KERNEL RECURRENT LEARNING (KERL)

Anonymous authors
Paper under double-blind review

ABSTRACT

We describe Kernel Recurrent Learning (KeRL), a reduced-rank, temporal eligibility-trace based approximation to backpropagation through time (BPTT) for training recurrent neural networks (RNNs) that gives competitive performance to BPTT on long time-dependence tasks. The approximation replaces the rank-4 credit assignment tensor by a reduced-rank product of a sensitivity weight and a temporal sensitivity kernel. In this structured approximation motivated by node perturbation, sensitivity weights and relevant time scales are learned by applying perturbations. The rule represents another step toward biologically plausible or neurally inspired ML, with relaxed architectural requirements (no symmetric return weights), a smaller memory demand (no unfolding and storage of states over time), and a shorter feedback time.

1 INTRODUCTION

Animals and humans excel at learning tasks that involve long-term temporal dependencies. A key challenge of learning such tasks is the problem of spatiotemporal credit assignment: the learner must find which of many past neural states is causally connected to the currently observed error, then allocate credit across neurons in the brain. When the time-dependencies between network states and errors are long, learning becomes difficult.

In machine learning, the current standard for training recurrent architectures is *Backpropagation Through Time* (BPTT, Rumelhart et al. (1985), Werbos (1990)). BPTT assigns temporal credit or blame by unfolding a recurrent neural network in time up to a horizon length T, processing the input in a forward pass, and then backpropagating the error back in time in a backward pass (see Fig 1a).

From a biological perspective, BPTT – like backpropagation in feedforward neural networks – is implausible for many reasons. For each weight update, BPTT requires using the transpose of the recurrent weights to send the error backwards. This requires that the network either uses two way synapses, or uses a symmetric copy of the feedforward weights to backpropagate the error. In either case, the network must alternatingly gate its dynamical process to run forwards or backwards depending on whether activity or errors are being sent through the network. Second, the network must remember all its previous states, going T steps back in time, placing a heavy demand on memory. Third, the time-complexity of computation of the gradient scales like T, making BPTT slow for learning long time scale dependencies. For unbiased gradient learning, T should match the length of the task or the maximum temporal lag between network states and errors, but in practice T is truncated to mitigate computational costs, introducing a bias.

Some of these problems can be alleviated. The backpropagation phase can be avoided in algorithms such as Real Time Recurrent Learning (RTRL, Williams & Zipser (1989)) and Unbiased Online Gradient Optimization (UORO, Tallec & Ollivier (2017), Ollivier et al. (2015)), which keep track of the credit assignment tensor in a feedforward way. Additionally, it has been shown that in feedforward networks at least, random feedback connections for backpropagation can replace the symmetric return connections (Lillicrap et al. (2016) and Nøkland (2016)). Finally, it has been argued that neurons may encode error information in the time derivative of their firing rates using an STDP-like learning rule (Bengio et al. (2015)), and that gradient updates can be computed as a relaxation to equilibrium (Scellier & Bengio (2017)). This scheme may allow the forward and backward phases to exist more harmoniously.

The present work is another step in the direction of providing heuristics and relaxed approximations to backpropagation-based gradient learning for recurrent networks . KeRL confronts the problems

of efficiency and biological plausibility. It avoids the lengthy backpropagation phase by imposing temporal structure on the sensitivity tensor, allowing the time-dependent part of the calculation to proceed in a feedforward way. Instead of storing all past states in memory, synapses integrate over past states during the forward pass (see Fig 1b). Furthermore, an asymmetric structure naturally emerges from KeRL, in which the feedback path is much shorter than the feedforward path.

We implemented KeRL and BPTT in both a batched and online setting. In the batched setting, data is prepared into batches of fixed sequence length T and fed into a T step unrolled graph. Using BPTT, the error is backpropagated through the entire graph. BPTT should outperform KeRL in these scenarios, as it computes the exact gradient with respect to the information supplied. Furthermore, when error is only supplied at the end of an episode, there is no computational advantage to computing the gradient forwards, as done in KeRL.

In the online setting, the network is fed a constant stream of data (batch size 1) and receives an error signal at each time step. In this setup, BPTT has to truncate the gradient for computational expediency whereas KeRL can utilize information over long time scales. In our experiments, online-KeRL runs at a speed comparable to few step truncated online-BPTT. These results lead us to believe that KeRL would be most effective in a data-rich setting, where it is more advantageous to process the data quickly than it is to utilize the full informational content. Such data rich environments are encountered frequently in the real world, but rarely in typical RNN tasks.

