$\mathrm{BP}(\lambda)$: BIAS-FREE ONLINE LEARNING VIA SYNTHETIC GRADIENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training recurrent neural networks typically relies on backpropagation through time (BPTT). BPTT depends on forward and backward passes to be completed, rendering the network locked to these computations before loss gradients are available. Recently, Jaderberg et al. proposed synthetic gradients to alleviate the need for full BPTT. In their implementation synthetic gradients are learned through a mixture of backpropagated gradients and bootstrapped synthetic gradients, analogous to the temporal difference (TD) algorithm in Reinforcement Learning (RL). However, as in TD learning, heavy use of bootstrapping can result in bias which leads to poor synthetic gradient estimates. Inspired by the accumulate $\mathrm{TD}(\lambda)$ in RL, we propose a fully online method for learning synthetic gradients which avoids the use of BPTT altogether: *accumulate $BP(\lambda)$*. As in accumulate $\mathrm{TD}(\lambda)$, we show analytically that accumulate $\mathrm{BP}(\lambda)$ can control the level of bias by using a mixture of temporal difference errors and recursively defined eligibility traces. We next demonstrate empirically that our model outperforms the original implementation for learning synthetic gradients in a variety of tasks, and is particularly suited for capturing longer timescales. Finally, building on recent work we reflect on accumulate $\mathrm{BP}(\lambda)$ as a principle for learning in biological circuits. In summary, inspired by RL principles we introduce an algorithm capable of bias-free online learning via synthetic gradients.

1 INTRODUCTION

A common approach for solving temporal tasks is to use recurrent neural networks (RNNs), which with the right parameters can effectively integrate and maintain information over time. The temporal distance between inputs and subsequent task loss, however, can make optimising these parameters challenging. The backpropagation through time (BPTT) algorithm is the classical solution to this problem that is applied once the task is complete and all task losses are propagated backwards in time through the preceding chain of computation. Exact loss gradients are thereby derived and are used to guide updates to the network parameters.

However, BPTT can be undesirably expensive to perform, with its memory and computational requirements scaling intractably with the task duration. Moreover, the gradients can only be obtained after the RNN forward and backward passes have been completed. This makes the network parameters effectively locked until those computations are carried out. One common solution to alleviate these issues is to apply truncated BPTT, where error gradients are only backpropagated within fixed truncation windows, but this approach can limit the network's ability to capture long-range temporal dependencies.

One proposed method which avoids the need for many-step BPTT whilst still capturing long-range dependencies is to apply synthetic gradients (Jaderberg et al., 2017; Czarnecki et al., 2017). In this method gradients from future errors are predicted by a separate network, a "synthesiser", given the current RNN activity (Fig. 1**a**). Synthetic gradients enable the network to model long-range dependencies on future errors whilst avoiding the waiting time imposed by BPTT. The independence of memory and computational complexity with respect to the total task length makes synthetic gradients an attractive alternative compared to BPTT (Marschall et al., 2020). Recently, these properties have led neuroscientists to speculate that synthetic gradients are computed at the systems-level in

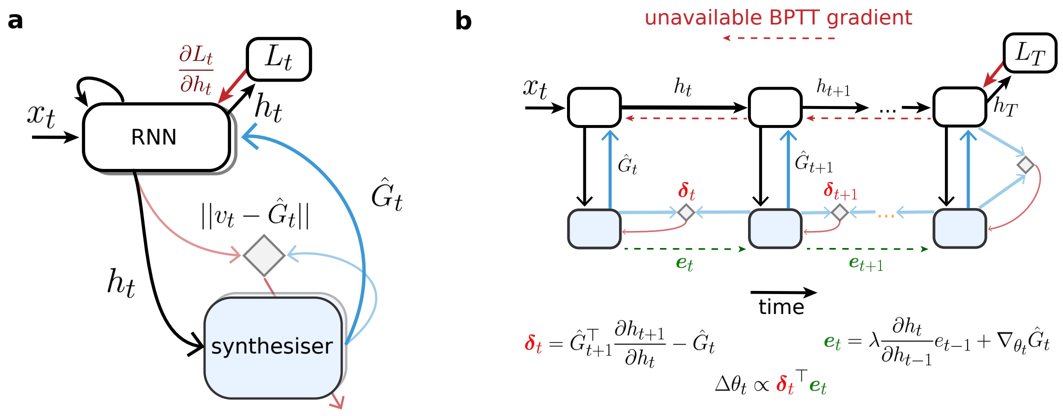
a
x_t
RNN
$\frac{\partial L_t}{\partial h_t}$
L_t
h_t
$||v_t - \hat{G}_t||$
$\hat{G}_t$
h_t
synthesiser
b
unavailable BPTT gradient
x_t
h_t
h_{t+1}
L_T
h_T
$\hat{G}_t$
$\hat{G}_{t+1}$
δ_t
δ_{t+1}
e_t
e_{t+1}
time
$\delta_t = \hat{G}_{t+1}^\top \frac{\partial h_{t+1}}{\partial h_t} - \hat{G}_t$
$e_t = \lambda \frac{\partial h_t}{\partial h_{t-1}} e_{t-1} + \nabla_{\theta_t} \hat{G}_t$
$\Delta\theta_t \propto \delta_t{}^\top e_t$

Figure 1: **Schematic of a recurrent neural network (RNN) which learns via synthetic gradients**. (**a**) External input x_t is provided to the RNN which has hidden state h_t. Due to recurrency this state will affect the task loss at the current timestep L_t and future timesteps $L_{>t}$ not yet seen. A distinct synthesiser network receives h_t as input and estimates its future loss gradient $\hat{G}_t \approx \frac{\partial L_{>t}}{\partial h_t}$, which is provided to the RNN for learning. The synthesiser learns to mimic a target gradient v_t. How v_t is defined and learned is the focus of this paper. (**b**) An illustration of the accumulate $\text{BP}(\lambda)$ algorithm for learning synthetic gradients in an unrolled version of the network. Current activity h_t must be correctly associated to the later task loss L_T. Here the parameters θ of the synthesiser are updated via a mixture of temporal difference errors δ (red) and eligibility traces e (green). As in accumulate $\text{TD}(\lambda)$ in RL, δ is computed online using bootstrapping whilst e propagates forwards with a decay component λ with $0 \leq \lambda \leq 1$. Together, they approximate the true loss gradient. In contrast to the original synthetic gradient algorithm by Jaderberg et al. (2017), our model does not require BPTT.

the brain, explaining a range of experimental observations (Marschall et al., 2019; Pemberton et al., 2021; Boven et al., 2023).

Despite their promise, the full potential of approximating BPTT with synthetic gradients has not yet been realised. In particular, it is not yet clear what are the optimal conditions for learning synthetic gradients. In its original implementation, Jaderberg et al. use synthetic gradients alongside truncated BPTT, and define the synthesiser target as a mixture of backpropagated gradients with its own predicted (future) gradient. That is, the synthesiser uses its own estimations – *bootstrapping* – for learning. As the original authors note, this is highly reminiscent of temporal difference (TD) algorithms used in Reinforcement Learning (RL) which use bootstrapping for estimating the future return (Sutton & Barto, 2018). Indeed, in their supplementary material Jaderberg et al. extend this analogy and introduce the notion of the λ-weighted synthetic gradient, which is analogous to the λ-return in RL. However, λ-weighted synthetic gradients were only presented conceptually and it remained unclear whether they would be of practical benefits as they still require BPTT.

In this study, inspired by established RL theory, we make conceptual and experimental advancements on λ-weighted synthetic gradients. In particular, we propose an algorithm for learning synthetic gradients, *accumulate* $BP(\lambda)$, which mirrors the accumulate $\text{TD}(\lambda)$ algorithm in RL (Van Seijen et al., 2016). Just as how accumulate $\text{TD}(\lambda)$ provides an online solution to learning the λ-return in RL, we show that accumulate $\text{BP}(\lambda)$ provides an online solution to learning λ-weighted synthetic gradients. The algorithm uses forward-propagating eligibility traces and has the advantage of not requiring (even truncated) BPTT at all. Moreover, we demonstrate that accumulate $\text{BP}(\lambda)$ can alleviate the bias involved in directly learning bootstrapped estimations as suffered in the original implementation (Jaderberg et al., 2017).

We now provide a brief background into the application of synthetic gradients for RNN learning. We then introduce the accumulate $\text{BP}(\lambda)$ algorithm and demonstrate both analytically and empirically can it alleviates the problem of bias suffered in the original implementation. Next, we touch upon accumulate $\text{BP}(\lambda)$ as a mechanism for learning in biological circuits. Finally, we discuss the limitations and conclusions of our work.

2 Background

	Reinforcement Learning	Synthetic Gradients
G_t	$\sum_{\tau \geq 1} \gamma^{\tau-1} R_{t+\tau}$	$\sum_{\tau \geq 1} \gamma^{\tau-1} \frac{\partial L_{t+\tau}}{\partial h_t}$
$\hat{G}_t$	$V(\phi_t; \theta)$	$g(h_t; \theta)$
$G_t^{(n)}$	$\sum_{\tau=1}^{n} \gamma^{\tau-1} R_{t+\tau} + \gamma^n \hat{G}_{t+n}$	$\sum_{\tau=1}^{n} \gamma^{\tau-1} \frac{\partial L_{t+\tau}}{\partial h_t} + \gamma^n \hat{G}_{t+n}^\top \frac{\partial h_{t+n}}{\partial h_t}$
G_t^λ	$(1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t$	

Table 1: Relationship between terms used in value estimation in RL and synthetic gradient (SG) estimation in supervised learning. For RL R_t denotes the reward, ϕ_t the representation of the current state, and V the value function for future return. For main text for more details.

2.1 Synthetic gradients for supervised learning

Consider an RNN with free parameters Ψ performing a task of sequence length T (which may be arbitrarily long). At time t RNN dynamics follow $h_t = f(x_t, h_{t-1}; \Psi)$, where h is the RNN hidden state, x is the input, and f is the RNN computation. Let $L_t = \mathcal{L}(\hat{y}_t, y_t)$ denote the loss at time t, where $\hat{y}_t$ is the RNN-dependent prediction, y_t is the desired target, and $\mathcal{L}$ the task loss function (e.g. mean-squared error). Let $L_{>t} = \sum_{t<\tau\leq T} L_\tau$ denote the total loss strictly after timestep t.

During training, we wish to update Ψ to minimise all losses from timestep t onwards. Using gradient descent this is achieved with $\Psi = \Psi - \eta \frac{\partial \sum_{t\leq\tau\leq T} L_\tau}{\partial \Psi}$ for some RNN learning rate η. We can write this gradient as

$$\frac{\partial \sum_{t\leq\tau\leq T} L_\tau}{\partial \Psi} = \frac{\partial (L_t + L_{>t})}{\partial \Psi} \tag{1}$$

$$= \frac{\partial L_t}{\partial \Psi} + \frac{\partial L_{>t}}{\partial \Psi} \tag{2}$$

$$= \left(\frac{\partial L_t}{\partial h_t} + \frac{\partial L_{>t}}{\partial h_t} \right) \frac{\partial h_t}{\partial \Psi}. \tag{3}$$

Whilst the terms $\frac{\partial L_t}{\partial h_t}$ and $\frac{\partial h_t}{\partial \Psi}$ above are relatively easy to compute and available at timestep t, $\frac{\partial L_{>t}}{\partial h_t}$ can present challenges. Without BPTT, this term is simply taken as zero, $\frac{\partial L_{>t}}{\partial h_t} = 0$, and future errors are effectively ignored. With BPTT, this term is only computed after the forward pass and the corresponding backward pass so that all future errors are observed and appropriately backpropagated. This has memory and computational complexity which scales with T, and thereby relies on arbitrarily long waits before the loss gradient is available.

