Journal of Machine Learning Research 1 (2000) 1-48 Submitted 4/00; Published 10/00

Unbiased Scalable Online Recurrent Learning

Editor: Kevin Murphy and Bernhard Schölkopf

Abstract

Online scalable recurrent learning is challenging. Two popular gradient-based methods for recurrent learning are BPTT, and RTRL. BPTT looks at the complete sequence before computing gradients, and is unsuitable for online updates. RTRL can do online updates, but generally scales poorly with an increase in the number of parameters. In this paper, we propose two constraints that make RTRL scalable. We show that by either decomposing the network into independent modules, or learning a recurrent network incrementally and selectively, we can make RTRL scale linearly with the number of parameters without introducing bias, or increasing the variance of the gradient estimates. The two approaches result in different algorithms, that can be combined. We show the strengths and weaknesses of the proposed algorithms on two online prediction problems.

Keywords: Agent-state construction, online learning, scalable learning, recurrent learning, representation learning

1. Introduction

Learning by interacting with the world is a powerful framework for building systems that can autonomously achieve goals in complex worlds. A key ingredient for building autonomous systems is agent-state construction—learning a compact representation of the history of interactions that helps in predicting and controlling the future. One solution for state construction is to use differentiable recurrent neural networks (RNNs).

State construction using RNNs requires structural credit assignment—identifying how to change network parameters to improve predictions. In RNNs, a parameter can influence a prediction made many steps in the future. As a result, effective credit assignment requires tracking the influence of parameters on future predictions. Two popular algorithms for gradient-based structural credit assignment in recurrent neural networks are Back-propagation through time (BPTT) (Werbos, 1988; Robinson and Fallside, 1987) and real-time recurrent learning (RTRL) (Williams and Zipser 1989).

BPTT and RTRL are not suitable for online state construciton for large problems. BPTT requires storing all past activations for estimating the gradient. Additionally, it requires computation proportional to the length of the sequence seen so far. This makes BPTT unsuitable for online learning on long or never-ending streams of data. RTRL can estimate the gradient on the go, and does not require more computation as the length of the sequence grows. However, RTRL scales pooly with an increase in number of parameters of the RNN. Both BPTT and RTRL can be approximated, to make them more suitable for online learning.

Existing approximations to gradient-based learning approximate the estimate of the gradient, but keep the function class of the network the same. These approximations either introduce bias, which can result in poor performance or even divergence, or increase the variance of the estimate, making learning extremely slow. In this work, we propose a different strategy; instead of introducing bias in the estimate of the gradient, or increase the variance of the estimate, we instead limit the function class of the RNNs in a way that allows us to compute unbiased low-variance gradient estimate in a scalable way. Limiting the function class also introduces bias, however since learning is still done using the correct gradient, our method is less susceptible to instablity in learning.

2. Background

2.1 Recurrent learning

$$y(t) = \sum_{k=0}^{d} h_k(t) w_k(t) \tag{1}$$

where

$$h(t) = f_{\theta(t)}\left(h(t-1), x(t)\right) \tag{2}$$

here $\theta(t)$ are the parameters of the RNN at time t, whereas f_θ is a function that represents the RNN. Given this notation, we can write the gradient of a prediction $y(t)$ w.r.t the parameters θ as

$$\frac{\partial y(t)}{\partial \theta} = \frac{\partial y(t)}{\partial h(t)} \frac{\partial h(t)}{\partial \theta} \tag{3}$$

We can expand the second term as:

$$\frac{\partial h(t)}{\partial \theta} = \frac{\partial h(t)}{\partial \theta(t)} + \frac{\partial h(t)}{\partial h(t-1)} \frac{\partial h(t-1)}{\partial \theta} \tag{4}$$

to get

$$\frac{\partial y(t)}{\partial \theta} = \frac{\partial y(t)}{\partial h(t)} \left(\frac{\partial h(t)}{\partial \theta(t)} + \frac{\partial h(t)}{\partial h(t-1)} \frac{\partial h(t-1)}{\partial \theta} \right) \tag{5}$$

Note that we can use equation 4 to recursively expand equation 5, until we can get rid of $\partial \theta$ term. BPTT estimates the gradient of $y(t)$ by repeatedly using equation 4.

BPTT extends the back-propagation algorithm for feed-forward networks—independently proposed by Werbos (1974) and Rumelhart *et al.* (1986)—to RNNs by storing network activations from prior steps, and repeatedly applying the chain-rule starting from the output of the network and ending at the activations at the beginning of the sequence. More precisely, we can write the gradient of a prediction $y(t)$ w.r.t the parameters θ as:

where $h(t)$ is the internal state of the network at time t. Then, BPTT comput BPTT is unsuitable for online learning as it requires memory proportional to the length of the sequence. Moreover, it delays gradient computation until the end of the sequence. For online learning, this sequence can be arbitrarily long or never-ending.

RTRL—an alternative to BPTT—was proposed by Williams and Zipser (1989). RTRL relies on forward-mode differentiation—using chain-rule to compute gradients in the direction of time—to compute gradients recursively. Unlike BPTT, RTRL does not delay gradient-computation until the final step. The memory requirement of RTRL also does not depend on the sequence length. As a result, it is more suitable for real-time online learning.

Algorithm 1 Back-propagation through time

```
Require: $x_1, x_2, x_3, x_4, \cdots, x_n$
Require: $h_0$
Require: $\theta_1$
  $X \leftarrow x$
  $N \leftarrow n$
  for $t$ in $1, 2, 3, \cdots, n$ do
      $h_t = f_{\theta_t}(h_{t-1}, x_t)$
  end for
  $y_t = \sum_{k=0}^{d} w_k h_k^t$
  grad $= 0$
  for $t$ in $n, n-1, \cdots, 1$ do

  end for
```

Algorithm 2 Real-time recurrent learning

```
Require: $n \geq 0$
```

Unfortunately, RTRL requires maintaining the Jacobian $\frac{\partial h(t)}{\partial \theta}$ at every step, which requires $O(|h||\theta|)$ memory, where $|h|$ is the size of state of the network and $|\theta|$ is the number of total parameters. The Jacobian is recursively updated by applying chain rule as:

$$\frac{\partial h(t+1)}{\partial \theta} = \frac{\partial h(t+1)}{\partial \theta(t+1)} + \frac{\partial h(t+1)}{\partial h(t)} \frac{\partial h(t)}{\partial \theta},$$

which requires $O(|h|^2|\theta|)$ operations and scales poorly to large networks.

