

A Taxonomy of Recurrent Learning Rules

Guillermo Martín-Sánchez¹[0000-0002-5387-9579], Sander Bohte²[0000-0002-7866-278X], and Sebastian Otte¹[0000-0002-0305-0463]

¹ Neuro-Cognitive Modeling, University of Tübingen
Sand 14,72076 Tübingen, Germany
guillemartinsan@gmail.com

sebastian.otte@uni-tuebingen.de

² Machine Learning group, CWI
Science Park 123, NL-1098XG Amsterdam, The Netherlands
S.M.Bohte@cwi.nl

Abstract. Backpropagation through time (BPTT) is the de facto standard for training recurrent neural networks (RNNs), but it is non-causal and non-local. Real-time recurrent learning is a causal alternative, but it is highly inefficient. Recently, e-prop was proposed as a causal, local, and efficient practical alternative to these algorithms, providing an approximation of the exact gradient by radically pruning the recurrent dependencies carried over time. Here, we derive RTRL from BPTT using a detailed notation bringing intuition and clarification to how they are connected. Furthermore, we frame e-prop within in the picture, formalising what it approximates. Finally, we derive a family of algorithms of which e-prop is a special case.

Keywords: recurrent neural networks · backpropagation through time · real-time recurrent learning · forward propagation · e-prop.

1 Introduction

Backpropagation through time (BPTT) [6] is currently the most used algorithm for training *recurrent neural networks* (RNNs) and is derived from applying the chain rule (backpropagation) to the computational graph of the RNN unrolled in time. It suffers however from undesired characteristics both in terms of biological plausibility and large scale applicability: (i) it is *non-causal*, since at each time step it requires future activity to compute the current gradient of the loss with respect to the parameters; and (ii) it is *non-local*, since it requires reverse error signal propagating across all neurons and all synapses. An equivalent algorithm is *real-time recurrent learning* (RTRL) [8]. It uses eligibility traces that are computed at each time step recursively in order to be causal, and can therefore be computed online. However, this comes at the cost of very high computational and memory complexity, since all temporal forward dependencies have to be maintained over time. RTRL is, hence, also non-local. Recently, a new online learning algorithm, called *e-prop* [2] has been proposed, which is tailored for training *recurrent spiking neural networks* (RSNNs) with local neural dynamics.

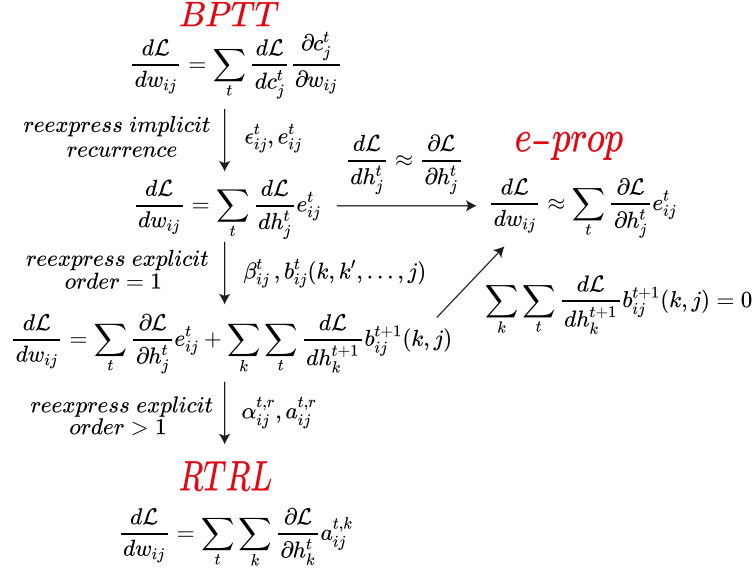


Fig. 1. Overview of all the algorithms and how they relate to each other.

The aim was to find an alternative to BPTT (and RTRL) that is causal, local, but also computational and memory efficient.

In this paper, we look in depth into the formalisation of BPTT and RTRL and formalise e-prop into the picture. To do so, we use the computational graph and notation of the architecture in the e-prop paper [2] to understand how these three algorithms relate to each other. Furthermore, in a posterior paper [9], it was shown that e-prop was an approximation of RTRL. Here, by formalising also RTRL in the same framework we indeed confirm the connection and make it more explicit (cf. Fig. 1). In the process, we uncover a family of algorithms determined by the level of approximation allowed to benefit from causality and locality. The main focus of this paper is to give intuition and understanding of all of these gradient computation rules.

1.1 Background

The most common way to train a model in supervised learning is to compute the gradient of a given loss \mathcal{L} with respect to the parameters θ , $d\mathcal{L}/d\theta$, and use this gradient in some gradient descent scheme of the form $\theta(\tau + 1) = \theta(\tau) - f(d\mathcal{L}/d\theta)$, where τ refers to the current update iteration and f is some gradient postprocessing. Therefore, we here focus on the algorithms for the computation (or approximation) of this gradient.