The aim of synthetic gradients is to provide an immediate prediction $\hat{G}_t$ of the future loss gradient, $\hat{G}_t \approx G_t := \frac{\partial L_{>t}}{\partial h_t}$ (Jaderberg et al., 2017). We use notation G_t both to represent "gradient" but also to highlight its resemblance to the return in RL. Note that this gradient is a vector of the same size as h. As in Jaderberg et al. (2017), we consider $\hat{G}_t$ to be computed via a separate "synthesiser" network: $\hat{G}_t = g(h_t; \theta)$, where g denotes the synthesiser computation with parameters θ. An approximation for the loss gradient with respect to the RNN parameters can then be written as

$$\frac{\partial \sum_{t\leq\tau\leq T} L_\tau}{\partial \Psi} \approx \left(\frac{\partial L_t}{\partial h_t} + \hat{G}_t \right) \frac{\partial h_t}{\partial \Psi}. \tag{4}$$

This is available at timestep t and removes the dependence of the memory and computational complexity on T as in Eq. 3.

2.2 Learning synthetic gradients

The quality of the synthetic gradient used in Eq. 4 is critical for enabling good RNN parameter updates. However, how the synthesiser parameters themselves (θ) should be learned remains relatively unexplored and is the focus of this paper. One option is to directly minimise the distance between the synthesiser prediction and true backpropagated loss gradient; that is, minimise $\left\| G_t - \hat{G}_t \right\|_2$ where $\|.\|_2$ denotes the Euclidean norm. However, deriving G_t relies on full backpropagation and the synthesiser network now suffers from the exact locking constraints we wish to avoid.

To alleviate this problem Jaderberg et al. proposed a creative solution based on bootstrapping. In particular, they formulated a target gradient for the synthesiser which uses backpropagated gradients for nearby losses and the synthesiser's own prediction for losses later on in the task sequence. Specifically, given some truncation size n, the synthesiser parameters are optimised to minimise $\left\| G_t^{(n)} - \hat{G}_t \right\|_2$, where $G_t^{(n)}$ is defined as

$$G_t^{(n)} = \sum_{t<\tau\leq t+n} \gamma^{\tau-t-1} \frac{\partial L_\tau}{\partial h_t} + \gamma^n \hat{G}_{t+n} \frac{\partial h_{t+n}}{\partial h_t}, \tag{5}$$

where γ is some fixed discount factor with $0 \leq \gamma \leq 1$ which weighs down future loss gradients. In its original implementation all loss gradients were weighted equally such that $\gamma = 1$, but we find in our experiments that some degree of future discount such that $\gamma < 1$ is necessary. Note that the left term side of Eq. 5 is derived using BPTT with truncation size n, whilst the right term is the bootstrapped term which models error gradients beyond the truncation horizon. Interestingly, $G_t^{(n)}$ can be mapped to the n-step return in Reinforcement Learning (see Table 1), and we accordingly refer to this term as the n-step synthetic gradient.

Learning synthetic gradients using Eq. 5 has been shown to be effective for various temporally challenging tasks (Jaderberg et al., 2017). However, just as in RL, choosing an appropriate truncation size can be challenging. Specifically, whilst small truncation sizes n will reduce waiting times and the complexity of deriving $G_t^{(n)}$, small n may lead to an overly strong dependency on the boostrapped term such that the synthesiser suffers from bias in its estimation. This is akin to the bias often obtained by TD learning algorithms in RL (Sutton & Barto, 2018).

3 ACCUMULATE BP(λ)

To correct for bias in the synthesiser predictions, whilst simultaneously not requiring truncated BPTT, we build on well established RL theory. Specifically, inspired by the TD(λ) in RL, we formulate an algorithm for learning synthetic gradients – accumulate BP(λ) – which has the advantage of reduced bias compared to the original n-step implementation.

Like TD(λ), the core idea of accumulate BP(λ) is that learning can be improved by employing *eligibility traces* in the synthesiser parameters θ (dotted green arrow in Fig. 1**b**). Specifically, let λ be such that $0 \leq \lambda \leq 1$. At timestep t we compute an eligibility trace vector e_t which is defined[1] recursively according to

$$e_t = \gamma\lambda \frac{\partial h_t}{\partial h_{t-1}} e_{t-1} + \nabla_{\theta_t} g(h_t; \theta_t), \tag{6}$$

where e_0 is defined as the zero vector, $e_0 = 0$. Note that this is the same as the eligibility trace update defined in accumulate TD except that g maps onto a vector space (as opposed to scalar rewards) and the Jacobian $\frac{\partial h_t}{\partial h_{t-1}}$ is incorporated to model the temporal dependency between RNN states. We then define the temporal difference error δ_t at timestep t (red arrow in Fig. 1**b**). This is computed as the difference between the synthesiser prediction and the 1-step synthetic gradient $G_t^{(1)}$ as follows

$$\delta_t = \frac{\partial L_{t+1}}{\partial h_t} + \gamma g(h_{t+1}; \theta_t)^\top \frac{\partial h_{t+1}}{\partial h_t} - g(h_t; \theta_t). \tag{7}$$

[1]Note that there is some abuse of notation with respect to the matrix multiplication operations defined in Eqs. 8 and 6. If $\text{in}_\theta, \text{out}_\theta$ are the sizes of the input and output dimension of θ, respectively, then the eligibility trace e_t is a three-dimensional tensor of shape $(|h|, \text{out}_\theta, \text{in}_\theta)$, where $|h|$ is size of h. To compute the matrix product Ae_t for a matrix A of shape $(r, |h|)$ we concatenate the latter two dimensions of e_t so that is of shape $(|h|, \text{out}_\theta \times \text{in}_\theta)$ and once the product is computed reshape it as $(r, \text{out}_\theta, \text{in}_\theta)$. Note that if $r = 1$ (as in Eq. 8) then the first dimension is removed.

Algorithm 1 RNN learning with accumulate $\mathrm{BP}(\lambda)$. Updates RNN parameters using estimated gradients provided by synthesiser function g.

```
Require: $\Psi_0, \theta_0, \{(x_t, y_t)\}_{1\le t\le T}, \eta, \alpha, \gamma, \lambda$
  $\Psi \leftarrow \Psi_0$                                  {init. RNN parameters}
  $\theta \leftarrow \theta_0$                                  {init. synthesiser parameters}
  $h, \partial_h, e \leftarrow 0$                             {init. RNN state, Jacobian, and elig. trace}
  for $t = 1$ to $T$ do
    $e \leftarrow \gamma\lambda\partial_h e + \nabla_\theta g(h;\theta)$          {update eligibility trace}
    $h' \leftarrow f(x_t, h; \Psi)$, $L \leftarrow \mathcal{L}(h', y_t)$         {compute next hidden state and task loss}
    $\partial_h \leftarrow \frac{\partial h'}{\partial h}$, $\partial_L \leftarrow \frac{\partial L}{\partial h'}$, $\partial_\Psi \leftarrow \frac{\partial h'}{\partial \Psi}$      {compute local gradients}
    $\delta \leftarrow [\partial_L + \gamma g(h';\theta)]^\top \partial_h - g(h;\theta)$   {compute synthesiser TD error}
    $\Delta\theta \leftarrow \alpha\delta^\top e$                     {update synthesiser parameters}
    $\Delta\Psi \leftarrow \eta[\partial_L + g(h';\theta)]^\top \partial_\Psi$         {update RNN parameters}
    $h \leftarrow h'$                               {update RNN hidden state}
  end for
```

The update for the synthesiser parameters is then defined as

$$\theta_{t+1} = \theta_t + \alpha\delta_t^\top e_t. \tag{8}$$

Together, Eqs. 6-8 define the accumulate $\mathrm{BP}(\lambda)$ algorithm. An implementation whereby the RNN parameters are simultaneously updated with $\mathrm{BP}(\lambda)$ is provided in Algorithm 1.

The main analytical result in this paper shows that accumulate $\mathrm{BP}(\lambda)$ provides an efficient online method for learning λ-weighted synthetic gradients [2], which are λ-weighted mixtures of n-step synthetic gradients and are analogous to λ-returns in RL (see Theorem A.1 in Appendix). Formally, by applying accumulate $\mathrm{BP}(\lambda)$ the synthesiser approximately learns the λ-weighted synthetic gradient G_t^λ defined as

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_t^{(n)} + \lambda^{T-t-1}G_t. \tag{9}$$

Just as in the λ-return in RL, a higher choice of λ reduces the dependence on bootstrapping. For example, without eligibility traces (i.e. $\lambda = 0$, accumulate $\mathrm{BP}(0)$) the synthesiser learns the one-step synthetic gradient $G_t^{(1)}$ and arrives at the original implementation with truncation size $n = 1$ (Jaderberg et al., 2017). Conversely, when $\lambda = 1$ (accumulate $\mathrm{BP}(1)$), the synthesiser learns to approximate the fully backpropagated, unbiased error gradient G_t. Importantly, Eqs. 6 and 7 only consider gradients between variables of at most 1 timestep apart. In this respect, at least in the conventional sense, there is no use of BPTT.

4 Experiments

Next, we tested empirically the ability of accumulate $\mathrm{BP}(\lambda)$, which we henceforth simply call $\mathrm{BP}(\lambda)$, to produce good synthetic gradients and can drive effective RNN parameter updates. In all experiments we take the synthesiser computation g simply as a linear function of the RNN hidden state, $g(h_t;\theta) = \theta h_t$.

4.1 Approximating true error gradients in a toy task

We first analyse the alignment of $\mathrm{BP}(\lambda)$-derived synthetic gradients and true gradients derived by full BPTT, which we use to quantify the bias in synthesiser predictions. For this we consider a toy task in which a fixed (randomly connected) linear RNN receives a static input x_1 at timestep 1 and null input onwards, $x_t = 0$ for $t > 1$. To test the ability of $\mathrm{BP}(\lambda)$ to transfer error information across time the error is only defined at the last timestep L_T, where L_T is the mean-squared error

[2]The relatively small cost of accumulate $\mathrm{BP}(\lambda)$ is notable when contrasted with the online method for directly learning λ-weighted synthetic gradients – online λ-SG algorithm (akin to the online λ-return algorithm) – whose computational complexity scales intractably with the task length at T^2 (see Appendix section A).

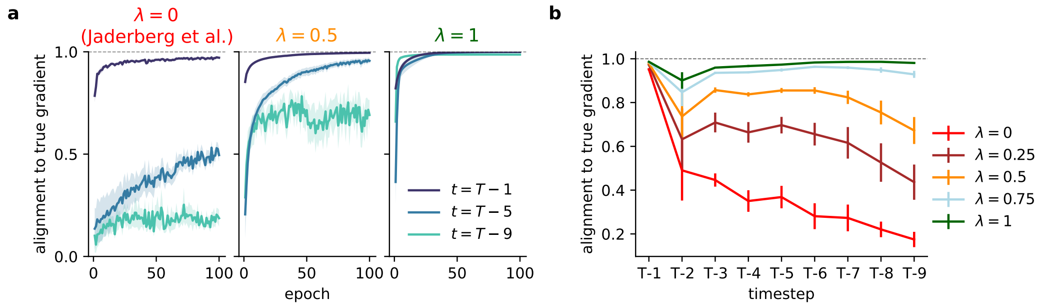

Figure 2: $\mathrm{BP}(\lambda)$ **approximations of BPTT gradients over time.** In this toy task, input is only provided at timestep 1 and the task target is only available at the end of the task at time $T = 10$. (**a**) Alignment between synthetic gradients and true gradients for a fixed RNN model across different timesteps within the task. Synthetic gradients are learned using (accumulate) $\mathrm{BP}(\lambda)$. Alignment is defined using the cosine similarity metric. (**b**) The average alignment over the last 10% of epochs in (a) across all timesteps.