A promising direction to scale gradient-based credit-assignment to large networks is to approximate the gradient. Elman (1990) proposed to ignore the influence of parameters on future predictions completely for training RNNs. This resulted in a scalable but biased algorithm. Williams and Peng (1990) proposed a more general algorithm called Truncated BPTT (T-BPTT). T-BPTT tracks the influence of all parameters on predictions made up to k steps in the future. T-BPTT limits the BPTT computation to last k steps, and works well for a range of problems (Mikolov *et al.*, 2009, 2010; Sutskever, 2013 and Kapturowski *et al.*, 2018). Its main limitation is that the resultant gradient is blind to long-range dependencies. Mujika *et al.* (2018) showed that on a simple copy task, T-BPTT failed to learn dependencies beyond the truncation window. Tallec *et al.* (2017) demonstrated T-BPTT can even diverge when a parameter has a negative long-term effect on a target and a positive short-term effect.

RTRL can also be approximated to reduce its computational overhead. Ollivier *et al.* (2015) and Tallec *et al.* (2017) proposed NoBacktrack and UORO. Both of these algorithms provide stochastic unbiased estimates of the gradient and scale well. However, their estimates have high variance and require extremely small step sizes for effective learning. Cooijmans, Martens (2019) and Menick *et al.* (2021) showed that, for practical problems, UORO does not perform well compared to other biased approximations due to the high variance in its gradient estimates . Menick *et al.* (2021) proposed an approximation to RTRL called SnAp-k. SnAp-k tracks the influence of a parameter on a state only if the parameter

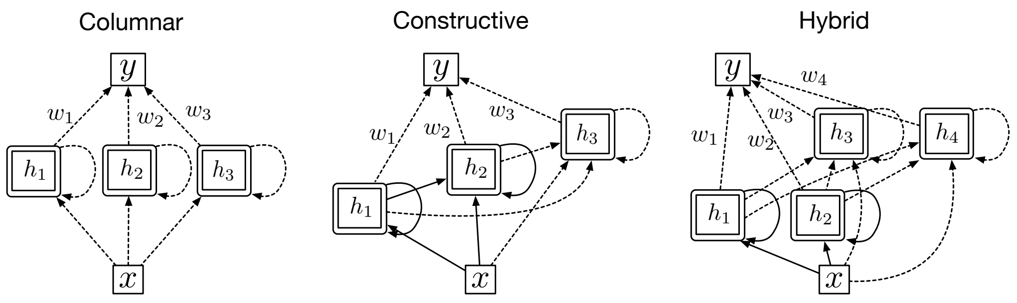

Figure 1: TODO: Make the caption concise. Three structures of recurrent neural networks that can be trained without truncation. Recurent networks with a columnar structure can be trained end-to-end using gradients without any truncation, only requiring $O(n)$ operations and memory per step. However, columnar networks do not have hierarchical recurrent features—recurrent features made out of other recurrent features. Constructive networks have hierarchical recurrent features, however must be trained incrementally to prevent bias. Incremental learning is achieved by initializing all w_i to zero, and learning h_1, h_2, and h_3 in order. Finally, columnar and constructive networks can be combined to get a hybrid network. The pairs (h_1, h_2) and (h_3, h_4) do not depend on each other, and can learn in parallel. Hoever, (h_3, h_4) must be learned after (h_1, h_2) have been learned and fixed.

can influence the state within k steps. It first identifies parameters whose influence on a state is zero for k steps and then assumes the future influence to be zero as well. For the remaining parameters, it tracks their influence on all future predictions. The bias introduced by SnAp-k is fundamentally different than the bias introduced by T-BPTT. SnAp-1 can be computationally efficient but introduces significant bias. SnAp-k for $k > 1$ reduces bias but can be as expensive as RTRL for dense RNNs. Menick *et al.* (2021) further proposed using sparse connections as a way to make SnAp more scalable. Connection sparsity reduces the number of parameters that can influence a state within k steps.

Among all the approximations, SnAp-1 is the only method that scales linearly with number of parameters, is capable of learning long-term dependencies, and

2.2 General value functions and

3. Problem Formulation

We formulate the problem as predicting the discounted sum of a cumulant from an online stream of data. More formally, the agent sees a feature vector $x(t) \in \mathcal{R}^d$ at timestep t and predicts the discounted sum of a cumulant y_t given by:

$$\begin{aligned} V(S_t) &= \mathbb{E}[G_t|S_t] \\ &= \mathbb{E}[\sum_{k=0}^{\infty} \gamma^k y(k)] \end{aligned} \tag{6}$$

This problem formulation can be used to represent temporal predictions, such as policy evaluation and general value functions. Additionally, by setting $\gamma = 0$, our problem formulation can represent classical supervised recurrent learning benchmarks. The goal of the agent is to minimize the mean squared error between the prediction $\hat{V}(x_{1:t})$ and $V(S)$. Note that for all our experiments, the data generation policy π is fixed and can be ignored.