2 OTHER METHODS

Truncated BPTT is the standard method for training RNNs. Typically the data is sliced up into minibatches of sequence length T, and then fed into a computational graph of the same length. This places an artificial limit on the RNN memory, and introduces biases into the gradients. Furthermore, BPTT suffers from the vanishing gradient problem for large horizon length T (Pascanu et al. (2013)), especially under squashing non-linearities. KeRL alleviates these difficulties by learning the appropriate timescales for interneuron sensitivity, then regularizing the network towards a solution with appreciable interneuron interactions on long timescales.

RTRL keeps track of the credit assignment tensor in a feedforward way, eliminating the need for a separate backward pass as in BPTT. However, this requires the network to keep track of a rank-3 tensor, resulting in a complexity of $\mathcal{O}(N^4)$ per timestep. UORO alleviates these computational difficulties by factoring the RTRL tensor into lower-rank approximations, but requires taking vector norms at each step. Both of these methods require non-local computations that could not be done on known biological hardware. Finally Decoupled Neural Interfaces, (DNI Jaderberg et al. (2016)) creates a synthetic gradient by using a separate RNN to continually predict the future loss with respect to the hidden state.

Like DNI, KeRL computes an approximate gradient, using a different set of parameters to learn about the network dynamics. Additionally, DNI and KeRL both compute the gradient using only local communication. However, KeRL is distinguished by its simplicity and biological plausibility. Instead of using a separate network, synapses computes their updates by integrating over their presynaptic input. The learned parameters are intuitive: a set of timescales to describe the memory of each neuron, and a set of sensitivity weights to describe how strongly the neurons interact on average.

3 THE LEARNING RULE

Consider a single-layer RNN in discrete time (indexed by t), with inputs $\mathbf{x}^t$, hidden unit activations $\mathbf{h}^t$, and a readout layer $\mathbf{y}^t$. The dynamics of the recurrently connected hidden units are given by:

$$\mathbf{h}^{t+1} = \sigma\left(\mathbf{net}^{t+1}\right) = \sigma\left(W^{rec}\mathbf{h}^t + W^{in}\mathbf{x}^t + \mathbf{b}\right), \tag{1}$$

where W^{rec}, W^{in} are the recurrent and input weights, $\mathbf{b}$ are the hidden biases, σ is a general pointwise non-linearity, and net_j^t represents the net input to neuron j at time t. The readout is given by $\mathbf{y}^t = \sigma^{out}\left(W^{out}\mathbf{h} + \mathbf{b}^{out}\right)^t$. The objective function is $C = C(\mathbf{y}^T, \hat{\mathbf{y}}^T)$ in the case where error feedback is received at the end of an episode and $C = \sum_{t=0}^{T} C^t$ when errors $C^t = C(\mathbf{y}^t, \hat{\mathbf{y}}^t)$

are received throughout an episode. $\hat{\mathbf{y}}^t$ are the output targets, used when available to evaluate the objective function. The parameters $W^{in}, W^{rec}, W^{out}, \mathbf{b}, \mathbf{b}^{out}$ are trainable.

Assuming the readout weights and biases are trained in a typical fashion, the parameters $W = \{W^{in}, W^{rec}, \text{and } \mathbf{b}\}$ of the RNN are trained with the following learning rule:

$$\Delta W^t_{jk} = -\eta \sum_i \delta^t_i \beta_{ij} e^t_{jk} \tag{2}$$

In this description $\delta^t_i \equiv \frac{dC}{dh^t_i} = \sum_l \frac{dC}{dy^t_l} W^{out}_{li}$ represents the gradient of the cost with respect to the current hidden state, β_{ij} is a set of learned sensitivity weights, and $e^t_{jk} \equiv \sum_\tau e^{-\gamma_j \tau} \frac{\partial h^{t-\tau}_j}{\partial W_{jk}} = \sum_\tau e^{-\gamma_j \tau} \sigma'(\text{net}^{t-\tau}_j) s^{t-\tau}_k$ is an eligibility-like trace (Fiete & Seung (2006)) that describes how strongly synapse W_{jk} contributed to the current error based on how far in the past it was active. Here $s^{t-\tau}_k = \{x^{t-\tau}_k, h^{t-\tau-1}_k, \text{and } \delta^{\text{dirac}}_{jk}\}$ stands in for the presynaptic input to the parameter being updated. The sensitivity weights, β_{ij} and the kernel time-constants, γ_j of the eligibility are learned parameters.