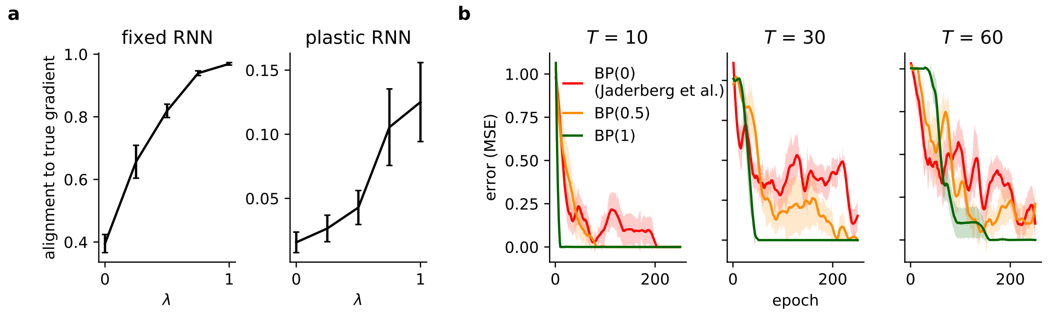

Figure 3: $\mathrm{BP}(\lambda)$ **drives better RNN learning in a toy task**. (**a**) Average cosine similarity between synthetic gradients and true gradients for fixed (left) and plastic (right) RNNs. Cosine similarity for plastic RNNs is taken over the first 5 training epochs, since this initial period is the key stage of learning (i.e. before the task is perfected). (**b**) Learning curves of RNNs which are updated using synthetic gradients derived by $\mathrm{BP}(\lambda)$ over different sequence lengths T. Results show average (with SEM) over 5 different initial conditions.

(MSE) between a two-dimensional target y_T and a linear readout of the final hidden activity h_T. We use a task length of $T = 10$ timesteps. Note that since the network is linear and the loss is a function of the MSE, the linear synthesiser should in principle be able to learn perfectly model the true BPTT gradient (Czarnecki et al., 2017).

As expected, we find that a high λ improves the alignment of synthetic gradients and true gradients compared to the $\lambda = 0$ case (i.e. $\mathrm{BP}(0)$ as in Jaderberg et al. (2017); Fig. 2**a**). Specifically, these results show, as predicted, that the heavy reliance on bootstrapping of $\mathrm{BP}(0)$ means that loss gradients near the end of the sequence must first be faithfully captured before earlier timesteps can be learned. When eligibility traces are applied in $\mathrm{BP}(1)$, however, the over-reliance on the bootstrapped estimate is drastically reduced and the synthesiser can very quickly learn faithful predictions across all timesteps. Indeed, we observe that high λ avoids the deterioration of synthetic gradient quality at earlier timesteps as suffered by $\mathrm{BP}(0)$ (Fig. 2**b**). We also observe that $\mathrm{BP}(\lambda)$ often outperforms n-step synthetic gradients (Fig. S1).

Next, to verify that $\mathrm{BP}(\lambda)$ is actually beneficial for RNN learning, we followed the scheme of Jaderberg et al. (2017) and applied RNN weight updates together with synthesiser weight updates (Algorithm 1). To ensure that the RNN needs to learn a temporal association (as opposed to a fixed readout), in this case we consider 3 input/target pairs and consider large sequence lengths T; we also now apply a $\tanh$ non-linearity to bound RNN activity.

Consistent with our results for the case of fixed RNNs, we observe that $\mathrm{BP}(\lambda)$ gives rise to better predictions for high λ when the RNN is learning (Fig. 3**a**). Notably, however, the alignment is weaker for plastic RNNs when compared to fixed RNNs. This is because of the differences lister above which make it harder for the synthesiser, since changes to the RNN will affect its error gradients and therefore lead to a moving synthesiser target. Nonetheless, we find that $\mathrm{BP}(\lambda)$ improves RNN learning even in the presence of relatively long temporal credit assignment (Fig. 3**b**). Next, we contrasted the ability of both $\mathrm{BP}(\lambda)$ and n-step methods to achieve near-zero error. Our results show that $\mathrm{BP}(1)$ is able to solve tasks more than double the length of those solved by the next best model (Table 2 and Fig. S2).

	BPTT				BPTT + SG				no BPTT			
	n=2	n=3	n=4	n=5	n=2	n=3	n=4	n=5	n=1	BP(0)	BP(0.5)	BP(1)
toy task	16	10	20	8	36	38	24	20	4	16	32	**90**
sequential MNIST	73.4	78.4	82.0	87.1	78.2	83.0	86.5	90.4	64.1	69.6	76.6	**90.6**
copy-repeat	9.0	9.0	12.0	15.0	12.0	9.0	15.0	23.0	8.2	9.0	15.8	**29.0**

Table 2: **Overview of model performance for sequential MNIST and copy-repeat tasks**. Results for toy and copy-repeat tasks shows the average task sequence length solved by the models (see text). Results for the sequential MNIST task show the average test accuracy as a percentage after training. Values denote average over 5 different initial conditions.

4.2 Sequential MNIST task

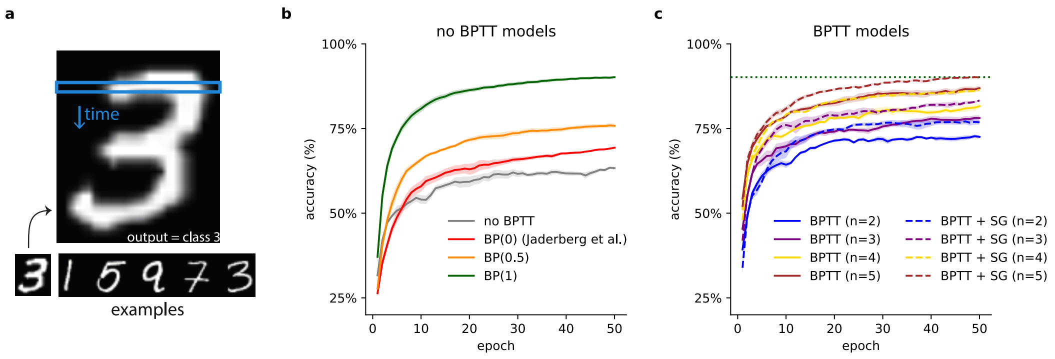

Figure 4: **Performance of** $\mathrm{BP}(\lambda)$ **in sequential MNIST task**. (**a**) Schematic of task. Rows of an MNIST image are fed sequentially as input and the model must classify the digit at the end. (**b**) Validation accuracy during training for $\mathrm{BP}(\lambda)$ models. (**c**) Validation accuracy during training for models which learn synthetic gradients (SG) with n-step truncated BPTT as in original implementation (Jaderberg et al., 2017); final performance of $\mathrm{BP}(1)$ (as in (b); dotted green) is given for reference. Results show mean performance over 5 different initial conditions with shaded areas representing standard error of the mean.

To test the ability of $\mathrm{BP}(\lambda)$ to generalise to non-trivial tasks, we now consider the sequential MNIST task (Le et al., 2015). In this task the RNN is provided with a row-by-row representation of an MNIST image which it must classify at the end (Fig. 4**a**). That is, as in the toy task above, the task loss is only defined at the final timestep and must be effectively associated to prior inputs. Since this is a harder task we now use non-linear LSTM units in the RNN which are better placed for these temporal tasks (Hochreiter & Schmidhuber, 1997).

Our results show that the use of $\mathrm{BP}(\lambda)$ significantly improves on a standard no-BPTT model, i.e. with $\frac{\partial L_{>t}}{\partial h_t} = 0$ in Eq. 3 (Fig. 4**b**). Consistent with our results from the toy task we find that a high λ produces faster rates of learning and higher performance. For example, $\mathrm{BP}(1)$ achieves error less than a third of that achieved by the eligibility trace free model, $\mathrm{BP}(0)$ ($\sim$ 10% vs $\sim$ 30%). Moreover, $\mathrm{BP}(1)$ generally outperforms the models considered by Jaderberg et al. (2017) which rely on truncated BPTT and the n-step synthetic gradient with $n > 1$ (Fig. 4**c** and Table 2).

4.3 COPY-REPEAT TASK

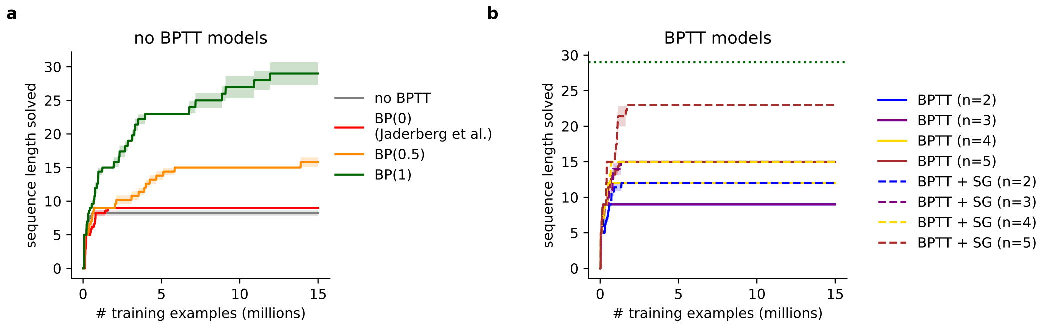

Figure 5: **Performance of** $\mathrm{BP}(\lambda)$ **in copy-repeat task**. (**a**) Maximum sequence length solved for $\mathrm{BP}(\lambda)$ models. A sequence length is considered as solved if the model achieves an average of 0.15 bits error for a given length. (**b**) Maximum sequence length solved for models with n-step synthetic gradient (SG) learning methods (Jaderberg et al., 2017); best task performance of $\mathrm{BP}(1)$ (as in (a); dotted green) is shown for reference. See also Table 2 for more details. Results show mean performance over 5 different initial conditions with shaded areas representing standard error of the mean.

Finally, we consider a task with more complex and longer temporal dependencies – the copy-repeat task (Graves et al., 2014). In this task the model receives as input a start delimiter followed by an 8-dimensional binary sequence of length N and a repeat character R. The model must then output the sequence R times before finishing with a stop character. The total sequence length is then $T = N \times (R + 1) + 3$. We follow the procedure as set out in Jaderberg et al. (2017) and deem a sequence length solved if the average error is less than 0.15 bits. Once solved we increment N and R alternatively.

We again observe that $\mathrm{BP}(1)$ provides the best BPTT-free solution in solving large sequence lengths and outperforms the n-step synthetic gradient methods (Fig. 5 and Table. 2).

5 RELEVANCE FOR LEARNING IN BIOLOGICAL NETWORKS

The $\mathrm{BP}(\lambda)$ algorithm is of potential interest to neuroscience. Unlike BPTT which is considered biologically implausible (Lillicrap & Santoro, 2019), $\mathrm{BP}(\lambda)$ is fully online and avoids the need to store intermediate activations (and complex gradient calculations back in time) across an arbitrary number of timesteps. Moreover, the application of synthetic gradients has relatively cheap computational and memory costs when compared to other online learning algorithms (Marschall et al., 2020). Perhaps most interestingly, $\mathrm{BP}(\lambda)$ employs a combination of retrospective and prospective learning, each of which are thought to take place in the brain (Namboodiri & Stuber, 2021). Specifically, whilst the main network learns via prospective learning signals (i.e. synthetic gradients), the synthesiser itself learns in a retrospective manner by using forward-propagating eligibility traces.