3.1 Animal Learning Benchmarks

A suitable benchmark that fits this problem formulation is the trace patterning benchmark introduced by Rafiee (2020). In this benchmark, we are provided with a set of binary stimuli $o_t \in \mathbb{R}^d$ at each timestep t, which includes a set of n Conditioned Stimuli CS_t, one Unconditioned Stimuli US_t and m distractor signals. There are k random sets of configurations of CSs that will activate the US. If the active values of CS_t match one of these activation patterns, the value $\text{US}_{t+\text{ISI}}$ will be set to 1, where ISI (*inter-stimulus interval*) is a problem hyperparameter that is uniformly sampled from a certain range after every occurence of the US. In this problem, the goal is to learn eq. 6 using TD error where

$$y(k) = \text{US}_{t+k+1} \tag{7}$$

This benchmark is suitable for our evaluations since in order to do well, the learned model not only has to learn long-term temporal associations, it also has to identify the correct patterns of CSs that activate the US. Moreover, since this is a continuous online prediction problem, the algorithms need to scale well with time.

TODO: maybe we should also describe gamma here instead of the empirical evaluation section since it is also a problem parameter

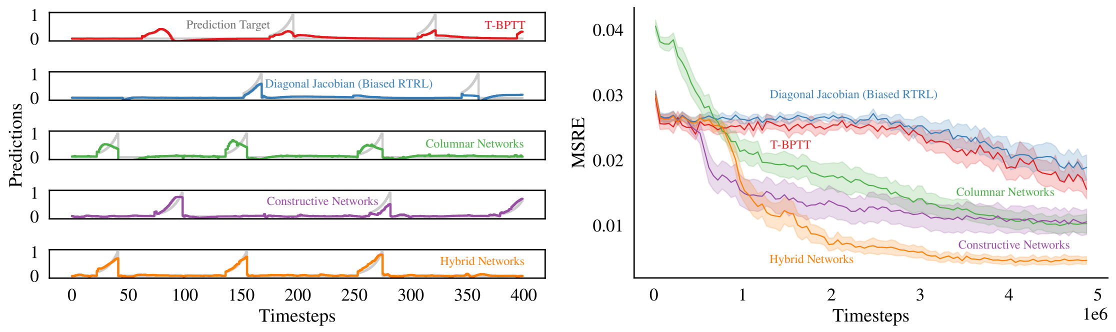

Figure 2: Left: Comparison of the predictions produced by the different methods for the last 400 timesteps of training. Right: Performance when an LSTM is trained using the proposed and other online baseline methods on the trace patterning benchmark (lower is better). All the runs are averaged using 30 seeds, and the error regions are the 95% confidence intervals. T-BPTT uses a truncation length k=27. We can see from this plot that the Diagonal Jacobian method performs the worst while the hybrid networks achieve the lowest error. TODO: T-BPTT is only 15 runs while the remaining curves are 30 runs. TODO: Maybe hybrid networks should get a better name so that people can refer to it easily?

TODO: maybe we should describe the ITI and how the probability of activation works.

4. Divergence for truncated-backprop and Snap-1

Because truncated backprop and Snap-1 use biased estimates of the gradients for learning updates, they can cause the prediction error and parameters to diverge under pathological cases. We present examples of divergence for both truncated-bptt and SnAp-1. In real-world case, such divergence happens rarely, if ever. However the examples still serve a useful pedagogical role, to clarify the limitation of the two algorithms.

For truncated backprop, we use a simple recurrent network with one hidden unit, one feature, and one prediction. The recurrence is defined as adding the value of the last hidden state to the current state. We fix the weight for the recurrent connection and the outgoing connection, and learn only the incoming weight. The optimal solution is $w_{in} = -10$. We let the recurrent network learn for a thousand steps using a step size of 10^{-3} and show the behavior pf th elearning algorithm in Figure . Note that the learning

5. Proposed methods

We propose imposing a specific structure on either the connections of the recurrent network, or the learning algorithms, to achieve scalable learning without truncation. We propose columnar networks to ac

6. Empirical evaluation

We first evaluate the performance of the proposed methods and baselines on the trace patterning benchmark described in section 3.1. In these experiments, we set ISI $=$ Unif$(14, 26)$, $\gamma = \frac{1}{1-\mathbb{E}[ISI]}$, $n = 6$ CSs, $k = 10$ activation patterns with a 0.5 probability of activation and $m = 5$ distractor signals. The details of the remaining hyper-parameters and their tuning is given in the Appendix **??**.

All of these methods are used to train a network with a single LSTM hidden layer, followed by a single output unit which predicts $y(k)$. The number of hidden recurrent units are adjusted for each method in order to ensure a fair comparison. This adjustment is done in order to ensure that the amount of computational resources consumed per step is the same for all the methods. A consequence of this constraint is that in the case of T-BPTT, we have to use a relatively smaller network since we need multiple passes to calculate the intermediate gradients for the computation of the eligibility traces. Despite this, we have found from our experiments that there is no significant difference in the performance of T-BPTT as compared to using a larger hidden state than we are using.

For the T-BPTT experiments, we used a truncation length of k=27 and a hidden size of 18. This truncation length essentially makes the T-BPTT equivalent to BPTT since $k >$ max(ISI). The columnar network has 108 independent recurrent units in the hidden layer. The constructive networks begin with a single recurrent feature in the hidden layer and add a new feature every 400,000 steps. By the end of the training, this network ends up with a total of 23 recurrent features. The hybrid approach begins with 20 recurrent features in the hidden layer, and adds 20 features every 800,000 steps, until it reaches a maximum of 108 features. TODO: How should we describe Diagonal Jacobian here?

The results of the experiment are shown in Fig. 6. We can see that the Diagonal Jacobian and the T-BPTT methods perform the worst on this benchmark. The columnar and the constructive approaches achieve a significantly lower error than these baselines. Furthermore, these two methods achieve an asymptotic performance that is similar to each other. Finally, the hybrid approach, which is a combination of the constructive and the columnar approaches, achieves the lowest asymptotic error out of all the methods presented.

From these experiments, we have shown that despite our methods limiting the class of the function approximators that we can use, the unbiased and low-variance gradient updates allow these methods to achieve a potentially better performance in the challenging online prediction tasks, while still maintaining linear scalability with time.

TODO: The T-BPTT curve here is not using eligibility traces. Should the size still be adjusted?

TODO: Where are the plots for columnar with 23 total features stored? I should plot those instead maybe.