In particular, we focus on a general class of RNN models where we have n computational units. These units have hidden states at each time step c_i^t that influence the hidden state at the next time step c_i^{t+1} (implicit recurrence) as well as the output of the unit at the current time step h_i^t . The output h_i^t of a unit at a given time step influences the hidden state of the same and other units at the

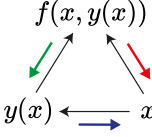
$$\frac{df}{dx} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial x}$$


Fig. 2. Simple example of computational graph and distinction between total and partial derivative of f with respect to x .

following time step c_j^{t+1} (explicit recurrence) through a weighted connection w_{ij} . Finally, these outputs also account for the model’s computation (either directly or through some other computations, e.g. a linear readout) and therefore are subject to evaluation by a loss function \mathcal{L} . The formalization here is agnostic to the particular dimensionality and computational relation between the variables and therefore apply for different RNNs, such as LSTMs [4] or RSNNs [1].

For a function $f(x, y(x))$ we distinguish the notation of the total derivative df/dx and the partial derivative $\partial f/\partial x$ because the first one represents the whole gradient through all paths, while the second one expresses only the direct relation between the variables. To illustrate: using the chain rule (cf. Fig. 2) and with the example $y = 2x$ and $f(x, y(x)) = xy$, the total derivative is calculated as:

$$\frac{df}{dx} = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y} \frac{\partial y}{\partial x} = y + x \cdot 2 = 4x \quad (1)$$

2 Backpropagation Through Time

In RNNs, since previous states affect the current state, the trick to applying the chain rule is to unroll the RNN in time, obtain a virtual feed-forward architecture and apply to this computational graph error-backpropagation [7]. The resulting algorithm is BPTT [6] and it is the currently most used algorithm to compute $d\mathcal{L}/dw_{ij}$ since it reuses many previous computations to be highly efficient. Here, we focus our attention on the role of the recurrences dividing the algorithm into the following steps:

Explicit Recurrences: Compute $d\mathcal{L}/dh_j^t$ using the recursive definition given by the explicit recurrences (cf. Fig. 3A):

$$\begin{aligned} \frac{d\mathcal{L}}{dh_j^t} &= \frac{\partial \mathcal{L}}{\partial h_j^t} + \frac{d\mathcal{L}}{dc_j^{t+1}} \frac{\partial c_j^{t+1}}{\partial h_j^t} + \sum_{k \neq j} \frac{d\mathcal{L}}{dc_k^{t+1}} \frac{\partial c_k^{t+1}}{\partial h_j^t} \\ &= \frac{\partial \mathcal{L}}{\partial h_j^t} + \sum_k \frac{d\mathcal{L}}{dc_k^{t+1}} \frac{\partial c_k^{t+1}}{\partial h_j^t} \end{aligned} \quad (2)$$

Implicit Recurrences: Compute $d\mathcal{L}/dc_j^t$ using the value of the previous step and the recursive definition given by the implicit recurrence (cf. Fig. 3B):

$$\frac{d\mathcal{L}}{dc_j^t} = \frac{d\mathcal{L}}{dh_j^t} \frac{\partial h_j^t}{\partial c_j^t} + \frac{d\mathcal{L}}{dc_j^{t+1}} \frac{\partial c_j^{t+1}}{\partial c_j^t} \quad (3)$$

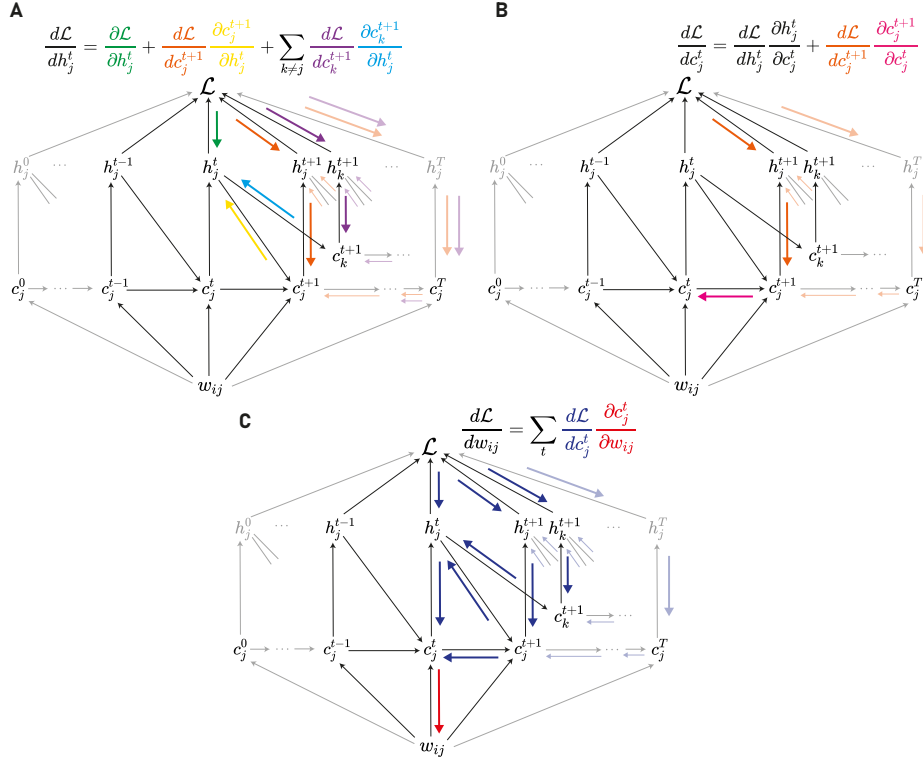


Fig. 3. Computational graph for A) explicit recurrences gradients, B) implicit recurrence gradients and C) final computation of BPTT.