One possible candidate for the expression of synthetic gradients in the nervous system are neuromodulators. For example, dopaminergic neurons are known to encode expectation of future reward or error signals (Hollerman & Schultz, 1998) and have been observed to play an important role in mediating synaptic plasticity (Yagishita et al., 2014; Gerstner et al., 2018). Furthermore, as required for synthetic gradient vectors, there is increasing evidence for significant heterogeneity in the dopaminergic population in areas such as the ventral tegmental area, both in its variety of encoded signals and the targeted downstream circuits (Lerner et al., 2015; Beier et al., 2015; Avvisati et al., 2022). Each signal may thus reflect a predicted gradient with respect to the target cortical cell or cell ensemble.

More recently, it has also been suggested that a particular subcortical structure – the cerebellum – predicts cortical error gradients via the cortico-cerebellar loop (Pemberton et al., 2021; Boven et al., 2023). The cerebellum would thus act as a synthesiser for the brain, and it is suggested that a bootstrapped learning strategy may be in line with experimentally observations (Ohmae S, 2019; Kawato et al., 2021). $\mathrm{BP}(\lambda)$ offers an extra degree of biological plausibility to these studies by removing the need for BPTT at all. Moreover, the algorithm makes specific predictions regarding the need for eligibility traces (cf. Eq. 6) at key learning sites such as the cerebellar parallel fibres (Kawato et al., 2011).

6 Limitations

Like the originally proposed algorithm for synthetic gradients, $\mathrm{BP}(\lambda)$ makes the assumption that future error gradients can be modeled as a function of the current activity in the task-performing network. The algorithm may therefore suffer when the mapping between activity and future errors becomes less predictable, for example when the task involves stochastic inputs. Indeed, we find that whilst $\mathrm{BP}(\lambda)$ works well for tasks with fairly structured temporal structure as in the experiments presented, the algorithm can fall short in tasks with little or no temporal correlation between task inputs. For example, we failed to observe meaningful gains with $\mathrm{BP}(\lambda)$ in the ListOps task which employs randomly generated inputs (Tay et al., 2020). On a related note, it may be that in the case of variable, non-deterministic settings $\mathrm{BP}(\lambda)$ may be prone to a high variance in the synthesiser target gradient, resulting in potential instability during learning as can occur, for example, in Monte Carlo algorithms in RL. We emphasise that in all of the tasks presented higher λ values are optimal, but whether mid-range values can be beneficial for certain task conditions deserves future exploration.

Additionally, whilst the operations of $\mathrm{BP}(\lambda)$ are time-dependent, the algorithm can still be expensive if there are many hidden units in the RNN. Specifically, if there are $|h|$ RNN units and a linear synthesiser is applied then, due to the requirements of storing and updating the synthesiser eligibility traces, the memory and computational complexity of the algorithm is $\mathcal{O}(|h|^3)$ and $\mathcal{O}(|h|^4)$ respectively; this is as costly as the notoriously expensive real-time recurrent learning algorithm (Williams & Zipser, 1989). We postulate that one way to alleviate this issue is to not learn error gradients with respect to RNN activity directly, but instead some low dimensional representation of that error gradient. In particular, it has been recently demonstrated that gradient descent often takes place within a small subspace Gur-Ari et al. (2018). In principle, therefore, the synthesiser could learn some encoding of the error gradient of dimensionality $s << |h|$, significantly reducing the complexity of $\mathrm{BP}(\lambda)$. We predict that such low dimensional representations, which can reduce the noise of high dimensional error gradients, may also lead to more stable synthesiser learning. Speculating further, it may be that this bottleneck role is played by the thalamus in the communication of predicted error feedback in cerebellar-cortico loops (see previous section; Pemberton et al. (2021)).

7 Conclusion

BPTT can be expensive and enforce long waiting times before gradients become available. Synthetic gradients remove these locking constraints imposed by BPTT as well as the associated computational and memory complexity with respect to the task length (Jaderberg et al., 2017). However, the bootstrapped n-step algorithm for learning synthetic gradients as proposed by Jaderberg et al. can lead to biased estimates and also maintains some dependence on (truncated) BPTT to be performed.

Inspired by the $\mathrm{TD}(\lambda)$ algorithm in RL we propose a novel algorithm for learning synthetic gradients: $\mathrm{BP}(\lambda)$. This algorithm applies forward propagating eligibility traces in order to reduce the bias of its estimates and is fully online. We thus extend the work of Jaderberg et al. in developing a computational bridge between estimating expected return in the RL paradigm and future error gradients in supervised learning.

Through a combination of analytical and empirical work we show that $\mathrm{BP}(\lambda)$ outperforms the original implementation for synthetic gradients. Moreover, our model offers an efficient online solution for temporal supervised learning that is of relevance for both artificial and biological networks.

REFERENCES

Riccardo Avvisati, Anna-Kristin Kaufmann, Callum J Young, Gabriella E Portlock, Sophie Cancemi, Rui Ponte Costa, Peter J Magill, and Paul D Dodson. Distributional coding of associative learning within projection-defined populations of midbrain dopamine neurons. *bioRxiv*, 2022.

Kevin T Beier, Elizabeth E Steinberg, Katherine E DeLoach, Stanley Xie, Kazunari Miyamichi, Lindsay Schwarz, Xiaojing J Gao, Eric J Kremer, Robert C Malenka, and Liqun Luo. Circuit architecture of vta dopamine neurons revealed by systematic input-output mapping. *Cell*, 162(3): 622–634, 2015.

Ellen Boven, Joseph Pemberton, Paul Chadderton, Richard Apps, and Rui Ponte Costa. Cerebro-cerebellar networks facilitate learning through feedback decoupling. *Nature Communications*, 14 (1):1–18, 2023.

Wojciech Marian Czarnecki, Grzegorz Swirszcz, Max Jaderberg, Simon Osindero, Oriol Vinyals, and Koray Kavukcuoglu. Understanding synthetic gradients and decoupled neural interfaces. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 904–912. JMLR. org, 2017.

Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

Wulfram Gerstner, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea. Eligibility traces and plasticity on behavioral time scales: experimental support of neohebbian three-factor learning rules. *Frontiers in neural circuits*, 12:53, 2018.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Guy Gur-Ari, Daniel A Roberts, and Ethan Dyer. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL `https://doi.org/10.1162/neco.1997.9.8.1735`.

Jeffrey R Hollerman and Wolfram Schultz. Dopamine neurons report an error in the temporal prediction of reward during learning. *Nature neuroscience*, 1(4):304–309, 1998.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1627–1635. JMLR. org, 2017.

Mitsuo Kawato, Shinya Kuroda, and Nicolas Schweighofer. Cerebellar supervised learning revisited: biophysical modeling and degrees-of-freedom control. *Current opinion in neurobiology*, 21 (5):791–800, 2011.

Mitsuo Kawato, Shogo Ohmae, Huu Hoang, and Terry Sanger. 50 years since the Marr, Ito, and Albus models of the cerebellum. *Neuroscience*, 462:151–174, 2021.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

Talia N Lerner, Carrie Shilyansky, Thomas J Davidson, Kathryn E Evans, Kevin T Beier, Kelly A Zalocusky, Ailey K Crow, Robert C Malenka, Liqun Luo, Raju Tomer, et al. Intact-brain analyses reveal distinct information carried by snc dopamine subcircuits. *Cell*, 162(3):635–647, 2015.

Timothy P Lillicrap and Adam Santoro. Backpropagation through time and the brain. *Current opinion in neurobiology*, 55:82–89, 2019.

Owen Marschall, Kyunghyun Cho, and Cristina Savin. Evaluating biological plausibility of learning algorithms the lazy way. In *Real Neurons & Hidden Units: Future directions at the intersection of neuroscience and artificial intelligence @ NeurIPS 2019*, 2019. URL `https://openreview.net/forum?id=HJgPEXtIUS`.

Owen Marschall, Kyunghyun Cho, and Cristina Savin. A unified framework of online learning algorithms for training recurrent neural networks. *Journal of Machine Learning Research*, 21 (135):1–34, 2020.

Vijay Mohan K Namboodiri and Garret D Stuber. The learning of prospective and retrospective cognitive maps within neural circuits. *Neuron*, 109(22):3552–3575, 2021.

Medina JF Ohmae S. Plasticity of ponto-cerebellar circuits generates a prospective error signal in climbing fiber. *Program No. 579.01. Neuroscience 2019 Abstracts. Chicago, IL: Society for Neuroscience, 2019. Online*, 2019.

Joseph Pemberton, Ellen Boven, Richard Apps, and Rui Ponte Costa. Cortico-cerebellar networks as decoupling neural interfaces. *Advances in Neural Information Processing Systems*, 34, 2021.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

Harm Van Seijen, A Rupam Mahmood, Patrick M Pilarski, Marlos C Machado, and Richard S Sutton. True online temporal-difference learning. *The Journal of Machine Learning Research*, 17 (1):5057–5096, 2016.

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

Sho Yagishita, Akiko Hayashi-Takagi, Graham C R Ellis-Davies, Hidetoshi Urakubo, Shin Ishii, and Haruo Kasai. A critical time window for dopamine actions on the structural plasticity of dendritic spines. *Science*, 345(6204):1616–1620, 2014.

A Relationship between learning synthetic gradients and learning expected return in Reinforcement Learning

This section serves as a more detailed supplement to the problem of learning synthetic gradients, and its exploited relationship to Reinforcement Learning (RL).

A.1 Description of problem

The problem of learning synthetic gradients can be stated as trying to minimise the synthesiser loss function $L^g(\theta)$ defined as

$$L^g(\theta) := \mathbb{E}_{h_t \sim P}\left[\frac{1}{2}\left\|v_t - \hat{G}_t\right\|_2^2\right] \tag{10}$$

$$= \frac{1}{2}\sum_t P\left(h_t\right)\left\|v_t - g(h_t;\theta)\right\|_2^2, \tag{11}$$

where P is the probability distribution over RNN hidden states, v_t is the target synthetic gradient for state h_t, and $\|.\|_2$ denotes the Euclidean norm. By taking samples of hidden states h_t and applying the chain rule, the stochastic updates for θ can be written as

$$\theta_{t+1} = \theta_t + \alpha\left(v_t - g(h_t;\theta_t)\right)^\top \nabla_\theta g(h_t;\theta_t), \tag{12}$$

where α is the synthesiser learning rate. Note the transpose operation $\cdot^\top$ since all terms in Eq. 12 are vectors.

Ideally v_t is the true error gradient, $v_t = G_t$, but this requires full BPTT which is exactly what synthetic gradients are used to avoid.

A.2 Original solution to learning synthetic gradients

In its original formulation, Jaderberg et al. instead propose a target which is based on mixture of backpropagated error gradients within a fixed window and a bootstrapped synthetic gradient to incorporate errors beyond. The simplest version of this, which only uses one-step BPTT, only incorporates the error at the next timestep and relies on a bootstrap prediction for all timesteps onwards. We denote this target $G_t^{(1)}$.

$$G_t^{(1)} = \frac{\partial L_{t+1}}{\partial h_t} + \gamma \hat{G}_{t+1}\frac{\partial h_{t+1}}{\partial h_t}, \tag{13}$$

where, inspired by RL, $\gamma \in [0,1]$ is the gradient *discount factor* which if $\gamma < 1$ scales down later error gradients. Note that in its original implementation $\gamma = 1$, but in our simulations we find it important to set $\gamma < 1$ (see Section D).