7. Temporally symmetric hierarchical recurrent learning using generate and test

The primary limitation of the constructive and hybrid approaches is that they are not suitable for continual learning. Once a feature is frozen, it never changes, even if the environment does. In the constructive case, the agent can, at best, continually update one recurrent feature. In the hybrid case, the agent can update k recurrent units added last. Inability to update all features after a certain number of time steps make the agent lose the ability to modify its feature representation. In a sense, the agent becomes a linear learner.

We can address this limitation by identifying features that are no longer useful, and pruning them. These pruned features can then be replaced with newly constructed features. Similar ideas have been proposed in literature under the term generate and test.

To effectively combine our method with generate and test, we need to answer two questions—(1) how useless features are identified, and (2) how new features replace them. It is unclear what an ideal algorithm to answer these two questions are. In this work, we limit ourselves to the simple algorithms for both questions. The goal here is to present one temporally symmetric version of the hybrid approach, and not to figure out the best temporally symmetric version.

To identify useless features, we simply look at how much a feature contributes to its outgoing connection, and use the sum of contribution to all outgoing connections as the utility of the feature. The contribution of a feature to an outgoing connection is defined as the

8. Conclusions

Use hybrid networks.

Acknowledgments

We would like to acknowledge support for this project from the National Science Foundation (NSF grant IIS-9988642) and the Multidisciplinary Research Program of the Department of Defense (MURI N00014-00-1-0637).

Appendix A.

A. Hyper-parameter Settings

We performed an exhaustive search over all the hyper-parameter ranges for each of the methods that is presented in this paper. For each hyper-parameter configuration, we averaged its performance over at-least three runs and determined the best setting based on the lowest average final error. The sweep ranges for the different hyper-parameters are provided in Table 1 and the best hyper-parameters are provided in Table 2.

	Hyper-parameter	Sweep Ranges
α	Step-size	$\{3^{-1}, 1^{-1}, 3^{-2}, 1^{-2}, 3^{-3}, 3^{-4}\}$
λ	Eligibility Traces Decay Rate	$\{0, 0.5, 0.9, 0.99\}$
k	Truncation Length (T-BPTT)	$\{1, 10, 27, 50\}$
n	Number of Initial Features (Hybrid)	$\{5, 10, 20\}$
η	Feature Addition Interval (Constructive/Hybrid)	$x * 10,000;\ x \in \{4, 10, 20, 40, 80\}$

Table 1: Sweep ranges for each hyper-parameter. TODO: The notation for last two rows is just placeholder. Replace it with something appropriate.

Hyper-parameter	T-BPTT	Diagonal Jacobian	Columnar	Constructive	Hybrid
α	1^{-1}	1^{-2}	1^{-1}	1^{-2}	1^{-2}
λ	0	0.99	0.9	0.99	0.99
k	27	-	-	-	-
n	18	108	108	1	20
η	-	-	-	400,000	800,000

Table 2: The best hyper-parameter configuration found for each method according to the average final error over multiple runs. Note that although we specify the n parameter for T-BPTT, Diagonal Jacobian and Columnar approaches, no new feature will be added during their training. It is simply the width of the network.

B. Implementation Details

We implemented the Diagonal Jacobian, Columnar, Constructive and the Hybrid Networks in C++. T-BPTT is implemented using Pytorch. The code for replicating these results (along with an optional python wrapper for the C++ code with python environment) is released at https://github.com/khurramjaved96/constructive_rnn

Appendix B.

C. Forward-mode gradient computation of an LSTM cell

Here we derive the update equations for recursively computing the gradients of a single LSTM based recurrent column. Each column has a single hidden unit. Because all columns are identical, the same update equations can be used to learn the complete Columnar Network. The state of an LSTM column is updated using following equations:

$$i(t) = \sigma(W_i^T x_k(t) + u_i h(t-1) + b_i) \tag{8}$$
$$f(t) = \sigma(W_f^T x_k(t) + u_f h(t-1) + b_f) \tag{9}$$
$$o(t) = \sigma(W_o^T x_k(t) + u_o h(t-1) + b_o) \tag{10}$$
$$g(t) = \phi(W_g^T x_k(t) + u_g h(t-1) + b_g) \tag{11}$$
$$c(t) = f(t)c(t-1) + i(t)g(t) \tag{12}$$
$$h(t) = o(t)\phi(c(t)) \tag{13}$$

where σ and ϕ are the sigmoid and tanh activation functions, $h(t)$ is the state of the column at time t and $W_i^T x_k(t) = \sum_{k=1}^m W_{i_k} x_k(t)$. The derivative of $\sigma(x)$ and $\phi(x)$ w.r.t to x are $\sigma(x)(1-\sigma(x))$ and $(1-\phi^2(x))$ respectively.

Let the length of input vector x be m. Then, W_i, W_f, W_o and W_g are vectors of length m whereas $u_i, b_i, u_f, b_f, u_o, b_o, u_g$ and b_g are scalars. We want to compute gradient of $h(t)$ with respect to all the parameters. We derive the update equations for $\frac{\partial h(t)}{\partial W_i}, \frac{\partial h(t)}{\partial u_i}, \frac{\partial h(t)}{\partial b_i}, \frac{\partial h(t)}{\partial W_f}, \frac{\partial h(t)}{\partial u_f}, \frac{\partial h(t)}{\partial b_f}, \frac{\partial h(t)}{\partial W_o}, \frac{\partial h(t)}{\partial u_o}, \frac{\partial h(t)}{\partial b_o}, \frac{\partial h(t)}{\partial W_g}, \frac{\partial h(t)}{\partial u_g}$, and $\frac{\partial h(t)}{\partial b_g}$ in the following sections.