BPTT: Finally, compute $d\mathcal{L}/dw_{ij}$ using the values obtained in the previous two steps for all time steps (cf. Fig. 3C):

$$\frac{d\mathcal{L}}{dw_{ij}} = \sum_t \frac{d\mathcal{L}}{dc_j^t} \frac{\partial c_j^t}{\partial w_{ij}} \quad (4)$$

We use explicit and implicit recurrences from the maximum time T backwards and for all $t \leq T$, and finally, sum all the results from Eq. 3. The existence of these recurrences makes BPTT present the following problems [5, 2]:

Non-locality: Due to the explicit recurrences, we need to take into account how the current synaptic strength w_{ij} between the neurons i and j affects the future value of the postsynaptic neuron: $\partial c_k^{t+1}/\partial h_j^t$ for all $k \neq j$ (cf. Eq. 2). This means that to compute the weight change for synapse magnitude w_{ij} we need information of the hidden variables c_k^{t+1} for all k . Moreover, this chain of dependencies continues at each time step, such that at the next time step we need information of the variables c_q^{t+2} for all q (including $q \neq j$) and so forth. The contraposition would be a **local** algorithm that does not require messages passing from every neuron to every synapse to compute the gradients, but rather only need information close to the given synapse.

Non-causality: Due to the three kinds of recurrences shown before we need to take into account all the gradients in the future (the same way as current layer computations need to use the gradients of posterior layers in feed-forward architectures), leading to two main problems. First, we need to compute the values of the variables (update locking) and the gradients (backwards locking) across all future time steps before computing the current gradient [3]. Secondly, all the values of all the variables across time have to be saved during the inference phase to be used while computing the gradients, requiring a memory overhead ($O(nT)$ with n neurons and T time steps). The contraposition would be a **causal** algorithm, that at each time step would only need information from previous and current activity to compute the current gradient. Therefore, it could do it at each time step (**online**) and while the inference is running.

3 Real-Time Recurrent Learning

RTRL [8] is a causal learning algorithm that can be implemented as an online learning algorithm and that computes the same gradient as BPTT, at the cost of being more computationally expensive. We derive the equation for RTRL starting with BPTT (cf. Eq. 4) via re-expressing the gradients that connect the computation with future gradients to obtain a causal algorithm. These gradients correspond to the implicit and explicit recurrences.

3.1 Re-expressing Implicit Recurrence

First, we re-express the implicit recurrence gradient $\partial c_j^{t+1}/\partial c_j^t$.

Unrolling the recursion: To unroll, we plug the equation of implicit recurrence Eq. 3 into Eq. 4:

$$\begin{aligned} \frac{d\mathcal{L}}{dw_{ij}} &= \sum_{t'} \left(\frac{d\mathcal{L}}{dh_j^{t'}} \frac{\partial h_j^{t'}}{\partial c_j^{t'}} + \frac{d\mathcal{L}}{dc_j^{t'+1}} \frac{\partial c_j^{t'+1}}{\partial c_j^{t'}} \right) \frac{\partial c_j^{t'}}{\partial w_{ij}} \\ &= \sum_{t'} \left(\frac{d\mathcal{L}}{dh_j^{t'}} \frac{\partial h_j^{t'}}{\partial c_j^{t'}} + \left(\frac{d\mathcal{L}}{dh_j^{t'+1}} \frac{\partial h_j^{t'+1}}{\partial c_j^{t'+1}} + (\dots) \frac{\partial c_j^{t'+2}}{\partial c_j^{t'+1}} \right) \frac{\partial c_j^{t'+1}}{\partial c_j^{t'}} \right) \frac{\partial c_j^{t'}}{\partial w_{ij}} \quad (5) \\ &= \sum_{t'} \sum_{t \geq t'} \frac{d\mathcal{L}}{dh_j^t} \frac{\partial h_j^t}{\partial c_j^t} \frac{\partial c_j^t}{\partial c_j^{t-1}} \dots \frac{\partial c_j^{t'+1}}{\partial c_j^{t'}} \frac{\partial c_j^{t'}}{\partial w_{ij}} \end{aligned}$$

Flip time indices: The derived formula is non-causal since it requires future gradients (for each t' we sum products of gradients with factors starting from $t \geq t'$). To make it causal, we change the indices as follows:

$$\frac{d\mathcal{L}}{dw_{ij}} = \sum_t \frac{d\mathcal{L}}{dh_j^t} \frac{\partial h_j^t}{\partial c_j^t} \sum_{t' \leq t} \frac{\partial c_j^t}{\partial c_j^{t-1}} \dots \frac{\partial c_j^{t'+1}}{\partial c_j^{t'}} \frac{\partial c_j^{t'}}{\partial w_{ij}} \quad (6)$$

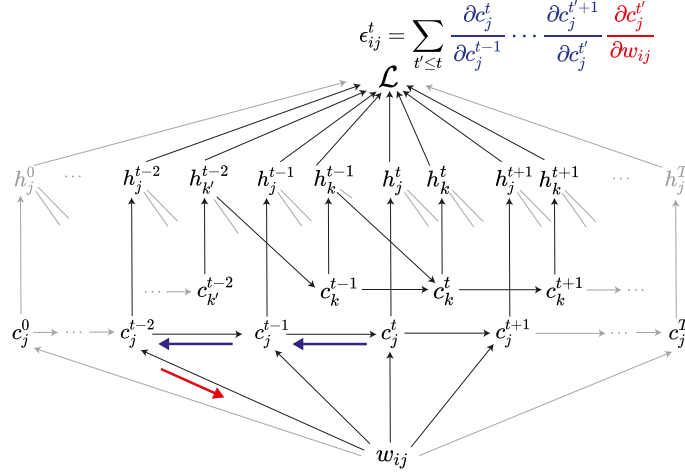


Fig. 4. Computational graph for the implicit variable ϵ_{ij}^t with $t' = t - 2$.