Notably, Eq. 13 resembles the bootstrapped target involved in the TD algorithm in RL (Sutton & Barto, 2018). In particular, $G_t^{(1)}$ can be considered analogous to the one-step return at state S_t defined as $R_{t+1} + \gamma V(S_{t+1})$, where R_{t+1} is the reward, S_{t+1} is the subsequent state, and V is the (bootstrapped) value function.

A.3 n-step synthetic gradients

In practice, in its original implementation Jaderberg et al. primarily consider applying synthetic gradients alongside truncated BPTT for truncation size $n > 1$.

In this case, the left side term in Eq. 13 can be extended to incorporate loss gradients backpropagated within the n timesteps of the truncation. Jaderberg et al. then set the synthesiser target as $v_t = G_t^{(n)}$

where $G_t^{(n)}$ is the *n-step synthetic gradient* defined as

$$G_t^{(n)} = \sum_{t<\tau\leq t+n} \gamma^{\tau-t-1}\frac{\partial L_\tau}{\partial h_t} + \gamma^n \hat{G}_{t+n}\frac{\partial h_{t+n}}{\partial h_t}, \tag{14}$$

which is analogous to the n-step return in RL. Importantly, in practice this scheme uses truncated BPTT to both define the synthesiser target and the error gradient used to update the RNN. Specifically, Jaderberg et al. apply the n backpropagated gradients to directly learn Ψ as well as θ, with the synthetic gradients themselves only applied at the end of each truncation window (see original paper for details).

Just as in RL, increasing the truncation size n provides a target which assigns more weight to the observations than the bootstrapped term, thus reducing its *bias* and potentially leading to better synthesiser learning. On the other hand, Eq. 14 requires n-step BPTT and thereby enforces undesirable waiting time and complexity for large n.

A.4 Extension to λ-based methods

In the supplementary material of the original paper (Jaderberg et al., 2017), Jaderberg et al. further develop the relationship between learning synthetic gradients and methods in Reinforcement Learning by introducing the notion of a λ-weighted synthetic gradient.

We now define λ-weighted synthetic gradients as used in this study. We highlight that our definition is similar, but not the same, as that first introduced in Jaderberg et al. (2017). Specifically, our definition incorporates loss gradients for losses strictly after the current timestep and can therefore naturally be used in the context of learning the synthesiser.

A.5 λ-weighted synthetic gradient

Let λ be such that $0 \leq \lambda \leq 1$. We define the *λ-weighted synthetic gradient* as

$$G_t^\lambda := (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}G_t^{(n)} \tag{15}$$

$$= (1-\lambda)\sum_{n=1}^{T-t-1}\lambda^{n-1}G_t^{(n)} + \lambda^{T-t-1}G_t, \tag{16}$$

which is analogous to the λ-return in RL.

Note the distinction in notation between the λ-weighted synthetic gradient G_t^λ and the n-step synthetic gradient $G_t^{(n)}$. Moreover, note that Eq. 16 is similar but distinct to the recursive definition as proposed in Jaderberg et al. (2017) (see Appendix section C). In particular, whilst Jaderberg et al. also incorporate the loss gradient at the current timestep in their definition, Eq. 16 only considers strictly future losses. Since the synthesiser itself is optimised to produce future error gradients (Eq. 4), this enables the λ-weighted synthetic gradient to be directly used as a synthesiser target.

As in RL, a higher choice of λ results in stronger weighting of observed gradients compared to the bootstrapped terms. For example, whilst $G_t^0 = G_t^{(1)}$ is just the one-step synthetic gradient which relies strongly on the bootstrapped prediction (cf. Eq. 13), $G_t^1 = G_t$ is the true (unbiased) gradient as obtained via full BPTT.

We also define the *interim λ-weighted synthetic gradient* $G_k^{\lambda|H}$ as

$$G_k^{\lambda|H} := (1-\lambda)\sum_{n=1}^{H-k-1}\lambda^{n-1}G_k^{(n)} + \lambda^{H-k-1}G_k^{(H-k)}. \tag{17}$$

Unlike Eq. 16, this is available at time ("horizon") H with $k < H < T$.

Table 1 provides an overview of the terms defined along with their respective counterparts in RL.

A.5.1 Offline λ-SG algorithm

We define the *offline λ-SG algorithm* to learn θ, which is analogous to the offline λ-return algorithm in RL but for synthetic gradients (SG), by taking $v_t = G_t^\lambda$ in Eq. 12. It is offline in the sense that it requires the completion of the sequence at timestep T before updates are possible.

A.5.2 Online λ-SG algorithm

We define the *online λ-SG algorithm* to learn θ, which is analogous to the online λ-return algorithm in RL but for synthetic gradients. At the current timestep t, the algorithm updates θ based on its prediction over all prior RNN hidden states h_k from $k = 0$ up to $k = t$. Explicitly, the update at timestep t for state k is

$$\theta_k^t = \theta_{k-1}^t + \alpha \left(G_k^{\lambda|t} - g(h_k; \theta_{k-1}^t) \right)^\top \nabla_{\theta_{k-1}^t} g(h_k; \theta_{k-1}^t), \tag{18}$$

where $\theta_0^0 = \theta_{\text{init}}$ is the initialisation weight and $\theta_0^t = \theta_{t-1}^{t-1}$ for $t > 0$.

The information used for the weight updates in Eq. 18 is available at the current timestep and the algorithm is therefore online. Moreover, as in RL, the online λ-SG algorithm produces weight updates similar to the offline λ-SG algorithm. In particular, at the end of the sequence with the horizon $H = T$ note that $G_t^{\lambda|H}$ and G_t^λ are the same.

However, to store the previous hidden states and iteratively apply the online λ-SG algorithm requires undesirable computational cost. In particular, the $1 + 2 + \cdots + T$ operations in Eq. 18 result in computational complexity which scales intractably with T^2.

A.6 Accumulate BP(λ)

The accumulate BP(λ) algorithm is defined by Eqs. 6, 7 and 8 in the main text. The main analytical result in this paper is that accumulate BP(λ) provides an approximation to learning λ-weighted synthetic gradients, which are analogous to λ-returns in RL. This theorem uses the term Δ_i^t defined as

$$\Delta_i^t := \left(\bar{G}_i^{\lambda|t} - g(h_i; \theta_0) \right)^\top \nabla_{\theta_i} g(h_i; \theta_i), \tag{19}$$

where $\bar{G}_i^{\lambda|t}$ is the λ-weighted synthetic gradient which uses the initial weight vector θ_0 for all synthetic gradient estimations.

Theorem A.1. *Let θ_0 be the initial weight vector, θ_t^{BP} be the weight vector at time t computed by accumulate BP(λ), and θ_t^λ be the weight vector at time t computed by the online λ-SG algorithm. Furthermore, assume that $\sum_{i=0}^{t-1} \Delta_i^t$ does not contain any zero-elements. Then, for all time steps t:*

$$\frac{\left\|\theta_t^{BP} - \theta_t^\lambda\right\|_2}{\left\|\theta_t^{BP} - \theta_0\right\|_2} \to 0 \quad as \quad \alpha \to 0$$

.

Proof: The full proof is presented next in Section B. In general, the structure of the proof closely follows that provided in the analogous RL paradigm for accumulate $TD(\lambda)$ (Van Seijen et al., 2016) $\square$.

B Proof of Theorem A.1

As this section is more involved, for clarity we denote scalar variables in italics and vectors in bold.

Approach: We demonstrate that Theorem A.1 holds for arbitrary synthesiser parameter θ^{ji} (at the ith column and jth row of θ) ; that is,

$$\frac{\left\|\theta_t^{ji,BP} - \theta_t^{ji,\lambda}\right\|_2}{\left\|\theta_t^{ji,BP} - \theta_0^{ji}\right\|_2} \to 0, \quad \text{as} \quad \alpha \to 0.$$

This is proved in Proposition B.7.

For reference, we explicitly write the weight updates for parameter θ^{ji} according to the accumulate BP(λ) algorithm:

$$\boldsymbol{e}_t^{ji} = \gamma\lambda \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \boldsymbol{e}_{t-1}^{ji} + \nabla_{\theta_t^{ji}} g(\boldsymbol{h}_t; \boldsymbol{\theta}_t), \tag{20}$$

$$\boldsymbol{\delta}_t = \frac{\partial L_{t+1}}{\partial \boldsymbol{h}_t} + \gamma g(\boldsymbol{h}_{t+1}; \boldsymbol{\theta}_t)^\top \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} - g(\boldsymbol{h}_t; \boldsymbol{\theta}_t), \tag{21}$$

$$\theta_{t+1}^{ji} = \theta_t^{ji} + \alpha \boldsymbol{\delta}_t^\top \boldsymbol{e}_t^{ji}. \tag{22}$$

we also define the helper variable $\boldsymbol{\delta}_{a,t}$ as

$$\boldsymbol{\delta}_{a,t} = \frac{\partial L_{t+1}}{\partial \boldsymbol{h}_a} + \gamma g(\boldsymbol{h}_{t+1}; \boldsymbol{\theta}_t)^\top \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_a} - g(\boldsymbol{h}_t; \boldsymbol{\theta}_t)^\top \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_a} = \boldsymbol{\delta}_t^\top \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_a}. \tag{23}$$

In general, we follow the structure of the analogous proof for the accumulate TD(λ) algorithm (Van Seijen et al., 2016).

B.1 Statements and proofs

Lemma B.1. $\boldsymbol{G}_a^{\lambda|t+1} = \boldsymbol{G}_a^{\lambda|t} + (\lambda\gamma)^{t-a}\boldsymbol{\delta}'_{a,t}$, *where*

$$\boldsymbol{\delta}'_{a,t} = \frac{\partial L_{t+1}}{\partial \boldsymbol{h}_a} + \gamma g(\boldsymbol{h}_{t+1}; \boldsymbol{\theta}_t)^\top \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_a} - g(\boldsymbol{h}_t; \boldsymbol{\theta}_{t-1})^\top \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_a}. \tag{24}$$

Proof:

$$\begin{aligned}
\boldsymbol{G}_a^{\lambda|t+1} - \boldsymbol{G}_a^{\lambda|t} =& (1-\lambda)\sum_{n=1}^{t-a} \lambda^{n-1}\boldsymbol{G}_a^{(n)} + \lambda^{t-a}\boldsymbol{G}_a^{(t+1-a)} \qquad \text{Definition: Equation 17} \\
&- (1-\lambda)\sum_{n=1}^{t-a-1} \lambda^{n-1}\boldsymbol{G}_a^{(n)} - \lambda^{t-a-1}\boldsymbol{G}_a^{(t-a)} \\
=& (1-\lambda)\lambda^{t-a-1}\boldsymbol{G}_a^{(t-a)} + \lambda^{t-a}\boldsymbol{G}_a^{(t+1-a)} - \lambda^{t-a-1}\boldsymbol{G}_a^{(t-a)} \\
=& \lambda^{t-a}\boldsymbol{G}_a^{(t+1-a)} - \lambda^{t-a}\boldsymbol{G}_a^{(t-a)} \\
=& \lambda^{t-a}\left(\boldsymbol{G}_a^{(t+1-a)} - \boldsymbol{G}_a^{(t-a)}\right) \\
=& \lambda^{t-a}\left(\sum_{k=1}^{t+1-a}\gamma^{k-1}\frac{\partial L_{a+k}}{\partial \boldsymbol{h}_a} + \gamma^{t+1-a} g(\boldsymbol{h}_t; \boldsymbol{\theta}_t)^\top \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_a} - \sum_{k=1}^{t-a}\gamma^{k-1}\frac{\partial L_{a+k}}{\partial \boldsymbol{h}_a} - \gamma^{t-a} g(\boldsymbol{h}_t; \boldsymbol{\theta}_{t-1})^\top \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_a}\right) \\
=& \lambda^{t-a}\left(\gamma^{t-a}\frac{\partial L_{t+1}}{\partial \boldsymbol{h}_a} + \gamma^{t+1-a} g(\boldsymbol{h}_t; \boldsymbol{\theta}_t)^\top \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_a} - \gamma^{t-a} g(\boldsymbol{h}_t; \boldsymbol{\theta}_{t-1})^\top \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_a}\right) \\
=& (\lambda\gamma)^{t-a}\underbrace{\left(\frac{\partial L_{t+1}}{\partial \boldsymbol{h}_a} + \gamma g(\boldsymbol{h}_t; \boldsymbol{\theta}_t)^\top \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_a} - g(\boldsymbol{h}_t; \boldsymbol{\theta}_{t-1})^\top \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_a}\right)}_{\boldsymbol{\delta}'_{a,t}} \qquad \square
\end{aligned}$$