C.1 $\frac{\partial h(t)}{\partial W_i}$

$W_i = (W_{i_1}, W_{i_2}, \cdots, W_{i_m})$ is a vector of length m. Since all elements of W_i are symmetric, we show gradient derivation for W_{i_j} without loss of generality. Let

$$TH_{W_{i_j}}(t) := \frac{\partial h(t)}{\partial W_{i_j}} \qquad \textit{(By definition)} \tag{14}$$
$$TH_{W_{i_j}}(0) := 0 \qquad \textit{(By definition)} \tag{15}$$
$$TC_{W_{i_j}}(t) := \frac{\partial c(t)}{\partial W_{i_j}} \qquad \textit{(By definition)} \tag{16}$$
$$TC_{W_{i_j}}(0) := 0 \qquad \textit{(By definition)} \tag{17}$$

Then:

$$\begin{aligned}
TH_{W_{i_j}}(t) &= \frac{\partial}{\partial W_{i_j}}\left(o(t)\phi(c(t))\right) && \textit{From equation 13 and definition 14} \\
&= o(t)\frac{\partial \phi(c(t))}{\partial W_{i_j}} + \phi(c(t))\frac{\partial o(t)}{\partial W_{i_j}} && \textit{Product rule of differentiation} \\
&= o(t)(1-\phi^2(c(t)))\frac{\partial c(t)}{\partial W_{i_j}} + \phi(c(t))\frac{\partial o(t)}{\partial W_{i_j}} && \textit{Derivative of } \phi(x) \textit{ is (1-}\phi^2(x)) \\
&= o(t)(1-\phi^2(c(t)))TC_{W_{i_j}}(t) + \phi(c(t))\frac{\partial o(t)}{\partial W_{i_j}} && \textit{From definition 16} \\
\frac{\partial o(t)}{\partial W_{i_j}} &= \frac{\partial}{\partial W_{i_j}}\sigma(W_o^T x(t) + u_o h(t-1) + b_o) && \textit{From equation 10} \\
&= \sigma(y)(1-\sigma(y))u_o TH_{W_{i_j}}(t-1) && \textit{Where } y \textit{ equals } W_o^T x(t) + u_o h(t-1) + \\
TC_{W_{i_j}}(t) &= \frac{\partial c(t)}{\partial W_{i_j}} && \textit{From definition 16} \\
&= \frac{\partial}{\partial W_{i_j}}(f(t)c(t-1) + i(t)g(t)) && \textit{From equation 12} \\
&= f(t)TC_{W_{i_j}}(t-1) + c(t-1)\frac{\partial f(t)}{\partial W_{i_j}} + \frac{\partial}{\partial W_{i_j}}(i(t)g(t)) && \textit{Product rule and definition 16} \\
&= f(t)TC_{W_{i_j}}(t-1) + c(t-1)\frac{\partial f(t)}{\partial W_{i_j}} + i(t)\frac{\partial g(t)}{\partial W_{i_j}} + g(t)\frac{\partial i(t)}{\partial W_{i_j}} && \textit{Product rule}
\end{aligned}$$

Where gradient of $g(t)$ w.r.t W_{i_j} is:

$$\begin{aligned}
\frac{\partial g(t)}{\partial W_{i_j}} &= \frac{\partial}{\partial W_{i_j}}\phi(W_g^T x(t) + u_g h(t-1) + b_g) && \textit{From equation 11} \\
&= (1-\phi^2(y))u_g TH_{W_{i_j}}(t-1) && \textit{Where } y \textit{ equals } W_g^T x(t) + u_g h(t-1) + b_g
\end{aligned}$$

, gradient of $f(t)$ w.r.t W_{i_j} is:

$$\begin{aligned}
\frac{\partial f(t)}{\partial W_{i_j}} &= \frac{\partial}{\partial W_{i_j}}\sigma(W_f^T x(t) + u_f h(t-1) + b_f) && \textit{From equation 9} \\
&= \sigma(y)(1-\sigma(y))u_f TH_{W_{i_j}}(t-1) && \textit{Where } y \textit{ equals } W_f^T x(t) + u_f h(t-1) + b_f
\end{aligned}$$

and gradient of $i(t)$ w.r.t W_{i_j} is:

$$\begin{aligned}
\frac{\partial i(t)}{\partial W_{i_j}} &= \frac{\partial}{\partial W_{i_j}}\sigma(W_i^T x(t) + u_i h(t-1) + i_f) && \textit{From equation 8} \\
&= \sigma(y)(1-\sigma(y))\left(x_j(t) + u_i TH_{W_{i_j}}(t-1)\right) && \textit{Where } y \textit{ equals } W_i^T x(t) + u_i h(t-1) + b_i
\end{aligned}$$

The derivation shows that using two traces per parameter of W_i, it is possible to compute the gradient of $h(t)$ w.r.t W_i recursively. We provide the derivations for parameters u_i and b_i below. We skip the step-by-step derivations for the remaining parameters as they are similar.

C.2 $\frac{\partial h(t)}{\partial u_i}$

$$
\begin{aligned}
TH_{u_i}(t) &:= \frac{\partial h(t)}{\partial u_i} && \textit{(By definition)} && (18)\\
TH_{u_i}(0) &:= 0 && \textit{(By definition)} && (19)\\
TC_{u_i}(t) &:= \frac{\partial c(t)}{\partial u_i} && \textit{(By definition)} && (20)\\
TC_{u_i}(0) &:= 0 && \textit{(By definition)} && (21)
\end{aligned}
$$