These parameters are updated at the same time as the W's, by tracking the effect of i.i.d. hidden perturbations ξ during the forward pass. In order to do so our hidden neuron must store two values, the true hidden state $\mathbf{h}$, and a perturbed hidden state $\tilde{\mathbf{h}}$ generated by applying noise to the neurons during the forward pass.

$$\tilde{\mathbf{h}}^{t+1} = \sigma\left(W^{rec}(\tilde{\mathbf{h}}^t + \xi^t) + W^{in}\mathbf{x}^t + \mathbf{b}\right), \tag{3}$$

We update the sensitivity parameters as follows:

$$\begin{aligned} \Delta\beta^t_{ij} &= -\eta_2 \delta^t_{\beta_i} \Omega^t_j(\xi) \\ \Delta\gamma^t_j &= -\eta_3 \sum_i \delta^t_{\beta_i} \beta_{ij} \Gamma^t_j(\xi) \end{aligned} \tag{4}$$

Here ${\delta_{\beta_\text{i}}}^t \equiv \sum_j \beta_{ij}\Omega^t_j - (\tilde{h}^t_i - h^t_i)$ represents the error in reconstructing the effect of the perturbation via the sensitivity weights and $\Omega^t_j(\xi) \equiv \sum_\tau e^{-\gamma_j \tau} \xi^{t-\tau}_j$ and $\Gamma^t_j(\xi) \equiv \frac{d\Omega^t_j}{d\gamma_j} = -\sum_\tau \tau e^{-\gamma_j \tau} \xi^{t-\tau}_j$ are integrals that neuron h_j performs over the applied perturbation ξ.

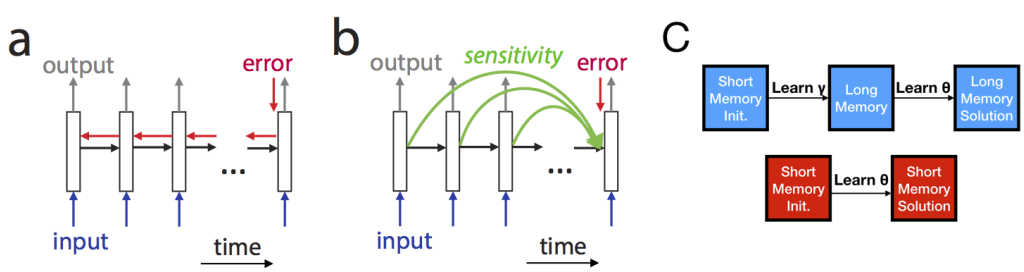

Figure 1: a), b) Schematic depicting the difference between the flow of information in a) BPTT, b) KeRL. c) Schematic depicting how KeRL (Blue) can be used to find long term memory solutions when BPTT (red) struggles

4 ANSATZ THAT LEADS TO THE LEARNING RULE

In the following we describe the approximations that reduce full gradient computations, via back-propagation through time, to the KeRL rule. Using BPTT derivatives with respect to the cost function are calculated as follows.

$$\frac{dC}{dW_{jk}} = \sum_i \delta^t_i \frac{dh^t_i}{dW_{jk}} \tag{5}$$

where δ_i^t is the error with respect to the current hidden state as defined earlier. We have introduced the credit assignment tensor $\frac{dh_i^t}{dW_{jk}}$ which describes how the current hidden state would have been affected by changing the parameters W throughout the task. We can write out the credit assignment as a sum of temporal variations:

$$\frac{dh_i^t}{dW_{jk}} = \sum_{\tau=0} \frac{\partial h_i^t}{\partial h_j^{t-\tau}} \frac{\partial h_j^{t-\tau}}{\partial W_{jk}} \tag{6}$$

The sensitivity, $\frac{\partial h_i^t}{\partial h_j^{t-\tau}}$, describes all pathways through which neuron h_j effects neuron h_i with time lag τ. The term $\frac{\partial h_j^{t-\tau}}{\partial W_{jk}}$ describes the direct dependence of the next hidden state on the parameters W. We make the following Ansatz for the sensitivity:

$$\frac{\partial h_i^t}{\partial h_j^{t-\tau}} = \beta_{ij} e^{-\gamma_j \tau} \tag{7}$$

In other words, we assume that neuron j affects neuron i with some weight given by β_{ij} and through some time scale given by γ_j. While the neurons surely interact in a more complicated way, we can learn the parameters β and γ that best encompass these interactions, allowing us to compute an approximate gradient. By making this approximation, we are able to factor our three-index credit assignment tensor into the product of two-index tensors

$$\frac{dh_i^t}{dW_{jk}} = \beta_{ij} e_{jk}^t \tag{8}$$

where β_{ij} and e_{jk}^t are the sensitivity weights and the eligibility trace as defined earlier. This formulation leads to Eqn. 2. Now we propose to learn a β, and γ that align the Ansatz gradient with the true gradient. Consider running two networks side by side, one with noise ξ^τ (Eqn. 3) and one without (Eqn. 1). Provided the perturbations are small, we expect that their effect on the hidden state is given by

$$\tilde{h}_i^t - h_i^t = \sum_{\tau,j} \frac{dh_i^t}{dh_j^{t-\tau}} \xi_j^{t-\tau} \tag{9}$$