Lemma B.2. $\boldsymbol{G}_a^{\lambda|t} = \boldsymbol{G}_a^{\lambda|a+1} + \sum_{b=a+1}^{t-1}(\gamma\lambda)^{b-a}\boldsymbol{\delta}'_{a,b}$

Proof: We apply Lemma B.1 recursively:

$$\begin{aligned}
\boldsymbol{G}_a^{\lambda|t} &= \boldsymbol{G}_a^{\lambda|t-1} + (\lambda\gamma)^{t-1-a}\boldsymbol{\delta}'_{a,t-1} \\
&= \left(\boldsymbol{G}_a^{\lambda|t-2} + (\lambda\gamma)^{t-2-a}\boldsymbol{\delta}'_{a,t-2}\right) + (\lambda\gamma)^{t-1-a}\boldsymbol{\delta}'_{a,t-1} \\
&= \boldsymbol{G}_a^{\lambda|a+1} + (\lambda\gamma)^{1}\boldsymbol{\delta}'_{a,a+1} + \cdots + (\lambda\gamma)^{t-1-a}\boldsymbol{\delta}'_{a,t-1} \\
&= \boldsymbol{G}_a^{\lambda|a+1} + \sum_{b=a+1}^{t-1}(\gamma\lambda)^{b-a}\boldsymbol{\delta}'_{a,b} \qquad \square
\end{aligned}$$

Lemma B.3. $\boldsymbol{G}_a^{\lambda|a+1} = \boldsymbol{\delta}'_{a,a} + g(\boldsymbol{h}_a;\boldsymbol{\theta}_{a-1})$

Proof:

$$\begin{aligned}
\boldsymbol{G}_a^{\lambda|a+1} &= \boldsymbol{G}_a^{1} \\
&= \frac{\partial L_{a+1}}{\partial \boldsymbol{h}_a} + \gamma g(\boldsymbol{h}_{a+1};\boldsymbol{\theta}_a)^\top \frac{\partial \boldsymbol{h}_{a+1}}{\partial \boldsymbol{h}_a} \\
&= \frac{\partial L_{a+1}}{\partial \boldsymbol{h}_a} + \gamma g(\boldsymbol{h}_{a+1};\boldsymbol{\theta}_a)^\top \frac{\partial \boldsymbol{h}_{a+1}}{\partial \boldsymbol{h}_a} - g(\boldsymbol{h}_t;\boldsymbol{\theta}_{t-1})^\top \frac{\partial \boldsymbol{h}_a}{\partial \boldsymbol{h}_a} + g(\boldsymbol{h}_a;\boldsymbol{\theta}_{a-1})^\top \frac{\partial \boldsymbol{h}_a}{\partial \boldsymbol{h}_a} \\
&= \boldsymbol{\delta}'_{a,a} + g(\boldsymbol{h}_a;\boldsymbol{\theta}_{a-1}) \qquad \square
\end{aligned}$$

Lemma B.4. $\sum_{b=a}^{t-1}(\gamma\lambda)^{b-a}\boldsymbol{\delta}'_{a,b} = \boldsymbol{G}_a^{\lambda|t} - g(\boldsymbol{h}_a;\boldsymbol{\theta}_{a-1})$

Proof:

$$\begin{aligned}
\boldsymbol{G}_a^{\lambda|t} &= \boldsymbol{G}_a^{\lambda|a+1} + \sum_{b=a+1}^{t-1}(\gamma\lambda)^{b-a}\boldsymbol{\delta}'_{a,b} && \text{Lemma B.2} \\
&= \boldsymbol{\delta}'_{a,a} + g(\boldsymbol{h}_a;\boldsymbol{\theta}_{a-1}) + \sum_{b=a+1}^{t-1}(\gamma\lambda)^{b-a}\boldsymbol{\delta}'_{a,b} && \text{Lemma B.3} \\
&= \boldsymbol{\theta}_{a-1}^\top \boldsymbol{h}_a + \sum_{b=a}^{t-1}(\gamma\lambda)^{b-a}\boldsymbol{\delta}'_{a,b} && \square
\end{aligned}$$

Lemma B.5. $\boldsymbol{e}_b^{ji} = \sum_{a=0}^{b}(\gamma\lambda)^{b-a}\frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_a}\nabla_{\theta_a^{ji}} g(\boldsymbol{h}_a;\boldsymbol{\theta}_a)$

Proof:

$$\begin{aligned}
\boldsymbol{e}_b^{ji} &= \gamma\lambda\frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_{b-1}}\boldsymbol{e}_{b-1}^{ji} + \nabla_{\theta_b^{ji}} g(\boldsymbol{h}_b;\boldsymbol{\theta}_b) \\
&= \gamma\lambda\frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_{b-1}}\left(\gamma\lambda\frac{\partial \boldsymbol{h}_{b-1}}{\partial \boldsymbol{h}_{b-2}}\boldsymbol{e}_{b-2}^{ji} + \nabla_{\theta_{b-1}^{ji}} g(\boldsymbol{h}_{b-1};\boldsymbol{\theta}_{b-1})\right) + \nabla_{\theta_b^{ji}} g(\boldsymbol{h}_b;\boldsymbol{\theta}_b) \\
&= (\gamma\lambda)^b\frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_{b-1}}\frac{\partial \boldsymbol{h}_{b-1}}{\partial \boldsymbol{h}_{b-2}}\cdots\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{h}_0}\nabla_{\theta_0^{ji}} g(\boldsymbol{h}_0;\boldsymbol{\theta}_0) + (\gamma\lambda)^{b-1}\frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_{b-1}}\cdots\frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{h}_1}\nabla_{\theta_1^{ji}} g(\boldsymbol{h}_1;\boldsymbol{\theta}_1) \\
&+ \cdots + \frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_{b-1}}\nabla_{\theta_{b-1}^{ji}} g(\boldsymbol{h}_{b-1};\boldsymbol{\theta}_{b-1}) + \nabla_{\theta_b^{ji}} g(\boldsymbol{h}_b;\boldsymbol{\theta}_b) \\
&= (\gamma\lambda)^b\frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_0}\nabla_{\theta_0^{ji}} g(\boldsymbol{h}_0;\boldsymbol{\theta}_0) + (\gamma\lambda)^{b-1}\frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_1}\nabla_{\theta_1^{ji}} g(\boldsymbol{h}_1;\boldsymbol{\theta}_1) \\
&+ \cdots + (\gamma\lambda)\frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_{b-1}}\nabla_{\theta_{b-1}^{ji}} g(\boldsymbol{h}_{b-1};\boldsymbol{\theta}_{b-1}) + \nabla_{\theta_b^{ji}} g(\boldsymbol{h}_b;\boldsymbol{\theta}_b) \\
&= \sum_{a=0}^{b}(\gamma\lambda)^{b-a}\frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_a}\nabla_{\theta_a^{ji}} g(\boldsymbol{h}_a;\boldsymbol{\theta}_a) \qquad \square
\end{aligned}$$

Lemma B.6. $\sum_{b=a}^{t-1}(\gamma\lambda)^{b-a}\boldsymbol{\delta}_{a,b} = \boldsymbol{G}_a^{\lambda|t} - g(\boldsymbol{h}_a;\boldsymbol{\theta}_{a-1}) + \mathcal{O}(\alpha)$

Proof: Using $\boldsymbol{\delta}_{a,t} = \boldsymbol{\delta}'_{a,t} - g(\boldsymbol{h}_t;\boldsymbol{\theta}_t)^\top \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_a} + g(\boldsymbol{h}_t;\boldsymbol{\theta}_{t-1})^\top \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_a}$, we have

$$\begin{aligned}
\sum_{b=a}^{t-1}(\gamma\lambda)^{b-a}\boldsymbol{\delta}_{a,b} &= \sum_{b=a}^{t-1}(\gamma\lambda)^{b-a}\left(\boldsymbol{\delta}'_{a,t} - g(\boldsymbol{h}_t;\boldsymbol{\theta}_t)^\top \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_a} + g(\boldsymbol{h}_t;\boldsymbol{\theta}_{t-1})^\top \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_a}\right) \\
&= \sum_{b=a}^{t-1}(\gamma\lambda)^{b-a}\boldsymbol{\delta}'_{a,b} - \sum_{b=a}^{t-1}(\gamma\lambda)^{b-a}\left(g(\boldsymbol{h}_t;\boldsymbol{\theta}_t) - g(\boldsymbol{h}_{t-1};\boldsymbol{\theta}_{t-1})\right)^\top \frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_a} \\
&= \sum_{b=a}^{t-1}(\gamma\lambda)^{b-a}\boldsymbol{\delta}'_{a,b} + \mathcal{O}(\alpha) \\
&= \boldsymbol{G}_a^{\lambda|t} - g(\boldsymbol{h}_a;\boldsymbol{\theta}_{a-1}) + \mathcal{O}(\alpha) && \text{Lemma B.4} \quad \square
\end{aligned}$$

Proposition B.7. *Let $\boldsymbol{\theta}_0$ be the initial weight vector and let θ^{ji} be an arbitrary parameter of $\boldsymbol{\theta}$. θ_t^{ji} be the weight at time t computed by accumulate BP(λ), and $\theta_t^{t,ji}$ be the weight at time t computed by the online λ -SG algorithm. Furthermore, assume that $\sum_{a=0}^{t-1}\Delta_a^{t,ji} \neq 0$, where $\Delta_a^{t,ji} := \left(\bar{\boldsymbol{G}}_a^{\lambda|t} - g(\boldsymbol{h}_a;\boldsymbol{\theta}_0)\right)^\top \nabla_{\theta_a^{ji}} g(\boldsymbol{h}_a;\boldsymbol{\theta}_a)$ and $\bar{\boldsymbol{G}}_a^{\lambda|t}$ is the λ-weighted synthetic gradient which uses $\boldsymbol{\theta}_0$ for all synthetic gradient estimations. Then, for all time steps t:*

$$\frac{\left\|\theta_t^{ji} - \theta_t^{t,ji}\right\|_2}{\left\|\theta_t^{ji} - \theta_0^{ji}\right\|_2} \to 0 \quad as \quad \alpha \to 0$$

.