Definition (Implicit variable) We define the implicit variable ϵ_{ij}^t as:

$$\epsilon_{ij}^t := \sum_{t' \leq t} \frac{\partial c_j^t}{\partial c_j^{t-1}} \dots \frac{\partial c_j^{t'+1}}{\partial c_j^{t'}} \frac{\partial c_j^{t'}}{\partial w_{ij}} \quad (7)$$

Backwards interpretation: Starting at c_j^t , the implicit variable represents the sum over all the paths going backwards through the implicit recurrence until $c_j^{t'}$ and from there to the synaptic weight w_{ij} (cf. Fig. 4).

Forwards interpretation: The implicit variable represents how the hidden variable of neuron j has been affected by the synapse weight w_{ij} through time, i.e. taking into account also how the hidden variables at previous time steps have affected the variables at the current time step through the implicit recurrence.

Incremental computation: Importantly, there is a recursive relation to this variable that allows it to be updated at each time step:

$$\epsilon_{ij}^t = \frac{\partial c_j^t}{\partial c_j^{t-1}} \epsilon_{ij}^{t-1} + \frac{\partial c_j^t}{\partial w_{ij}} \quad (8)$$

Definition (Implicit eligibility trace) Given the implicit variable ϵ_{ij}^t , we define the implicit eligibility trace e_{ij}^t as:

$$e_{ij}^t := \frac{\partial h_j^t}{\partial c_j^t} \epsilon_{ij}^t \quad (9)$$

Since $\partial h_j^t / \partial c_j^t$ is causal and local, and so is the implicit variable ϵ_{ij}^t (can be computed at each time step and is specific for each synapse), then the implicit eligibility trace e_{ij}^t is also **causal** and **local**.

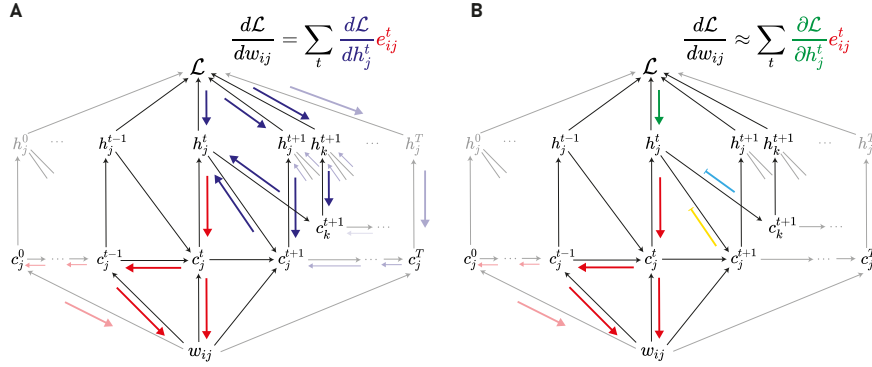


Fig. 5. Computational graph for A) BPTT re-expressed with implicit eligibility trace (cf. Eq. 10) B) symmetric e-prop.

Final equation with re-expressed implicit recurrence: With all of this combined, BPTT (cf. Eq. 4) has become (substituting Eq. 7 in Eq. 6) the following (cf. Fig. 5A):

$$\frac{d\mathcal{L}}{dw_{ij}} = \sum_t \frac{d\mathcal{L}}{dh_j^t} \frac{\partial h_j^t}{\partial c_j^t} e_{ij}^t = \sum_t \frac{d\mathcal{L}}{dh_j^t} e_{ij}^t \quad (10)$$

Even though e_{ij}^t is causal and local, this equation as a whole is not, since the factor $d\mathcal{L}/dh_j^t$ still includes explicit recurrences. E-prop will simply ignore these recurrences to solve this problem (cf. Fig. 5B, Section 4).

3.2 Re-expressing Explicit Recurrences of Order 1

Now we re-express the explicit recurrences' gradient $\partial c_k^{t'+1}/\partial h_j^t$ analogously to the implicit recurrence in the previous section. First, we plug Eq. 2 (explicit recurrences) into Eq. 10 (re-expressed implicit recurrence):

$$\begin{aligned} \frac{d\mathcal{L}}{dw_{ij}} &= \sum_{t'} \left(\frac{\partial \mathcal{L}}{\partial h_j^{t'}} + \sum_k \frac{d\mathcal{L}}{dc_k^{t'+1}} \frac{\partial c_k^{t'+1}}{\partial h_j^{t'}} \right) e_{ij}^{t'} \\ &= \sum_{t'} \frac{\partial \mathcal{L}}{\partial h_j^{t'}} e_{ij}^{t'} + \sum_k \sum_{t'} \frac{d\mathcal{L}}{dc_k^{t'+1}} \frac{\partial c_k^{t'+1}}{\partial h_j^{t'}} e_{ij}^{t'} \end{aligned} \quad (11)$$

The first factor of this sum is already causal since it only requires the direct derivative and the implicit eligibility trace introduced in Eq. 7. Focusing on the second factor, this term represents the gradient until $c_k^{t'+1}$, the jump to $h_j^{t'}$ and the implicit eligibility trace $e_{ij}^{t'}$ stored there that represents the sum over all of the paths from there to w_{ij} .