Now substituting in our Ansatz (Eqn. 7) we can learn the parameters β, and γ via the objective function.

$$C_{\beta,\gamma} = \sum_i (\tilde{h}_i^t - h_i^t - \sum_{\tau,j} \beta_{ij} e^{-\gamma_j \tau} \xi_j^{t-\tau})^2 \tag{10}$$

We change β and γ to minimize this objective function. This leads us to the results shown in Eqn. 4. We outline the training procedure in Alg. 1:

5 Empirical results

In this section we show that KeRL is competitive with BPTT across several benchmark tasks for RNNs. Additionally we show that KeRL can be implemented online and gives competitive results with BPTT and UORO.

We implemented batch SGD of both KeRL and BPTT on a one-layer RNN with a tanh non-linearity, and a one-layer IRNN (Le et al. (2015)). These networks are trained on the adding problem (Hochreiter & Schmidhuber (1997), Hochreiter et al. (2001)) and pixel-by-pixel MNIST (LeCun et al. (1998)) tasks. Additionally we implemented an online version of KeRL with an LSTM which we used on the A^n, B^n task (Gruslys et al. (2016)).

For BPTT, we tuned learning rate, η, and gradient clipping, gc (Pascanu et al. (2013)). For KeRL, we also separately varied the learning rate of the sensitivity weights and kernels, $\eta_2 = \eta_3 \neq \eta$. Additionally, we tried both RMSprop (Tieleman & Hinton) and Adam (Kingma & Ba (2014)) algorithms. In practice we found that the same hyperparameter settings η, gc tended to work well for both BPTT and KeRL. The additional hyperparameter for KeRL, η_2, was robust across several orders of magnitude.

Algorithm 1 Pseudocode table describing the implementation of KeRL on an RNN

while $t < T$ **do**
$\mathbf{h}^t \leftarrow \sigma\left(W\mathbf{h}^{t-1} + W^{in}\mathbf{x}^{t-1} + \mathbf{b}\right)$ /* propagate data forwards */
$\tilde{\mathbf{h}}^t \leftarrow \sigma\left(W(\tilde{\mathbf{h}}^{t-1} + \xi^{t-1}) + W^{in}\mathbf{x}^{t-1} + \mathbf{b}\right)$ /*propagate noisy network forwards */
${\Omega_j}^t \leftarrow e^{-\gamma_j}{\Omega_j}^{t-1} + {\xi_j}^t$ /*Integrate over perturbations*/
${\Gamma_j}^t \leftarrow e^{-\gamma_j}{\Gamma_j}^{t-1} - e^{-\gamma_j}{\Omega_j}^{t-1}$ /*Derivative of $\mathbf{\Omega}^t$ w.r.t. γ^t */
$e^t_{jk} \leftarrow e^{-\gamma_j}e^{t-1}_{jk} + \sigma'(\text{net}^t_j)s^t_k$ /*Integrate over input term*/
$\delta^t_{\beta_i} \leftarrow \sum_j \beta_{ij}\Omega^t_j - (\tilde{h}^t_j - h^t_j)$ /* Calculate error in predicting effect of perturbations */
$\beta^t_{ij} \leftarrow \beta^{t-1}_{ij} - \eta_2\delta^t_{\beta_i}\Omega^t_j$ /*Update sensitivity weights */
$\gamma^t_j \leftarrow \gamma^{t-1}_j - \eta_3\sum_i \delta^t_{\beta_i}\beta_{ij}\Gamma^t_j$ /*Update kernel coefficients */
$\delta^t_i = \sum_l \frac{dC}{dy^t_l}W^{out}_{li}$ /*Compute error in hidden state */
$W^t_{jk} \leftarrow W^{t-1}_{jk} - \eta\sum_i \delta^t_i\beta_{ij}e^t_{jk}$ /*Compute and apply gradients*/
end while

5.1 Adding Problem

We first trained our network on the adding problem, a task that was originally deemed to be very hard for RNNs. In this problem, the network is given two input streams. One is a sequence of randomly chosen numbers between zero and one, and the second is mask vector which is one for two randomly selected entries, and zero for all of the others. For long sequence lengths, this task is extremely difficult, as it requires the network to remember sparse pieces of information over long time scales, while ignoring long sequences of noise.