Proof: The updates according to accumulate BP(λ) (Equations 21 to 22) follow

$$\begin{aligned}
\theta_t^{ji} &= \theta_0^{ji} + \alpha\sum_{b=0}^{t-1}\boldsymbol{\delta}_b^\top \boldsymbol{e}_b && \text{Definition: Equation 20} \\
&= \theta_0^{ji} + \alpha\sum_{b=0}^{t-1}\boldsymbol{\delta}_b^\top\left(\sum_{a=0}^{b}(\gamma\lambda)^{b-a}\frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_a}\nabla_{\theta_a^{ji}} g(\boldsymbol{h}_a;\boldsymbol{\theta}_a)\right) && \text{Lemma B.5} \\
&= \theta_0^{ji} + \alpha\sum_{b=0}^{t-1}\sum_{a=0}^{b}(\gamma\lambda)^{b-a}\boldsymbol{\delta}_b^\top\frac{\partial \boldsymbol{h}_b}{\partial \boldsymbol{h}_a}\nabla_{\theta_a^{ji}} g(\boldsymbol{h}_a;\boldsymbol{\theta}_a) \\
&= \theta_0^{ji} + \alpha\sum_{b=0}^{t-1}\sum_{a=0}^{b}(\gamma\lambda)^{b-a}\boldsymbol{\delta}_{a,b}^\top\nabla_{\theta_a^{ji}} g(\boldsymbol{h}_a;\boldsymbol{\theta}_a) && \text{Definition: Equation 23} \\
&= \theta_0^{ji} + \alpha\sum_{a=0}^{t-1}\left(\sum_{b=a}^{t-1}(\gamma\lambda)^{b-a}\boldsymbol{\delta}_{a,b}^\top\right)\nabla_{\theta_a^{ji}} g(\boldsymbol{h}_a;\boldsymbol{\theta}_a) && \sum_{b=k}^{n}\sum_{a=k}^{b}x_{a,b} = \sum_{a=k}^{n}\sum_{b=a}^{n}x_{a,b} \\
&= \theta_0^{ji} + \alpha\sum_{a=0}^{t-1}\left(\boldsymbol{G}_a^{\lambda|t} - g(\boldsymbol{h}_a;\boldsymbol{\theta}_{a-1}) + \mathcal{O}(\alpha)\right)^\top\nabla_{\theta_a^{ji}} g(\boldsymbol{h}_a;\boldsymbol{\theta}_a) && \text{Lemma B.6} \\
&= \theta_0^{ji} + \alpha\sum_{a=0}^{t-1}\left(\bar{\boldsymbol{G}}_a^{\lambda|t} - g(\boldsymbol{h}_a;\boldsymbol{\theta}_0) + \mathcal{O}(\alpha)\right)^\top\nabla_{\theta_a^{ji}} g(\boldsymbol{h}_a;\boldsymbol{\theta}_a).
\end{aligned}$$

Where the last line holds as $\alpha \to 0$. On the other hand, the updates according to the online λ-SG algorithm produce

$$\begin{aligned}
\theta_t^{t,ji} &= \theta_0^{ji} + \alpha\sum_{a=0}^{t-1}\left(\boldsymbol{G}_a^{\lambda|t} - g(\boldsymbol{h}_a;\boldsymbol{\theta}_a)\right)^\top\nabla_{\theta_a^{ji}} g(\boldsymbol{h}_a;\boldsymbol{\theta}_a) && \text{Definition: Equation 18} \\
&= \theta_0^{ji} + \alpha\sum_{a=0}^{t-1}\left(\bar{\boldsymbol{G}}_a^{\lambda|t} - g(\boldsymbol{h}_a;\boldsymbol{\theta}_0) + \mathcal{O}(\alpha)\right)^\top\nabla_{\theta_a^{ji}} g(\boldsymbol{h}_a;\boldsymbol{\theta}_a).
\end{aligned}$$

Hence

$$\frac{\left\|\theta_t^{ji} - \theta_t^{t,ji}\right\|_2}{\left\|\theta_t^{ji} - \theta_0^{ji}\right\|_2} = \frac{\left\|\left(\theta_t^{ji} - \theta_t^{t,ji}\right)/\alpha\right\|_2}{\left\|\left(\theta_t^{ji} - \theta_0^{ji}\right)/\alpha\right\|_2} = \frac{\mathcal{O}(\alpha)}{C + \mathcal{O},(\alpha)}$$

where

$$C = \left\|\sum_{a=0}^{t-1} \left(\bar{\boldsymbol{G}}_a^{\lambda|t} - g(\boldsymbol{h}_a;\boldsymbol{\theta}_0)\right)^\top \nabla_{\theta_a^{ji}} g(\boldsymbol{h}_a;\boldsymbol{\theta}_a)\right\|_2 = \|\sum_{a=0}^{t-1} \Delta_a^{t,ji}\|_2.$$

From the condition that $\sum_{a=0}^{t-1} \Delta_a^{t,ji} \neq 0$ we have $C > 0$. Therefore

$$\frac{\left\|\theta_t^{ji} - \theta_t^{t,ji}\right\|_2}{\left\|\theta_t^{ji} - \theta_0^{ji}\right\|_2} \to 0 \quad \text{as} \quad \alpha \to 0 \qquad \square$$

C Recursive definition of $\boldsymbol{G}_t^\lambda$

In Proposition C.2 we provide a recursive definition of the λ-weighted synthetic gradient $\boldsymbol{G}_t^\lambda$ as defined in Equation 16. Note that this is similar, but not the same, as the recursive definition defined in A.3 in the supplementary material of Jaderberg et al. (2017). In particular, this definition strictly considers future losses as in the context of a synthesiser target, whilst that provided in Jaderberg et al. (2017) also considers the loss at the current timestep (L_t).

We first require the following Lemma

Lemma C.1. *For any sequence* $\{x_n\}_{n\geq 1}$,

$$\sum_{n=1}^{\infty} \lambda^{n-1} x_n = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} \sum_{k=1}^{n} x_k. \tag{25}$$

Proof: By the sum rule,

$$(1-\lambda) \sum_{n=1}^{\infty} \sum_{k=1}^{n} \lambda^{n-1} x_k = (1-\lambda) \sum_{k=1}^{\infty} \sum_{n=k}^{\infty} \lambda^{n-1} x_k$$

Now,

$$\begin{aligned}
\sum_{n=k}^{\infty} \lambda^{n-1} &= \sum_{n=1}^{\infty} \lambda^{n-1} - \sum_{n=1}^{k-1} \lambda^{n-1} \\
&= \sum_{n=0}^{\infty} \lambda^{n} - \sum_{n=0}^{k-2} \lambda^{n} \\
&= \frac{1}{1-\lambda} - \frac{1-\lambda^{k-1}}{1-\lambda} \\
&= \frac{\lambda^{k-1}}{1-\lambda}.
\end{aligned}$$

So

$$\begin{aligned}(1-\lambda)\sum_{n=1}^{\infty}\sum_{k=1}^{n}\lambda^{n-1}x_k &= (1-\lambda)\sum_{k=1}^{\infty}\left(\sum_{n=k}^{\infty}\lambda^{n-1}\right)x_k \\ &= (1-\lambda)\sum_{k=1}^{\infty}\frac{\lambda^{k-1}}{1-\lambda}x_k \\ &= \sum_{k=1}^{\infty}\lambda^{k-1}x_k \\ &= \sum_{n=1}^{\infty}\lambda^{n-1}x_n \qquad \square\end{aligned}$$

Proposition C.2. *We can define the λ-weighted synthetic gradient (Eq. 16) incrementally. In particular,*

$$\boldsymbol{G}_t^{\lambda} = \frac{\partial L_{t+1}}{\partial \boldsymbol{h}_t} + \gamma\lambda(\boldsymbol{G}_{t+1}^{\lambda})^{\top}\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} + \gamma(1-\lambda)g(\boldsymbol{h}_{t+1};\boldsymbol{\theta}_t)^{\top}\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t}. \tag{26}$$

Proof: Let $T \leq \infty$ be the sequence length and define $\boldsymbol{G}_t^{(\tau)} := 0$ for $\tau > T$. Then we can write $\boldsymbol{G}_{t+1}^{\lambda}$ as

$$\begin{aligned}\boldsymbol{G}_{t+1}^{\lambda} &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\boldsymbol{G}_{t+1}^{(n)} \\ &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\left[\sum_{k=1}^{n}\gamma^{k-1}\frac{\partial L_{t+1+k}}{\partial \boldsymbol{h}_{t+1}} + \gamma^{n}g(\boldsymbol{h}_{t+1+n};\boldsymbol{\theta}_{t+n})^{\top}\frac{\partial \boldsymbol{h}_{t+1+n}}{\partial \boldsymbol{h}_{t+1}}\right] \\ &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\sum_{k=1}^{n}\gamma^{k-1}\frac{\partial L_{t+1+k}}{\partial \boldsymbol{h}_{t+1}} + (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\gamma^{n}g(\boldsymbol{h}_{t+1+n};\boldsymbol{\theta}_{t+n})^{\top}\frac{\partial \boldsymbol{h}_{t+1+n}}{\partial \boldsymbol{h}_{t+1}} \\ &= \sum_{n=1}^{\infty}\lambda^{n-1}\gamma^{n-1}\frac{\partial L_{t+1+n}}{\partial \boldsymbol{h}_{t+1}} + (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\gamma^{n}g(\boldsymbol{h}_{t+1+n};\boldsymbol{\theta}_{t+n})^{\top}\frac{\partial \boldsymbol{h}_{t+1+n}}{\partial \boldsymbol{h}_{t+1}} \qquad \text{Lemma C.1} \\ &= \sum_{n=2}^{\infty}\lambda^{n-2}\gamma^{n-2}\frac{\partial L_{t+n}}{\partial \boldsymbol{h}_{t+1}} + (1-\lambda)\sum_{n=2}^{\infty}\lambda^{n-2}\gamma^{n-1}g(\boldsymbol{h}_{t+n};\boldsymbol{\theta}_{t+n-1})^{\top}\frac{\partial \boldsymbol{h}_{t+n}}{\partial \boldsymbol{h}_{t+1}}.\end{aligned}$$

So

$$\begin{aligned}
\boldsymbol{G}_t^\lambda &= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\boldsymbol{G}_t^{(n)} \\
&= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\left[\sum_{k=1}^{n}\gamma^{k-1}\frac{\partial L_{t+k}}{\partial \boldsymbol{h}_t} + \gamma^n g(\boldsymbol{h}_{t+n};\boldsymbol{\theta}_{t+n-1})^\top \frac{\partial \boldsymbol{h}_{t+n}}{\partial \boldsymbol{h}_t}\right] \\
&= (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\sum_{k=1}^{n}\gamma^{k-1}\frac{\partial L_{t+k}}{\partial \boldsymbol{h}_t} + (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\gamma^n g(\boldsymbol{h}_{t+n};\boldsymbol{\theta}_{t+n-1})^\top \frac{\partial \boldsymbol{h}_{t+n}}{\partial \boldsymbol{h}_t} \\
&= \sum_{n=1}^{\infty}\lambda^{n-1}\gamma^{n-1}\frac{\partial L_{t+n}}{\partial \boldsymbol{h}_t} + (1-\lambda)\sum_{n=1}^{\infty}\lambda^{n-1}\gamma^n g(\boldsymbol{h}_{t+n};\boldsymbol{\theta}_{t+n-1})^\top \frac{\partial \boldsymbol{h}_{t+n}}{\partial \boldsymbol{h}_t} \qquad \text{Lemma C.1} \\
&= \frac{\partial L_{t+1}}{\partial \boldsymbol{h}_t} + \lambda\gamma\left[\sum_{n=2}^{\infty}\lambda^{n-2}\gamma^{n-2}\frac{\partial L_{t+n}}{\partial \boldsymbol{h}_{t+1}}\right]\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} + (1-\lambda)\gamma g(\boldsymbol{h}_t;\boldsymbol{\theta}_t)^\top \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} \\
&+ \lambda\gamma\left[(1-\lambda)\sum_{n=2}^{\infty}\lambda^{n-2}\gamma^{n-1} g(\boldsymbol{h}_{t+n};\boldsymbol{\theta}_{t+n-1})^\top \frac{\partial \boldsymbol{h}_{t+n}}{\partial \boldsymbol{h}_{t+1}}\right]\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} \\
&= \frac{\partial L_{t+1}}{\partial \boldsymbol{h}_t} + \lambda\gamma\left[\sum_{n=2}^{\infty}\lambda^{n-2}\gamma^{n-2}\frac{\partial L_{t+n}}{\partial \boldsymbol{h}_{t+1}} + (1-\lambda)\sum_{n=2}^{\infty}\lambda^{n-2}\gamma^{n-1} g(\boldsymbol{h}_{t+n};\boldsymbol{\theta}_{t+n-1})^\top \frac{\partial \boldsymbol{h}_{t+n}}{\partial \boldsymbol{h}_t}\right]\frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} \\
&+ (1-\lambda)\gamma g(\boldsymbol{h}_{t+1};\boldsymbol{\theta}_t)^\top \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} \\
&= \frac{\partial L_{t+1}}{\partial \boldsymbol{h}_t} + \gamma\lambda(\boldsymbol{G}_{t+1}^\lambda)^\top \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} + \gamma(1-\lambda) g(\boldsymbol{h}_{t+1};\boldsymbol{\theta}_t)^\top \frac{\partial \boldsymbol{h}_{t+1}}{\partial \boldsymbol{h}_t} \qquad \square
\end{aligned}$$