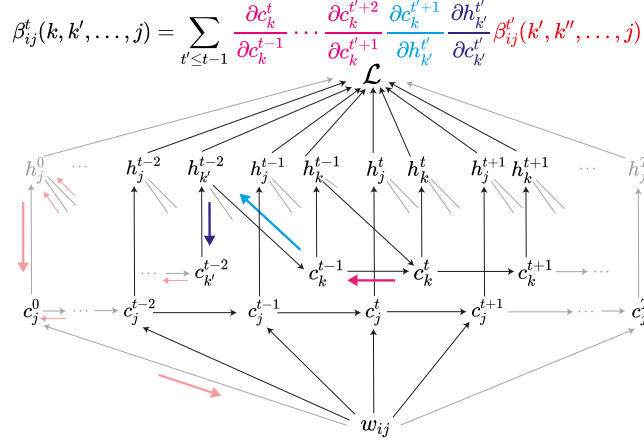


Fig. 6. Computational graph for the explicit variable $\beta_{ij}^t(k, k', \dots, j)$ with $t' = t - 1$.

Unrolling the recursion: We can now unroll the recursion by plugging the equation of explicit recurrences Eq. 2 into the second term of Eq. 11:

$$\begin{aligned}
\sum_{t'} \frac{d\mathcal{L}}{dc_k^{t'+1}} \frac{\partial c_k^{t'+1}}{\partial h_j^{t'}} e_{ij}^{t'} &= \sum_{t'} \left(\frac{d\mathcal{L}}{dh_k^{t'+1}} \frac{\partial h_k^{t'+1}}{\partial c_k^{t'+1}} + \frac{d\mathcal{L}}{dc_k^{t'+2}} \frac{\partial c_k^{t'+2}}{\partial c_k^{t'+1}} \right) \frac{\partial c_k^{t'+1}}{\partial h_j^{t'}} e_{ij}^{t'} \\
&= \sum_{t'} \left(\frac{d\mathcal{L}}{dh_k^{t'+1}} \frac{\partial h_k^{t'+1}}{\partial c_k^{t'+1}} + \left(\frac{d\mathcal{L}}{dh_k^{t'+2}} \frac{\partial h_k^{t'+2}}{\partial c_k^{t'+2}} + (\dots) \frac{\partial c_k^{t'+3}}{\partial c_k^{t'+2}} \right) \frac{\partial c_k^{t'+2}}{\partial c_k^{t'+1}} \right) \frac{\partial c_k^{t'+1}}{\partial h_j^{t'}} e_{ij}^{t'} \\
&= \sum_{t'} \sum_{t \geq t'} \frac{d\mathcal{L}}{dh_k^{t+1}} \frac{\partial h_k^{t+1}}{\partial c_k^{t+1}} \frac{\partial c_k^{t+1}}{\partial c_k^t} \dots \frac{\partial c_k^{t+2}}{\partial c_k^{t+1}} \frac{\partial c_k^{t+1}}{\partial h_j^{t'}} e_{ij}^{t'} \quad (12)
\end{aligned}$$

Flip time indices: We flip the indices again to have a causal formula:

$$\sum_{t'} \frac{d\mathcal{L}}{dc_k^{t'+1}} \frac{\partial c_k^{t'+1}}{\partial h_j^{t'}} e_{ij}^{t'} = \sum_t \frac{d\mathcal{L}}{dh_k^{t+1}} \frac{\partial h_k^{t+1}}{\partial c_k^{t+1}} \sum_{t' \leq t} \frac{\partial c_k^{t+1}}{\partial c_k^t} \dots \frac{\partial c_k^{t'+2}}{\partial c_k^{t'+1}} \frac{\partial c_k^{t'+1}}{\partial h_j^{t'}} e_{ij}^{t'} \quad (13)$$

Definition (Explicit variable) We define the explicit variable $\beta_{ij}^t(k, k', \dots, j)$ as:

$$\beta_{ij}^t(k, k', \dots, j) := \sum_{t' \leq t-1} \frac{\partial c_k^t}{\partial c_k^{t'-1}} \cdots \frac{\partial c_k^{t'+2}}{\partial c_k^{t'+1}} \frac{\partial c_k^{t'+1}}{\partial h_{k'}^{t'}} \frac{\partial h_{k'}^{t'}}{\partial c_{k'}^{t'}} \beta_{ij}^{t'}(k', k'', \dots, j) \quad (14)$$

with $\beta_{ij}^t(j) = \epsilon_{ij}^t$.

Backwards interpretation: The explicit variable represents the idea of starting at c_k^t , moving an arbitrary number of steps through the implicit recurrence $c_k^t \rightarrow h_{k'}^{t-1}$ until at a certain t' you jump to the output variable of another neuron $h_{k'}^{t'}$, down to its hidden variable $c_{k'}^{t'}$ and then start again, with a path, now starting at

$c_{k'}^{t'}$. In total, it considers all possible paths, with arbitrary length, spending an arbitrary number of steps in each of the neurons (through implicit recurrences) from c_k^t to $c_j^{t'}$ through the neurons k', k'', \dots and then times the implicit variable $e_{ij}^{t'}$ (cf. Fig. 6).