$$
\begin{aligned}
TH_{u_i}(t) &= \frac{\partial}{\partial u_i}\left(o(t)\phi(c(t))\right) && \textit{From equation 13}\\
&= o(t)\frac{\partial \phi(c(t))}{\partial u_i} + \phi(c(t))\frac{\partial o(t)}{\partial u_i} && \textit{Product rule}\\
&= o(t)(1-\phi^2(c(t)))\frac{\partial c(t)}{\partial u_i} + \phi(c(t))\frac{\partial o(t)}{\partial u_i} && \textit{Derivative of } \phi(x) \textit{ is } 1-\phi^2(x)\\
&= o(t)(1-\phi^2(c(t)))TC_{u_i}(t) + \phi(c(t))\frac{\partial o(t)}{\partial u_i} && \textit{Using definition 20}\\
\frac{\partial o(t)}{\partial u_i} &= \frac{\partial}{\partial u_i}\sigma(W_o^T x(t) + u_o h(t-1) + b_o) && \textit{Using equations 10}\\
&= \sigma(x)(1-\sigma(x))u_o TH_{u_i}(t-1) && \textit{Where } x \textit{ equal } W_o^T x(t) + u_o h(t-1) + b_o\\
TC_{u_i}(t) &= \frac{\partial c(t)}{\partial u_i} && \textit{Definition 20}\\
&= \frac{\partial}{\partial u_i}(f(t)c(t-1) + i(t)g(t)) && \textit{From equation 13}\\
&= f(t)TC_{u_i}(t-1) + c(t-1)\frac{\partial f(t)}{\partial u_i} + \frac{\partial}{\partial u_i}\left(i(t)g(t)\right) && \textit{Product rule}\\
&= f(t)TC_{u_i}(t-1) + c(t-1)\frac{\partial f(t)}{\partial u_i} + i(t)\frac{\partial g(t)}{\partial u_i} + g(t)\frac{\partial i(t)}{\partial u_i} && \textit{Product rule}
\end{aligned}
$$

Gradient of $g(t)$ w.r.t u_i is:

$$
\begin{aligned}
\frac{\partial g(t)}{\partial u_i} &= \frac{\partial}{\partial u_i}\phi(W_g^T x(t) + u_g h(t-1) + b_g) && \textit{From equations 13}\\
&= (1-\phi^2(y))u_g TH_{u_i}(t-1) && \textit{Where } y \textit{ equals } W_g^T x(t) + u_g h(t-1) + b_g
\end{aligned}
$$

, gradient of $f(t)$ w.r.t u_i is:

$$\begin{aligned}
\frac{\partial f(t)}{\partial u_i} &= \frac{\partial}{\partial u_i}\sigma(W_f^T x(t) + u_f h(t-1) + b_f) && \textit{From equations 1} \\
&= \sigma(y)(1-\sigma(y))u_f TH_{u_i}(t-1) && \textit{Where } y \textit{ equals } W_f^T x(t) + u_f h(t-1) + b_f
\end{aligned}$$

and the gradient of $i(t)$ w.r.t u_i is

$$\begin{aligned}
\frac{\partial i(t)}{\partial u_i} &= \frac{\partial}{\partial u_i}\sigma(W_i^T x(t) + u_i h(t-1) + b_i) && \textit{Using equations 1} \\
&= \sigma(y)(1-\sigma(y))\left(h(t-1) + u_i TH_{u_i}(t-1)\right) && \textit{Where } y \textit{ equals } W_i^T x(t) + u_i h(t-1) + b_i
\end{aligned}$$

C.3 $\frac{\partial h(t)}{\partial b_i}$

$$\begin{aligned}
TH_{b_i}(t) &:= \frac{\partial h(t)}{\partial b_i} && \textit{(By definition)} && (22) \\
TH_{b_i}(0) &:= 0 && \textit{(By definition)} && (23) \\
TC_{b_i}(t) &:= \frac{\partial c(t)}{\partial b_i} && \textit{(By definition)} && (24) \\
TC_{b_i}(0) &:= 0 && \textit{(By definition)} && (25)
\end{aligned}$$

$$\begin{aligned}
TH_{b_i}(t) &= \frac{\partial}{\partial b_i}\left(o(t)\phi(c(t))\right) && \textit{From equation 13} \\
&= o(t)\frac{\partial \phi(c(t))}{\partial b_i} + \phi(c(t))\frac{\partial o(t)}{\partial b_i} && \textit{Product rule} \\
&= o(t)(1-\phi^2(c(t)))\frac{\partial c(t)}{\partial b_i} + \phi(c(t))\frac{\partial o(t)}{\partial b_i} && \textit{Derivative of of } \phi(x) \textit{ is } 1-\phi^2(x) \\
&= o(t)(1-\phi^2(c(t)))TC_{b_i}(t) + \phi(c(t))\frac{\partial o(t)}{\partial b_i} && \textit{From definition 24} \\
\frac{\partial o(t)}{\partial b_i} &= \frac{\partial}{\partial b_i}\sigma(W_o^T x(t) + u_o h(t-1) + b_o) && \textit{From equations 10} \\
&= \sigma(y)(1-\sigma(y))u_o TH_{b_i}(t-1) && \textit{Where } y \textit{ equal } W_o^T x(t) + u_o h(t-1) + b_o \\
TC_{b_i}(t) &= \frac{\partial c(t)}{\partial b_i} && \textit{From definition 24} \\
&= \frac{\partial}{\partial b_i}(f(t)c(t-1) + i(t)g(t)) && \textit{From equation 12} \\
&= f(t)TC_{b_i}(t-1) + c(t-1)\frac{\partial f(t)}{\partial b_i} + \frac{\partial}{\partial b_i}i(t)g(t) && \textit{Product rule} \\
&= f(t)TC_{b_i}(t-1) + c(t-1)\frac{\partial f(t)}{\partial b_i} + i(t)\frac{\partial g(t)}{\partial b_i} + g(t)\frac{\partial i(t)}{\partial b_i} && \textit{Product rule}
\end{aligned}$$

Where gadient of $g(t)$ w.r.t b_i is:

$$\begin{aligned}\frac{\partial g(t)}{\partial b_i} &= \frac{\partial}{\partial b_i}\phi(W_g^T x(t) + u_g h(t-1) + b_g) && \textit{From equation 11}\\ &= (1-\phi^2(y))u_g TH_{b_i}(t-1) && \textit{Where } y \textit{ equal } W_g^T x(t) + u_g h(t-1) + b_g\end{aligned}$$

