma \left( \frac{y_k^t - a_k^t}{\sigma^2} \right), \text{ and } \Delta b_v^{V,\text{out}} = \eta C_V \sum_t \delta^t.
\]
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 $\pi_k$ 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 $\phi_1^t$ and $\phi_2^t$ of the two joints measured against the horizontal and the first link of the arm respectively, see Fig. 4A. For given angles, the position $y^t = (x^t, y^t)$ 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 $\delta t = 1$ 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 $\phi_1$ and $\phi_2$, 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 $2\pi$, 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 $v_{th} = 0.6$ 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 $y_k^i$, 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 $\overline{z}_i^t$ 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”:

\[
[v^+_j - v^+_{th}] + [\bar{v}^+_j - \bar{v}^+_{th}] + \bar{z}^{t-1}_i,
\]

where \(\bar{v}^+_j\) 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 \(\frac{v_{th}}{4}\) 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: \(\text{nmse} = \frac{\sum_{t,k} (y^*_t - y^*_{t,k})^2}{\sum_{t,k} (y^*_{t,k} - \bar{y}^*_k)^2}\), where we set \(\bar{y}^*_k = \frac{1}{T} \sum_t y^*_{t,k}\).

**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 \(v_{th} = 0.41\). 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**


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 $x^t$ consists of the current joint angles and input cues. The network produces output $y^t$ which is used to stochastically generate the actions $a^t$. In addition, the network produces the value prediction, which, along with the reward from the environment, is used to calculate the TD-error $\delta^t$. 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.
Movie S1

Rodent task from (1, 2) that requires long-term credit assignment for learning: a rodent has to learn to run along a linear track in a virtual environment, where it encounters several cues on the left and the right side along the way. It then has to run through a corridor without cues (giving rise to delays of varying lengths). At the end of the corridor, the rodent has to turn to either the left or the right side of a T-junction, depending on which side exhibited more cues along the way.
Movie S2

Dynamics of (BPTT) for the evidence accumulation task: First, a simulation of the network has to be carried out in order to produce the network state of all neurons for all time steps. After that the loss function $E$ can be evaluated. Then the simulated network activity is replayed backwards in time to assign credit to particular spikes that occurred before the loss function became non-zero. One sees that the slow time constants that are present in the dynamics of adapting thresholds of ALIF neurons result in slowly decaying non-vanishing gradients during the backpropagation through time. In contrast, for LIF neurons the backpropagated gradients vanish rather quickly.
The computation of the LSNN is accompanied by the computation of synapse specific eligibility traces. An error in the computation only becomes apparent during the so-called decision period at the end of a trial. In this last phase, a learning signal ($L_j$) that transmits deficiencies of the network output is provided separately to each neuron. As can be seen from the video that synapses that project to neurons with adapting thresholds (ALIF neurons) still have non-vanishing eligibility traces during the last phase, and hence can be combined with the learning signals at that time to implement long-term credit assignment.
Movie S4

Episodic memory task from (25) trained with random e-prop. The top row presents the actual movie clip, and the output produced by the trained LSNN. The middle row shows the input that is presented to the network: a channel that indicates which of the three learned clips had to be replayed, and an array of input neurons that indicate the current timing in the clip. The bottom row shows the spiking activity of a subset of the neurons in the LSNN (20 neurons out of 1000). As can be seen, the network learned via e-prop to distinguish well between the different clips and also, the LSNN was able to deal with scene cuts, which require the network to change its output abruptly.
Movie S5

Illustration of the delayed arm-reaching task from Fig. 4: The agent gets the position of the goal as the GOAL CUE during the first 100ms of a trial. This is followed by a delay period of variable length during which the arm receives a negative reward for moving outside the area in the center denoted by the dotted line. Noisy arm movements arise from the stochastic action selection of policy gradient, and the arm needs to be actively steered back into the circle to avoid further negative penalties. After the delay period, the agent gets a GO cue (the screen turns yellow), after which no further negative rewards occur. The agent gets a large positive reward if it reaches the small circle that was initially marked by the GOAL CUE.
Movie S6

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