Forwards interpretation: The explicit variable accounts for the influence of the activity of neuron j at any previous time step $c_j^{t'}$ to neuron k' at a future time step $c_{k'}^{t'}$ through the neurons k', k'', \dots

Incremental computation: The recursive relation to this variable that allows it to be updated at each time step is:

$$\beta_{ij}^t(k, k', \dots, j) = \frac{\partial c_k^t}{\partial c_{k'}^{t-1}} \beta_{ij}^{t-1}(k, k', \dots, j) + \frac{\partial c_k^t}{\partial h_{k'}^{t-1}} \frac{\partial h_{k'}^{t-1}}{\partial c_{k'}^{t-1}} \beta_{ij}^{t-1}(k', \dots, j) \quad (15)$$

Definition (Explicit eligibility trace) Given the explicit variable $\beta_{ij}^t(k, k', \dots, j)$, we define the explicit eligibility trace $b_{ij}^t(k, k', \dots, j)$ as:

$$b_{ij}^t(k, k', \dots, j) := \frac{\partial h_k^t}{\partial c_k^t} \beta_{ij}^t(k, k', \dots, j) \quad (16)$$

with $b_{ij}^t(j) = e_{ij}^t$

Since $\partial h_k^t / \partial c_k^t$ is causal and local, and the explicit variable $\beta_{ij}^t(k, k', \dots, j)$ is causal but only partially local (it requires message passing from the presynaptic neuron k' to the postsynaptic neuron k), then the explicit eligibility trace $b_{ij}^t(k, k', \dots, j)$ is also **causal** but only **partially local**.

Final equation with re-expressed explicit recurrence of order 1: Substituting the explicit variable Eq. 14 in Eq. 13 yields:

$$\sum_k \sum_t \frac{d\mathcal{L}}{dc_k^{t+1}} \frac{\partial c_k^{t+1}}{\partial h_j^t} e_{ij}^{t'} = \sum_k \sum_t \frac{d\mathcal{L}}{dh_k^{t+1}} b_{ij}^{t+1}(k, j) \quad (17)$$

And substituting this back to the original equation (cf. Eq. 11):

$$\begin{aligned} \frac{d\mathcal{L}}{dw_{ij}} &= \sum_t \frac{\partial \mathcal{L}}{\partial h_j^t} e_{ij}^t + \sum_k \sum_t \frac{d\mathcal{L}}{dc_k^{t+1}} \frac{\partial c_k^{t+1}}{\partial h_j^t} e_{ij}^t \\ &= \sum_t \frac{\partial \mathcal{L}}{\partial h_j^t} e_{ij}^t + \sum_k \sum_t \frac{d\mathcal{L}}{dh_k^{t+1}} b_{ij}^{t+1}(k, j) \end{aligned} \quad (18)$$

Here it becomes clear how setting this second factor to 0 is what gives us e-prop (cf. the right arrow in Fig. 1), since we forcefully ignore the influence of a neuron on other neurons (and itself) through the explicit recurrences.

3.3 Re-expressing Explicit Recurrences of Order > 1

Now that we have seen how the explicit eligibility connects the activity of neuron j with other neurons through explicit recurrences, we can use it to re-express higher-order explicit recurrences.

Unroll the recursion: Starting from the equation with one order of explicit recurrence already re-expressed (cf. Eq. 18), and alternatively using the definition of explicit recurrences (cf. Eq. 2) and the action of the explicit eligibility trace (cf. Eq. 17), we can repeat the previous steps for higher orders:

$$\begin{aligned}
\frac{d\mathcal{L}}{dw_{ij}} &= \sum_t \frac{\partial \mathcal{L}}{\partial h_j^t} e_{ij}^t + \sum_{k_1} \sum_t \left(\frac{\partial \mathcal{L}}{\partial h_{k_1}^{t+1}} + \sum_{k_2} \frac{d\mathcal{L}}{dc_{k_2}^{t+2}} \frac{\partial c_{k_2}^{t+2}}{\partial h_{k_1}^{t+1}} \right) b_{ij}^{t+1}(k_1, j) \\
&= \sum_t \frac{\partial \mathcal{L}}{\partial h_j^t} e_{ij}^t + \sum_t \sum_{k_1} \frac{\partial \mathcal{L}}{\partial h_{k_1}^{t+1}} b_{ij}^{t+1}(k_1, j) + \sum_t \sum_{k_1, k_2} \frac{d\mathcal{L}}{dc_{k_2}^{t+2}} \frac{\partial c_{k_2}^{t+2}}{\partial h_{k_1}^{t+1}} b_{ij}^{t+1}(k_1, j) \\
&= \sum_t \frac{\partial \mathcal{L}}{\partial h_j^t} e_{ij}^t + \sum_t \sum_{k_1} \frac{\partial \mathcal{L}}{\partial h_{k_1}^{t+1}} b_{ij}^{t+1}(k_1, j) + \sum_t \sum_{k_1, k_2} \frac{d\mathcal{L}}{dh_{k_2}^{t+2}} b_{ij}^{t+2}(k_2, k_1, j) \\
&= \sum_t \sum_{t' \geq t} \sum_{k_0=j, k_1, \dots, k_{t'}} \frac{\partial \mathcal{L}}{\partial h_{k_t}^{t'}} b_{ij}^{t'}(k_t, \dots, k_1, k_0 = j) \tag{19}
\end{aligned}$$

This gives us a high overview of separating the different levels of explicit recurrences which will lead to the definition of the m-order e-prop (Section 4).