0.5	0.3	0.8	0.1	0.4	0.3
0	1	0	1	0	0

Table 1: Inputs for the adding problem. The desired output, obtained by summing the entries in the top row only when the entries in the bottom row are 1, is 0.4 in this example.

Here we tested learning rates η and gradient clippings gc over several orders of magnitude $\eta = \{1e-03, 1e-04, 1e-05, 1e-06, 1e-07\}$, $gc = \{1, 10, 100\}$ as well as two optimizers, Adam (Kingma & Ba (2014)) and RMSProp (Tieleman & Hinton). We then tuned $\eta_2 = \{1e-03, 1e-04, 1e-05, 1e-06, 1e-07\}$ on KeRL to get the best possible result. We initialized our kernels $\gamma_j = (\frac{1}{\text{sequence length}})^{n_j}$ where the $n_j \sim U([0, 2])$ are sampled uniformly and independently. We found that KeRL is relatively robust across η_2.

We tested for a variety of sequence lengths up to 400. We found that a IRNN trained with BPTT was able to perform extremely well on this task, but a RNN with tanh nonlinearity struggled to learn longer sequence lengths due to the vanishing gradient problem. While KeRL was unstable on the IRNN, it outperformed BPTT with the tanh nonlinearity (Fig. 2).

Learning Rule-Nonlinearity	Algorithm	Learning Rate	Gradient Clipping	Feedback Learning Rate
BPTT-tanh	RMS Prop	10^{-3}	100.0	N/A
BPTT-IRNN	RMS Prop	10^{-4}	100.0	N/A
KeRL-tanh	RMS Prop	10^{-3}	100.0	10^{-5}

Table 2: Hyperparameters for adding problem

We believe that KeRL outperforms BPTT on the tanh nonlinearity because our Ansatz forces the interneuron sensitivity $\frac{d\mathbf{h^t}}{d\mathbf{h^{t-\tau}}}$ to have relatively long timescales. We expect that by applying gradients generated by our Ansatz (instead of the true gradients) we push our network towards a solution with longer time scales via a feedback alignment mechanism (Lillicrap et al. (2016)). We diagram the idea in part c) of Fig. 1.

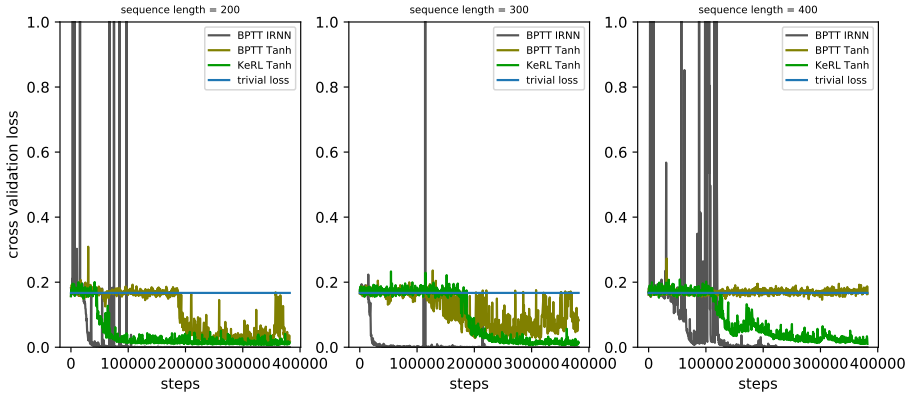

Figure 2: Single-trial example of cross validation loss on the adding problem for sequences of length 200,300,400.

To further investigate the importance of learning these timescales we tried implemented KeRL without training the coefficients γ. As it turned out, the network fared much worse without learning the time scales.

Seq. Len.	Fix β, Fix γ	Learn β, Fix γ	Fix β, Learn γ	Learn β, Learn γ	BPTT
200	0.014	.170	.031	0.008	0.020
400	.180	.171	0.076	0.031	.171

Table 3: Cross validation loss on adding problem after 30,000 training steps using several variants of KeRL and BPTT

We can see that learning the timescales appears to be even more important than learning the sensitivity weights. Also, we note that even when the sensitivity weights and kernels are held fixed, KeRL is able to perform the task for the shorter sequence of 200. This implies that a feedback-alignment-like mechanism may be doing some of the gradient alignment.