, gradient of $f(t)$ w.r.t b_i is:

$$\begin{aligned}\frac{\partial f(t)}{\partial b_i} &= \frac{\partial}{\partial b_i}\sigma(W_f^T x(t) + u_f h(t-1) + b_f) && \textit{From equation 9}\\ &= \sigma(y)(1-\sigma(y))u_f TH_{b_i}(t-1) && \textit{Where } y \textit{ equal } W_f^T x(t) + u_f h(t-1) + b_f\end{aligned}$$

and gradient of $i(t)$ w.r.t b_i is:

$$\begin{aligned}\frac{\partial i(t)}{\partial b_i} &= \frac{\partial}{\partial b_i}\sigma(W_i^T x(t) + u_i h(t-1) + b_i) && \textit{From equation 8}\\ &= \sigma(y)(1-\sigma(y))\left(u_i TH_{b_i}(t-1) + 1\right) && \textit{Where } y \textit{ equal } W_i^T x(t) + b_i h(t-1) + b_i\end{aligned}$$

C.4 $\frac{\partial h(t)}{\partial W_{f_j}}$

The derivations for the remaining parameters is analogous to what previous derivations. The final equations are as follows.

$$\begin{aligned}\frac{\partial g(t)}{\partial W_{f_j}} &= (1-\phi^2(y))(u_g TH_{W_{f_j}}(t-1))\\ \frac{\partial f(t)}{\partial W_{f_j}} &= \sigma(y)(1-\sigma(y))(x_j + u_f TH_{W_{f_j}}(t-1))\\ \frac{\partial i(t)}{\partial W_{f_j}} &= \sigma(y)(1-\sigma(y))(u_i TH_{W_{f_j}}(t-1))\\ \frac{\partial o(t)}{\partial W_{f_j}} &= \sigma(y)(1-\sigma(y))(u_o TH_{W_{f_j}}(t-1))\\ TC_{W_{f_j}} &= f(t)TC_{f_j}(t-1) + c(t-1)\frac{\partial f(t)}{\partial b_i} + i(t)\frac{\partial g(t)}{\partial b_i} + g(t)\frac{\partial i(t)}{\partial b_i}\\ TH_{W_{f_j}} &= o(t)(1-\phi^2(c(t)))TC_{W_{f_j}}(t) + \phi(c(t))\frac{\partial o(t)}{\partial W_{ij}}\end{aligned} \tag{26}$$

C.5 $\frac{\partial h(t)}{\partial W_{o_j}}$

$$\begin{aligned}
\frac{\partial g(t)}{\partial W_{o_j}} &= (1-\phi^2(y))(u_g TH_{W_{o_j}}(t-1)) \\
\frac{\partial f(t)}{\partial W_{o_j}} &= \sigma(y)(1-\sigma(y))(u_f TH_{W_{o_j}}(t-1)) \\
\frac{\partial i(t)}{\partial W_{o_j}} &= \sigma(y)(1-\sigma(y))u_i TH_{W_{o_j}}(t-1) \\
\frac{\partial o(t)}{\partial W_{o_j}} &= \sigma(x)(1-\sigma(x))(x_j + u_o TH_{W_{o_j}}(t-1)) \\
TC_{W_{o_j}} &= f(t)TC_{o_j}(t-1) + c(t-1)\frac{\partial f(t)}{\partial b_i} + i(t)\frac{\partial g(t)}{\partial b_i} + g(t)\frac{\partial i(t)}{\partial b_i} \\
TH_{W_{o_j}} &= o(t)(1-\phi^2(c(t)))TC_{W_{o_j}}(t) + \phi(c(t))\frac{\partial o(t)}{\partial W_{ij}}
\end{aligned} \tag{27}$$

C.6 $\frac{\partial h(t)}{\partial W_{g_j}}$

$$\begin{aligned}
\frac{\partial g(t)}{\partial W_{g_j}} &= (1-\phi^2(y))(x_j + u_g TH_{W_{g_j}}(t-1)) \\
\frac{\partial f(t)}{\partial W_{g_j}} &= \sigma(y)(1-\sigma(y))(u_f TH_{W_{g_j}}(t-1)) \\
\frac{\partial i(t)}{\partial W_{g_j}} &= \sigma(y)(1-\sigma(y))(u_i TH_{W_{g_j}}(t-1)) \\
\frac{\partial o(t)}{\partial W_{g_j}} &= \sigma(x)(1-\sigma(x))(u_o TH_{W_{g_j}}(t-1)) \\
TC_{W_{g_j}} &= f(t)TC_{g_j}(t-1) + c(t-1)\frac{\partial f(t)}{\partial b_i} + i(t)\frac{\partial g(t)}{\partial b_i} + g(t)\frac{\partial i(t)}{\partial b_i} \\
TH_{W_{g_j}} &= o(t)(1-\phi^2(c(t)))TC_{W_{g_j}}(t) + \phi(c(t))\frac{\partial o(t)}{\partial W_{ij}}
\end{aligned} \tag{28}$$

C.7 $\frac{\partial h(t)}{\partial u_o}$

$$\begin{aligned}
\frac{\partial g(t)}{\partial u_o} &= (1-\phi^2(y))(u_g TH_{u_o}(t-1)) \\
\frac{\partial f(t)}{\partial u_o} &= \sigma(y)(1-\sigma(y))(u_f TH_{u_o}(t-1)) \\
\frac{\partial i(t)}{\partial u_o} &= \sigma(y)(1-\sigma(y))(u_i TH_{u_o}(t-1)) \\
\frac{\partial o(t)}{\partial u_o} &= \sigma(x)(1-\sigma(x))(u_o TH_{u_o}(t-1) + h(t-1)) \\
TC_{u_o} &= f(t)TC_{i_j}(t-1) + c(t-1)\frac{\partial f(t)}{\partial b_i} + i(t)\frac{\partial g(t)}{\partial b_i} + g(t)\frac{\partial i(t)}{\partial b_i} \\
TH_{u_o} &= o(t)(1-\phi^2(c(t)))TC_{u_o}(t) + \phi(c(t))\frac{\partial o(t)}{\partial W_{ij}}
\end{aligned} \tag{29}$$