Flip time indices: As before we change the time indices and reorganise to allow for causality,

$$\frac{d\mathcal{L}}{dw_{ij}} = \sum_t \sum_k \frac{\partial \mathcal{L}}{\partial h_k^t} \frac{\partial h_k^t}{\partial c_k^t} \sum_{t' \leq t} \sum_{k_0=j, k_1, \dots, k_{t'-1}} \beta_{ij}^t(k, k_{t'-1}, \dots, k_1, k_0 = j) \tag{20}$$

Definition (Recurrence variable) We define the recurrence variable $\alpha_{ij}^{t,r}$ as:

$$\alpha_{ij}^{t,r} = \sum_{t' \leq t} \sum_{k_0=j, k_1, \dots, k_{t'-1}} \beta_{ij}^t(r, k_{t'-1}, \dots, k_1, k_0 = j) \tag{21}$$

Backwards interpretation: Starting at current time t in neuron r , the recurrence variable represents all combinations of paths through any combination of neurons $k_{t'-1}, \dots, k_1$ ending in neuron j .

Forwards interpretation: The recurrence variable accounts for the influence of the activity of neuron j at any previous time step to neuron r at the current timestep t through all possible paths through all neurons.

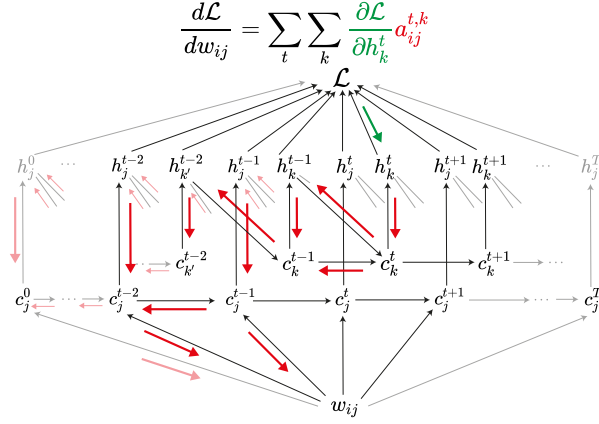


Fig. 7. Computational graph for final computation of RTRL.

Incremental computation: Once again, importantly, we have a recursive equation for computing the recurrence variable:

$$\alpha_{ij}^{t,r} = \frac{\partial c_r^t}{\partial c_r^{t-1}} \alpha_{ij}^{t-1,r} + \sum_k \frac{\partial c_r^t}{\partial h_k^{t-1}} \frac{\partial h_k^{t-1}}{\partial c_k^{t-1}} \alpha_{ij}^{t-1,k} \quad (22)$$

Definition (Recurrence eligibility trace) Given the recurrence variable $\alpha_{ij}^{t,r}$, we define the recurrence eligibility trace $a_{ij}^{t,r}$ as:

$$a_{ij}^{t,r} := \frac{\partial h_r^t}{\partial c_r^t} \alpha_{ij}^{t,r} \quad (23)$$

Since $\partial h_r^t / \partial c_r^t$ is causal and local, but the recurrence variable $\alpha_{ij}^{t,r}$ is causal but non-local, the recurrence eligibility trace $a_{ij}^{t,r}$ is also **causal** but **non-local**. It is non-local in an equivalent way as BPTT is not: each synapse ij requires to store a variable representing how the activation in the past of any other neuron r would affect its computation in the present, even if $r \neq i, j$ and through all possible paths of synapses. The recursive computation of $\alpha_{ij}^{t,r}$ requires of the summation of the recurrence variables of all the neurons requiring non-local communication.

Final equation of RTRL: Eq. 20 transforms into (by substituting Eq. 21 into Eq. 20) the final equation for RTRL (cf. Fig. 7):

$$\frac{d\mathcal{L}}{dw_{ij}} = \sum_t \sum_k \frac{\partial \mathcal{L}}{\partial h_k^t} \frac{\partial h_k^t}{\partial c_k^t} \alpha_{ij}^{t,k} = \sum_t \sum_k \frac{\partial \mathcal{L}}{\partial h_k^t} a_{ij}^{t,k} \quad (24)$$

We now have a causal but still non-local gradient computation algorithm.

4 E-prop

The e-prop algorithm approximates the gradient by not considering the explicit recurrences in RNNs. E-prop was originally formulated for RSNNs, since it is

considered more biologically plausible than BPTT and RTRL due to its characteristics of being causal and local [2]. The approximation of the gradient that defines e-prop is:

$$\frac{d\mathcal{L}}{dw_{ij}} \approx \sum_t \frac{\partial \mathcal{L}}{\partial h_j^t} e_{ij}^t \quad (25)$$

Through the derivation of RTRL from BPTT, e-prop has arisen naturally in three different places. This allows us for equivalent interpretations of the approximation, each more detailed than the previous one.