5.2 Pixel-by-pixel MNIST

For our second task we study pixel-by-pixel MNIST (LeCun, 1998). Here the RNN is given a stream of pixels left-to-right, top-to-bottom for a given handwritten digit from the MNIST data set. At the end of the sequence, the network is tasked with identifying the digit that it was shown. This problem is difficult, as the RNN must remember over an extremely long sequence length of 784 pixels. Again, we tuned over the same hyperparameters as in the adding problem, looking at the performance after $100,000$ training steps. We find that neither KeRL nor BPTT worked well with a tanh nonlinearity, but both were able to perform relatively well on an IRNN as seen in Fig. 3. Here KeRL preferred a slightly lower learning rate η than BPTT.

Additionally, we see in Fig. 3 that the BPTT gradients and the KeRL gradient are very strongly aligned during training. As a result we expect that

$$\Delta C = \frac{dC}{dW_{jk}} \cdot (\Delta W_{jk})_{KeRL} = -\eta * c * |\frac{dC}{dW_{jk}}|^2 < 0 \tag{11}$$

where c is the alignment between the two gradients. So given a small enough learning rate, we expect KeRL to demonstrate hill climbing.

While the KeRL algorithm is able to learn almost as quickly on pixel-by-pixel MNIST, it does not converge to as good of an optimum. Still, it performs reasonably well relative to BPTT on the task.

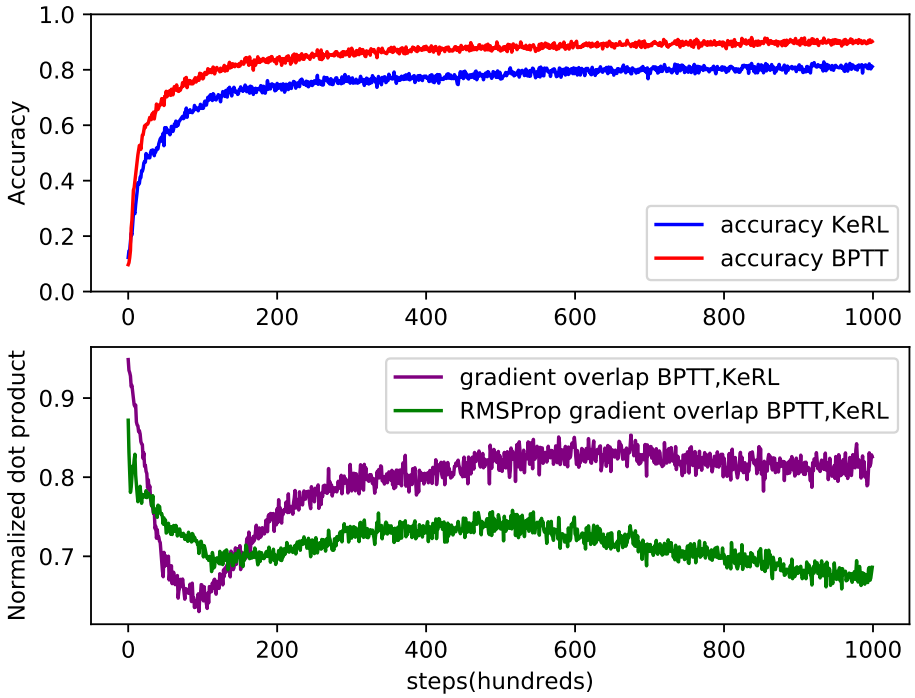

Figure 3: Cross validation accuracy on pixel-by-pixel MNIST using BPTT (red) and KeRL(blue). Normalized dot product between gradients (purple) and RMSProp gradients (green) computed by KeRL and BPTT

Learning Rule	Algorithm	Learning Rate	Gradient Clipping	Feedback Learning Rate
BPTT	RMSProp	10^{-5}	100.0	N/A
KeRL	RMSProp	10^{-6}	100.0	10^{-8}

Table 4: Hyperparameters for pixel-by-pixel MNIST

5.3 Online KeRL

While KeRL is comparable in speed to BPTT for batched learning, we expect it to be significantly faster for online learning. For non-truncated BPTT, we expect that for each step forwards, we need to send the error information all the way backwards. So we expect computation of the gradients to scale roughly like MT^2 where M is the time per matrix multiplication. For truncated BPTT with truncation length $S < T$, we expect computation of the gradient to scale like MST. However, for KeRL we only need to compute apply a few tensor operations at teach time step, so learning should scale like $O(1) * MT$. As a result online-KeRL should learn faster than online-BPTT for reasonably long truncation lengths.