C.8 $\frac{\partial h(t)}{\partial u_f}$

$$\begin{aligned}
\frac{\partial g(t)}{\partial u_f} &= (1-\phi^2(y))(u_g TH_{u_f}(t-1)) \\
\frac{\partial f(t)}{\partial u_f} &= \sigma(y)(1-\sigma(y))(u_f TH_{u_f}(t-1) + h(t-1)) \\
\frac{\partial i(t)}{\partial u_f} &= \sigma(y)(1-\sigma(y))(u_i TH_{u_f}(t-1)) \\
\frac{\partial o(t)}{\partial u_f} &= \sigma(x)(1-\sigma(x))(u_o TH_{u_f}(t-1)) \\
TC_{u_f} &= f(t)TC_{i_j}(t-1) + c(t-1)\frac{\partial f(t)}{\partial b_i} + i(t)\frac{\partial g(t)}{\partial b_i} + g(t)\frac{\partial i(t)}{\partial b_i} \\
TH_{u_f} &= o(t)(1-\phi^2(c(t)))TC_{u_f}(t) + \phi(c(t))\frac{\partial o(t)}{\partial W_{ij}}
\end{aligned} \tag{30}$$

C.9 $\frac{\partial h(t)}{\partial u_g}$

$$\begin{aligned}
\frac{\partial g(t)}{\partial u_g} &= (1-\phi^2(y))(u_g TH_{u_g}(t-1) + h(t-1)) \\
\frac{\partial f(t)}{\partial u_g} &= \sigma(y)(1-\sigma(y))(u_f TH_{u_g}(t-1)) \\
\frac{\partial i(t)}{\partial u_g} &= \sigma(y)(1-\sigma(y))(u_i TH_{u_g}(t-1)) \\
\frac{\partial o(t)}{\partial u_g} &= \sigma(x)(1-\sigma(x))(u_o TH_{u_g}(t-1)) \\
TC_{u_g} &= f(t)TC_{i_j}(t-1) + c(t-1)\frac{\partial f(t)}{\partial b_i} + i(t)\frac{\partial g(t)}{\partial b_i} + g(t)\frac{\partial i(t)}{\partial b_i} \\
TH_{u_g} &= o(t)(1-\phi^2(c(t)))TC_{u_g}(t) + \phi(c(t))\frac{\partial o(t)}{\partial W_{ij}}
\end{aligned} \tag{31}$$

C.10 $\frac{\partial h(t)}{\partial b_g}$

$$\begin{aligned}
\frac{\partial g(t)}{\partial b_g} &= (1-\phi^2(y))(u_g TH_{b_g}(t-1) + 1) \\
\frac{\partial f(t)}{\partial b_g} &= \sigma(y)(1-\sigma(y))(u_f TH_{b_g}(t-1)) \\
\frac{\partial i(t)}{\partial b_g} &= \sigma(y)(1-\sigma(y))(u_i TH_{b_g}(t-1)) \\
\frac{\partial o(t)}{\partial b_g} &= \sigma(x)(1-\sigma(x))(u_o TH_{b_g}(t-1)) \\
TC_{b_g} &= f(t)TC_{i_j}(t-1) + c(t-1)\frac{\partial f(t)}{\partial b_i} + i(t)\frac{\partial g(t)}{\partial b_i} + g(t)\frac{\partial i(t)}{\partial b_i} \\
TH_{b_g} &= o(t)(1-\phi^2(c(t)))TC_{b_g}(t) + \phi(c(t))\frac{\partial o(t)}{\partial W_{ij}}
\end{aligned} \tag{32}$$

C.11 $\frac{\partial h(t)}{\partial b_f}$

$$\begin{aligned}
\frac{\partial g(t)}{\partial b_f} &= (1-\phi^2(y))(u_g TH_{b_f}(t-1)) \\
\frac{\partial f(t)}{\partial b_f} &= \sigma(y)(1-\sigma(y))(u_f TH_{b_f}(t-1)+1) \\
\frac{\partial i(t)}{\partial b_f} &= \sigma(y)(1-\sigma(y))(u_i TH_{b_f}(t-1)) \\
\frac{\partial o(t)}{\partial b_f} &= \sigma(x)(1-\sigma(x))(u_o TH_{b_f}(t-1)) \\
TC_{b_f} &= f(t)TC_{i_j}(t-1) + c(t-1)\frac{\partial f(t)}{\partial b_i} + i(t)\frac{\partial g(t)}{\partial b_i} + g(t)\frac{\partial i(t)}{\partial b_i} \\
TH_{b_f} &= o(t)(1-\phi^2(c(t)))TC_{b_f}(t) + \phi(c(t))\frac{\partial o(t)}{\partial W_{ij}}
\end{aligned} \tag{33}$$

C.12 $\frac{\partial h(t)}{\partial b_o}$

$$\begin{aligned}
\frac{\partial g(t)}{\partial b_o} &= (1-\phi^2(y))(u_g TH_{b_o}(t-1)) \\
\frac{\partial f(t)}{\partial b_o} &= \sigma(y)(1-\sigma(y))(u_f TH_{b_o}(t-1)) \\
\frac{\partial i(t)}{\partial b_o} &= \sigma(y)(1-\sigma(y))(u_i TH_{b_o}(t-1)) \\
\frac{\partial o(t)}{\partial b_o} &= \sigma(x)(1-\sigma(x))(u_o TH_{b_o}(t-1)+1) \\
TC_{b_o} &= f(t)TC_{i_j}(t-1) + c(t-1)\frac{\partial f(t)}{\partial b_i} + i(t)\frac{\partial g(t)}{\partial b_i} + g(t)\frac{\partial i(t)}{\partial b_i} \\
TH_{b_o} &= o(t)(1-\phi^2(c(t)))TC_{b_o}(t) + \phi(c(t))\frac{\partial o(t)}{\partial W_{ij}}
\end{aligned} \tag{34}$$

References

Banafsheh Rafiee. Testbeds for reinforcement learning. *ArXiv*, abs/2011.04590, 2020.