First, and as originally proposed [2], we can understand e-prop from the equation that arises after re-expressing the implicit eligibility trace (cf. Eq. 10):

$$\frac{d\mathcal{L}}{dw_{ij}} = \sum_t \frac{d\mathcal{L}}{dh_j^t} e_{ij}^t$$

Here we approximate the non-causal and non-local total derivative by the causal and local partial derivative, i.e. $d\mathcal{L}/dh_j^t \approx \partial\mathcal{L}/\partial h_j^t$ (cf. Fig. 5).

Second, we can understand it from the equation after re-expressing the explicit recurrences of order 1 (cf. Eq. 18):

$$\frac{d\mathcal{L}}{dw_{ij}} = \sum_t \frac{\partial \mathcal{L}}{\partial h_j^t} e_{ij}^t + \sum_k \sum_t \frac{d\mathcal{L}}{dh_k^{t+1}} b_{ij}^{t+1}(k, j)$$

Here we see explicitly what we are ignoring in the approximation, since ignoring this non-causal and non-local second term, i.e. $\sum_k \sum_t \frac{d\mathcal{L}}{dh_k^{t+1}} b_{ij}^{t+1}(k, j) = 0$, defines e-prop. Ignoring these future dependencies to other neurons through explicit recurrences leads to a gradient computing algorithm that treats each neuron as producing an output only for the network's computation and not to communicate to other neurons. Therefore, synapses arriving at neurons that are not directly connected to the readout of the RNN, are not modified by e-prop (e-prop does not compute through additional feed-forward layers).

Finally, the most expressive of the interpretations comes from the equation that shows how to apply the re-expressing of the explicit recurrences of order 1, recursively, to re-express higher orders (cf. Eq. 19):

$$\begin{aligned} \frac{d\mathcal{L}}{dw_{ij}} &= \sum_t \frac{\partial \mathcal{L}}{\partial h_j^t} e_{ij}^t + \sum_k \sum_t \frac{d\mathcal{L}}{dh_k^{t+1}} b_{ij}^{t+1}(k, j) \\ &= \sum_t \frac{\partial \mathcal{L}}{\partial h_j^t} e_{ij}^t + \sum_t \sum_{k_1} \frac{\partial \mathcal{L}}{\partial h_{k_1}^{t+1}} b_{ij}^{t+1}(k_1, j) + \sum_t \sum_{k_1, k_2} \frac{d\mathcal{L}}{dh_{k_2}^{t+2}} b_{ij}^{t+2}(k_2, k_1, j) \\ &= \dots \end{aligned}$$

Here we define the *m-order e-prop* as the approximation resulting from setting in the above equation $\sum_t \sum_{k_0=j, k_1, \dots, k_m} \frac{d\mathcal{L}}{dh_{k_m}^{t+m}} b_{ij}^{t+m}(k_m, \dots, k_1, k_0 = j) = 0$. By increasing the order m we better approximate the gradient at the cost of needing the activities of other neurons m time steps ahead to compute the current gradient of the loss. Under this scope, standard e-prop [2] is just the 1-order e-prop (fully causal and local but the most inaccurate approximation). On the

other extreme, the T-order e-prop (nothing is approximated or set to 0) corresponds to the full gradient computation, in a middle form between BPTT and RTRL (the exact computation of the gradient but completely non-causal and non-local). Moreover, synapses arriving into neurons connected to the readout through up to $m - 1$ synapses will be modified by the m-order e-prop (m-order e-prop computes through up to $m - 1$ additional feed-forward layers).

5 Conclusion

In this paper, we formally explored how BPTT, RTRL, and e-prop relate to each other. We extended the general scheme for re-expressing recurrences as eligibility traces from [2] and applied it iteratively to go from BPTT to RTRL. In the process, we found intermediate expressions that allow for better intuition of these algorithms. Moreover, we showed how e-prop can be seen as an extreme case of a series of approximation algorithms, which we coin m-order e-prop.

References

1. Bellec, G., Salaj, D., Subramoney, A., Legenstein, R., Maass, W.: Long short-term memory and learning-to-learn in networks of spiking neurons (2018)
2. Bellec, G., Scherr, F., Subramoney, A., Hajek, E., Salaj, D., Legenstein, R., Maass, W.: A solution to the learning dilemma for recurrent networks of spiking neurons. *Nature Communications* **11** (07 2020)
3. Czarnecki, W.M., Świrszcz, G., Jaderberg, M., Osindero, S., Vinyals, O., Kavukcuoglu, K.: Understanding synthetic gradients and decoupled neural interfaces (2017)
4. Hochreiter, S., Schmidhuber, J.: Long short-term memory. *Neural computation* **9**(8), 1735–1780 (1997)
5. Marblestone, A., Wayne, G., Kording, K.: Toward an integration of deep learning and neuroscience. *Frontiers in Computational Neuroscience* **10** (06 2016)
6. Werbos, P.: Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE* **78**, 1550 – 1560 (11 1990)
7. Werbos, P., John, P.: Beyond regression : new tools for prediction and analysis in the behavioral sciences / (01 1974)
8. Williams, R.J., Zipser, D.: A learning algorithm for continually running fully recurrent neural networks. *Neural Computation* **1**, 270–280 (1989)
9. Zenke, F., Neftci, E.O.: Brain-inspired learning on neuromorphic substrates. *CoRR abs/2010.11931* (2020)