We tested online KeRL against another online learning algorithm UORO on the A^n, B^n task. Here the network is asked to predict the next character in a stream of letters. Each block consists of sequence of n As followed by a line-break, then a sequence of n Bs followed by another line-break. The length of the sequences, n, is a randomly generated number in some range. The network cannot solve this task perfectly, as it can not predict the number of As before it has seen the sequence, but can do well by matching the number of As to the number of Bs. Here we generated n within the range $\{1, 32\}$. The minimum average bit-loss for this task is 0.14.

In order to compare with other results in the literature, we implemented KeRL in a LSTM layer, where we let $\mathbf{h}$ represent a concatenation of the hidden and cell state. We describe the details of the implementation in the appendix. For expediency, we used the same hyperparameters as in Tallec & Ollivier (2017), which involved decaying the learning rate in time. $\eta^t = \eta/(1+\alpha\sqrt{t})$. We tuned the feedback learning rate $\eta_2 = \eta_2^t/(1+\alpha\sqrt{t})$.

Algorithm-Optimizer	η	η_2	α
KeRL-Adam	10^{-3}	10^{-2}	0.03

Table 5: A^n, B^n hyperparameters

Algorithm	KeRL	1 Step BPTT	2 Steps BPTT	16 Step BPTT	UORO
Bit Loss	0.149	0.178	0.149	.144	.147

Table 6: Average cross-entropy bit-loss (across 10^4 training steps) on the online A^n, B^n task after 10^6 training steps

Aside from the KeRL value, these results are taken from Tallec & Ollivier (2017). With very little hyperparameter tuning, online KeRL is able to do very well on the A^n, B^n task, coming close to the minimum entropy. Although 17-step BPTT and UORO outperformed KeRL, we expect KeRL to be significantly faster (wall clock speed) in direct comparisons since it only requires a few tensor operations at each time step.

6 Conclusions, discussion & future work

In this paper we show that the KeRL algorithm, which calculates gradients on an RNN using a local learning rule without a backpropagation phase, is able to perform roughly comparably to BPTT on a range of tasks. KeRL is able to learn long-time dependencies on two hard RNN tasks. Furthermore, we present limited evidence that KeRL algorithm may combat the vanishing gradient problem by imposing a prior on the temporal sensitivity kernels of hidden neurons. Finally, we show that KeRL can be implemented online, and compares well with other online learning rules.

KeRL is a step towards biologically plausible learning. It eschews the segmented two phase backpropagation algorithm for a computation that is largely feedforward. It does not require the segmentation and storage of all past states, instead using an integrated activity or eligibility-like trace, and it gives rise to a naturally asymmetric structure that is more akin to the brain.

While we show empirically that KeRL performs hill-climbing, there is no guarantee that the gradients computed by KeRL are unbiased. In practice, we see that the KeRL gradients tend to align well with the BPTT gradients. In sacrificing accuracy, KeRL is able to gain in speed and biological plausibility.

In the future, we hope to show that KeRL is able to perform well on more realistic tasks. We hope this inspires more work on training RNNs with shorter, more plausible feedback paths.

References

Yoshua Bengio, Thomas Mesnard, Asja Fischer, Saizheng Zhang, and Yuhuai Wu. Stdp as presynaptic activity times rate of change of postsynaptic activity. *arXiv preprint arXiv:1509.05936*, 2015.

Ila R Fiete and H Sebastian Seung. Gradient learning in spiking neural networks by dynamic perturbation of conductances. *Physical review letters*, 97(4):048104, 2006.

Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. Memory-efficient backpropagation through time. In *Advances in Neural Information Processing Systems*, pp. 4125–4133, 2016.

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

Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pp. 87–94. Springer, 2001.

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. *arXiv preprint arXiv:1608.05343*, 2016.

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.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7: 13276, 2016.

Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in neural information processing systems*, pp. 1037–1045, 2016.

Yann Ollivier, Corentin Tallec, and Guillaume Charpiat. Training recurrent networks online without backtracking. *arXiv preprint arXiv:1507.07680*, 2015.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318, 2013.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in computational neuroscience*, 11:24, 2017.

Corentin Tallec and Yann Ollivier. Unbiased online recurrent optimization. *arXiv preprint arXiv:1702.05043*, 2017.

T Tieleman and G Hinton. Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. Technical report, Technical Report. Available online: https://zh. coursera. org/learn/neuralnetworks/lecture/YQHki/rmsprop-divide-the-gradient-by-a-running-average-of-its-recent-magnitude (accessed on 21 April 2017).