D Experimental Details

For the experiments which analyse the alignment of synthetic gradients to true gradients (Figures 2 and S1), we use an RNN with a linear activation function. For the experiments for which we train the RNN using synthetic gradients (Figures 4 and 5), we use LSTM units in our RNN architecture (Hochreiter & Schmidhuber, 1997).

The model output for a given task is a (trained) linear readout of the RNN (output) state. The synthesiser network performs a linear operation (with a bias term) on the RNN hidden state (concatenation of cell state and output state for LSTM) to produce an estimate of its respective loss gradient. The synthetic gradient at the final timestep is defined as zero. As in Jaderberg et al. (2017), the synthesiser parameters θ are initialised at zero.

When truncated BPTT with truncation size n is used for a task sequence length T, the sequence is divided such that the first truncation is of size R where $R = T \mod n$ and each following truncation is of size n.

For (accumulate) BP(λ) we often find it empirically important to use a discount factor $\gamma < 1$. Except in the experiments using fixed RNNs (Fig. 2) for which $\gamma = 1$, for all conditions we set $\gamma = 0.9$.

We often observe it necessary to scale down the synthetic gradient before it is received by the RNN. We find this particularly necessary when using synthetic gradients alongside truncated BPTT, and in this case generally find a scaling factor 0.1 optimal as in Jaderberg et al. (2017). We find that BP(λ) is less sensitive to this condition and choose scaling factors depending on the task (see below).

Data is provided to the model in batches. Though it is in principle possible to update the model as soon as the synthetic gradient is provided (e.g. every timestep), we find that this can lead to unstable learning, particularly when explicit supervised signals are sparse (as in the sequential-MNIST task). For this reason we instead accumulate gradients over timesteps and update the model at the end of the batch sequence. We use an ADAM optimiser for gradient descent on the model parameters (Kingma & Ba, 2014).

All experiments are run using the PyTorch library. Code will be released after the anonymous peer review.

The toy task experiments used to analyse gradient alignment were conducted with an Intel i7-8665U CPU, where each run with a particular seed took approximately one minute. The sequential MNIST task and copy-repeat tasks were conducted on NVIDIA GeForce RTX 2080 Ti GPUs. Each run in the sequential MNIST task took approximately 3 hours or less (depending on the model used); each run in the copy-repeat task took approximately 12 hours or less.

D.1 Toy task used to analyse gradient alignment

Fo analyse gradient alignment for fixed RNNs in the toy task we provide the model an input at timestep 1 and the model is trained to produce a target 2-d coordinate on the unit-circle at the final timestep T where $T = 10$. The input is a randomly generated binary vector of dimension 10. The task error is defined as the mean-squared error between the model output and target coordinate.

We also consider plastic RNNs which are themselves updated at the same time as the synthesiser. To ensure that the RNN has to discover the temporal association between input and target, and not simply readout a fixed value, in this case we consider 3 input/target pairs, where the targets are spread equidistantly on the unit circle. For this case we are interested in the limits of temporal association that can be capture and consider increasing sequence lengths T; specifically, we consider T as multiples of 10 up to 100. We deem a sequence length solved by a model with a certain initialisation (i.e. a given seed) if the model is able to achieve less that 0.025 – averaged over the last 20 of 250 training epochs – for that sequence length and all smaller sequence lengths. To improve readability, for this task the training curves are smoothed using a Savitzky–Golay filter (with a filter window of length 25 and polynomial of order 3).

For this task we provide the model in batches of size 10, where 1 epoch involves 100 batches. The number of RNN units is 30 and the initial learning rate for the synthesiser is set as 1×10^{-4} and 1×10^{-3} for the fixed and plastic RNN cases respectively.

D.2 Sequential-MNIST task

For the sequential-MNIST task we present a row-by-row presentation of a given MNIST image to the model (Deng, 2012; Le et al., 2015). That is, the task is divided into 28 timesteps where at the ith timestep the ith row of 28 pixels is provided to the model. At the end of the presentation, the model must classify which digit the input represents. The task loss is defined as the cross entropy between the model output and the target digit.

For this task we provide the model in batches of size 50 with an initial learning rate of 3×10^{-4}. The number of hidden LSTM units is 30. For BP(λ) the synthetic gradient is scaled by a factor of 0.1.

During training, the models with the lowest validation score over 50 epochs are selected to produce the final test error.

D.3 Copy-repeat task

For the copy-repeat task (Graves et al., 2014) the model receives a delimiter before an 8-dimensional binary pattern of length N and then a repeat character R The model must then repeat the binary pattern R times followed by a stop character. The total sequence length is therefore $N \times (R+1)+3$. For easier absorption R is normalised by 10 when consumed by the model.

We follow the curriculum in Jaderberg et al. (2017) and alternatively increment N and R when a batch average less than 0.15 bits is achieved.

For this task we provide the model in batches of size 100. For the RNN and readout parameters we use an initial learning rate of 1×10^{-3}, whilst we find a smaller learning rate of 1×10^{-5} for the synthesiser parameters necessary for stable learning. The number of hidden LSTM units is 100. For BP(λ) we do not apply scaling to the synthetic gradient.

E Supplementary figures

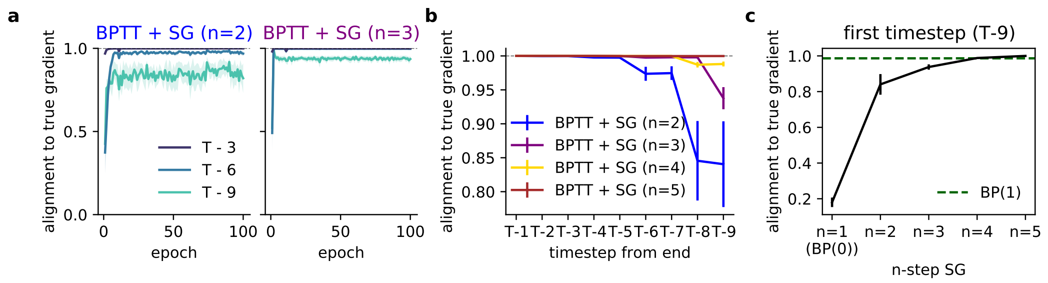

Figure S1: Learning synthetic gradients with the n-step synthetic gradient (Eq. 14) as in (Jaderberg et al., 2017) in the toy task; inputs are provided at timestep 1 and the corresponding target is only available at the end of the task at time $T = 10$. (**a**) Alignment between synthetic gradients and true gradients for a fixed RNN model across different timesteps within the task, where the synthetic gradients are learned with BPTT truncation size $n = 2$ (left) and $n = 3$ (right). Alignment is defined using the cosine similarity metric. (**b**) The average alignment over the last 10% of epochs in **a** across all timesteps. (**c**) Alignment of synthetic gradients at the first timestep of the task for different n; BP(1) (dotted green) is shown for reference. Results show average (with SEM) over 5 different initial conditions.

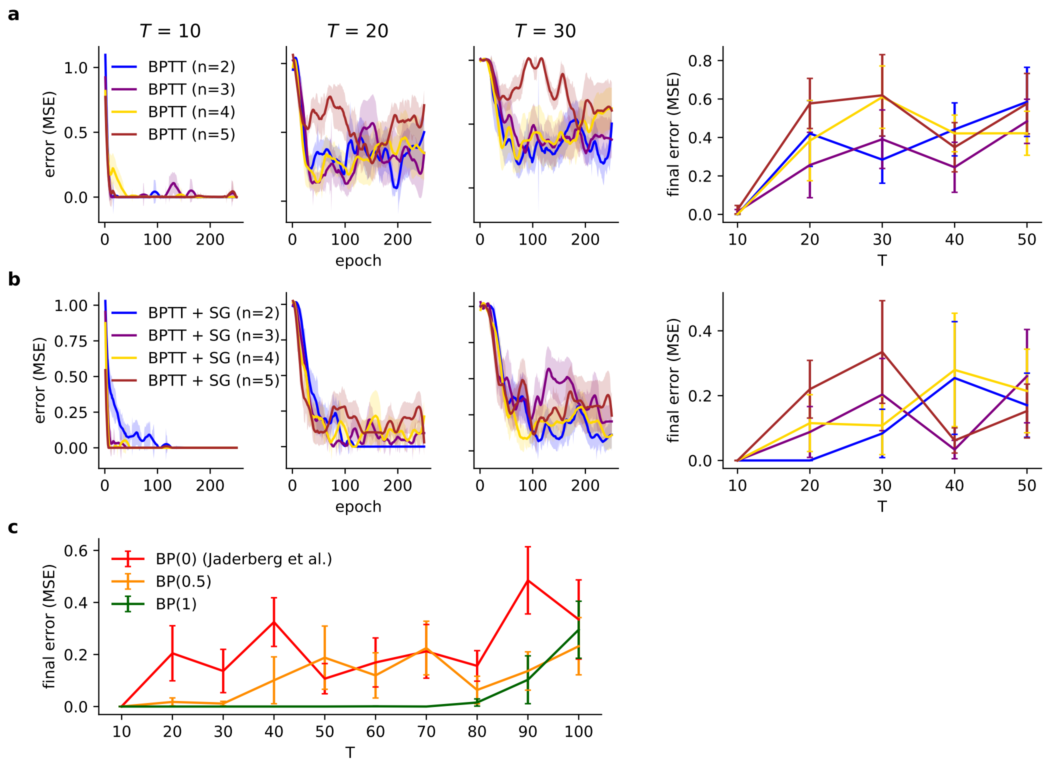

Figure S2: RNN learning using n-step synthetic gradients in the toy task (**a**) (Left) Learning curves of RNNs which are updated using n-step truncated BPTT over different sequence lengths T. (Right) Task error at end of training for increasing T. (**b**) Same as (a) but n-step synthetic gradients are also applied. (**c**) Task error for RNNs updated using BP(λ) at end of training for increasing T. Results show average (with SEM) over 5 different initial conditions.