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

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

A Implementing KeRL on an LSTM

In this section we describe how to implement KeRL on an LSTM Hochreiter & Schmidhuber (1997) in more detail. The dynamics of the LSTM (without peepholes) are as follows

$$\begin{aligned}
\mathbf{i}^t &= \sigma(W^{ii}\mathbf{x}^t + W^{ih}\mathbf{h}^{t-1} + b^i) \\
\mathbf{f}^t &= \sigma(W^{fi}\mathbf{x}^t + W^{fh}\mathbf{h}^{t-1} + b^f) \\
\mathbf{g}^t &= \tanh(W^{gi}\mathbf{x}^t + W^{gh}\mathbf{h}^{t-1} + b^g) \\
\mathbf{o}^t &= \sigma(W^{oi}\mathbf{x}^t + W^{oh}\mathbf{h}^{t-1} + b^o) \\
\mathbf{c}^t &= \mathbf{f}^t\mathbf{c}^{t-1} + \mathbf{g}^t\mathbf{i}^\mathbf{t} \\
\mathbf{h}^t &= \mathbf{o}^t\tanh(\mathbf{c}^t)
\end{aligned} \tag{12}$$

where $\mathbf{h}^t$ is the hidden state, $\mathbf{c}^t$ is the cell state and $\mathbf{i}^t, \mathbf{f}^t, \mathbf{g}^t, \mathbf{o}^t$ are the input, forget, cell and output gates respectively. In order to implement KeRL we consider the total hidden state $\mathbf{H} = \{\mathbf{h}, \mathbf{c}\}$ to be a concatenation of the hidden and cell states. This a suitable choice, as the next total hidden state can be fully determined by the current total hidden state and the parameters of the network. We let the first n indices of $\mathbf{H}$ be the hidden state and the next n be the cell state. Derivatives with respect to the cost function are given by

$$\frac{dC}{d\theta} = \sum_{i,j} \frac{\partial C}{\partial H_i^t} \frac{\partial H_i^t}{\partial H_j^{t-\tau}} \frac{\partial H_j^{t-\tau}}{\partial \theta} \tag{13}$$

where θ stands in for the twelve trainable weights and biases. Our sensitivity Ansatz is $\frac{\partial H_i^t}{\partial H_j^{t-\tau}} = \beta_{ij} e^{-\gamma_j \tau}$. The input terms are now partial derivatives of this total hidden state with respect to the input parameters. As an example

$$\frac{\partial H_j^t}{\partial W_{lk}^{ii}} = \begin{cases} \frac{\partial h_j^t}{\partial W_{lk}^{ii}} & \text{if } j \leq n \\ \frac{\partial c_{j-n}^t}{\partial W_{lk}^{ii}} & \text{if } j > n \end{cases} = \begin{cases} 0 & \text{if } j \leq n \\ \frac{\partial c_{j-n}^t}{\partial W_{lk}^{ii}} & \text{if } j > n \end{cases} \tag{14}$$

where $\frac{\partial h_j^t}{\partial W^{ii}} = 0$ since the hidden state only depends on these parameters through the cell state. As earlier, we train our input weights and kernels by tracking the effect of applying perturbations during the forward pass. Our sensitivity weights β are a 2 x 2 array of matrices linking the cell and hidden states of the past to the current cell and hidden states. Since readout occurs from the hidden state, $\frac{\partial C}{\partial c_i^t} = 0$, and we only need to consider $\frac{\partial h_i^t}{\partial c_j^{t-\tau}} = \beta_{ij}^{hc} e^{-\gamma_j^c \tau}$ and $\frac{\partial h_i^t}{\partial h_j^{t-\tau}} = \beta_{ij}^{hh} e^{-\gamma_j^h \tau}$. The sensitivity weights β^{hc} , β^{hh} and time scales γ^c, γ^h can be learned as in the case of the simple recurrent network by applying perturbations $\xi^{h,t}, \xi^{c,t}$ to hidden and cell state and minimizing the cost function: $C_{\beta,\gamma} = \sum_i (\tilde{h}_i^t - h_i^t - \sum_{\tau,j} \beta_{ij}^{hc} e^{-\gamma_j^c \tau} \xi_j^{c,t-\tau} - \sum_{\tau,j} \beta_{ij}^{hh} e^{-\gamma_j^h \tau} \xi_j^{h,t-\tau})^2$. Our example gradient with respect to the input weights of the input gate is given by

$$\frac{dC}{dW_{jk}^{ii}} = \sum_i \frac{dC}{dh_i^t} \beta_{ij}^{hc} \sum_{\tau=0} e^{-\gamma_j^c \tau} g_j^{t-\tau} \sigma'(\text{net}_j^{t-\tau}) x_k^{t-\tau} \tag{15}$$

where net_j^t represents the presynaptic input to i_j^t. The other gradients can be calculated in an analogous manner.