Improving the Gating Mechanism of Recurrent Neural Networks

Albert Gu [1] **Caglar Gulcehre** [2] **Tom Paine** [2] **Matt Hoffman** [2] **Razvan Pascanu** [2]

Abstract

Gating mechanisms are widely used in neural network models, where they allow gradients to backpropagate more easily through depth or time. However, their saturation property introduces problems of its own. For example, in recurrent models these gates need to have outputs near 1 to propagate information over long time-delays, which requires them to operate in their saturation regime and hinders gradient-based learning of the gate mechanism. We address this problem by deriving two synergistic modifications to the standard gating mechanism that are easy to implement, introduce no additional hyperparameters, and improve learnability of the gates when they are close to saturation. We show how these changes are related to and improve on alternative recently proposed gating mechanisms such as chrono initialization and Ordered Neurons. Empirically, our simple gating mechanisms robustly improve the performance of recurrent models on a range of applications, including synthetic memorization tasks, sequential image classification, language modeling, and reinforcement learning, particularly when long-term dependencies are involved.

1. Introduction

Recurrent neural networks (RNNs) are an established machine learning tool for learning from sequential data. However, RNNs are prone to the vanishing gradient problem, which occurs when the gradients of the recurrent weights become vanishingly small as they get backpropagated through time (Hochreiter, 1991; Bengio et al., 1994; Hochreiter et al., 2001). A common approach to alleviate the vanishing gradient problem is to use gating mechanisms, leading to models such as the long short term memory (Hochreiter & Schmidhuber, 1997, LSTM) and gated recurrent units (Chung et al., 2014, GRUs). These gated RNNs have been very successful in several different application areas such as in reinforcement learning (Kapturowski et al., 2018; Espeholt et al., 2018) and natural language processing (Bahdanau et al., 2014; Kočiskỳ et al., 2018).

At every time step, gated recurrent networks use a weighted combination of the history summarized by the previous state, and a function of the incoming inputs, to create the next state. The values of the gates, which are the coefficients of the weighted combination, control the length of temporal dependencies that can be addressed. This weighted update can be seen as an additive or residual connection on the recurrent state, which helps signals propagate through time without vanishing. However, the gates themselves are prone to a saturating property which can also hamper gradient-based learning. This can be problematic for RNNs, where carrying information for very long time delays requires gates to be very close to their saturated states.

We address two particular problems that arise with the standard gating mechanism of recurrent models. Firstly, learning when gates are in their saturation regime is difficult because gradients through the gates vanish as they saturate. We derive a modification to standard gating mechanisms that uses an auxiliary *refine gate* (Section 3.1) to modulate a main gate. This mechanism allows the gates to have a wider range of activations without gradients vanishing as quickly. Secondly, typical initialization of the gates is relatively concentrated. This restricts the range of timescales the model can address at initialization, as the timescale of a particular unit is dictated by its gates. We propose *uniform gate initialization* (Section 3.2) that addresses this problem by directly initializing the activations of these gates from a distribution that captures a wider spread of dependency lengths.

The main contribution of this paper is the **refine** gate mechanism. As the refine gate works better in tandem with uniform gate initialization, we call this combination the **UR** gating mechanism. We focus on comparing the UR gating mechanism against other approaches in our experiments. These changes can be applied to any gate (i.e. parameterized bounded function) and have minimal to no overhead in terms of speed, memory, code complexity, parameters, or hyper parameters. We apply them to the forget gate of recurrent models, and evaluate on several benchmark tasks that re-

[1]Stanford University, USA [2]DeepMind, London, UK. Correspondence to: Albert Gu <albertgu@stanford.edu>, Caglar Gulcehre <caglarg@google.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

quire long-term memory including synthetic memory tasks, pixel-by-pixel image classification, language modeling, and reinforcement learning. Finally, we connect our methods to other proposed gating modifications, introduce a framework that allows each component to be replaced with similar ones, and perform extensive ablations of our method. Empirically, the UR gating mechanism robustly improves on the standard forget and input gates of gated recurrent models. When applied to the LSTM, these simple modifications solve synthetic memory tasks that are pathologically difficult for the standard LSTM, achieve state-of-the-art results on sequential MNIST and CIFAR-10, and show consistent improvements in language modeling on the WikiText-103 dataset (Merity et al., 2016) and reinforcement learning tasks (Hung et al., 2018).

2. Gated Recurrent Neural Networks

Broadly speaking, RNNs are used to sweep over a sequence of input data x_t to produce a sequence of recurrent states $h_t \in \mathbb{R}^d$ summarizing information seen so far. At a high level, an RNN is just a parametrized function in which each sequential application of the network computes a state update $u : (x_t, h_{t-1}) \mapsto h_t$. Gating mechanisms were introduced to address the vanishing gradient problem (Hochreiter, 1991; Bengio et al., 1994; Hochreiter et al., 2001), and have proven to be crucial to the success of RNNs. This mechanism essentially smooths out the update using the following equation,

$$h_t = f_t(x_t, h_{t-1}) \circ h_{t-1} + i_t(x_t, h_{t-1}) \circ u(x_t, h_{t-1}), \quad (1)$$

where the *forget gate* f_t and *input gate* i_t are $[0,1]^d$-valued functions that control how fast information is forgotten or allowed into the memory state. When the gates are tied, i.e. $f_t + i_t = 1$ as in GRUs, they behave as a low-pass filter, deciding the time-scale on which the unit will respond (Tallec & Ollivier, 2018). For example, large forget gate activations close to $f_t = 1$ are necessary for recurrent models to address long-term dependencies.[1]

We will introduce our improvements to the gating mechanism primarily in the context of the LSTM, which is the most popular recurrent model.

$$f_t = \sigma(\mathcal{P}_f(x_t, h_{t-1})), \quad (2)$$
$$i_t = \sigma(\mathcal{P}_i(x_t, h_{t-1})), \quad (3)$$
$$u_t = \tanh(\mathcal{P}_u(x_t, h_{t-1})), \quad (4)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ u_t, \quad (5)$$
$$o_t = \sigma(\mathcal{P}_o(x_t, h_{t-1})), \quad (6)$$
$$h_t = o_t \tanh(c_t). \quad (7)$$

A typical LSTM (equations (2)-(7)) is an RNN whose state is represented by a tuple (h_t, c_t) consisting of a "hidden" state and "cell" state. The state update equation (1) is used to create the next cell state c_t (5). Note that the gate and update activations are a function of the previous hidden state h_{t-1} instead of c_{t-1}. Here, $\mathcal{P}_\star$ stands for a parameterized linear function of its inputs with bias $b_\star$, e.g.

$$\mathcal{P}_f(x_t, h_{t-1}) = W_{fx} x_t + W_{fh} h_{t-1} + b_f. \quad (8)$$

and $\sigma(\cdot)$ refers to the standard sigmoid activation function which we will assume is used for defining $[0,1]$-valued activations in the rest of this paper. The gates of the LSTM were initially motivated as a binary mechanism, switching on or off, allowing information and gradients to pass through. However, in reality, this fails to happen due to a combination of two factors: initialization and saturation. This can be problematic, such as when very long dependencies are present.

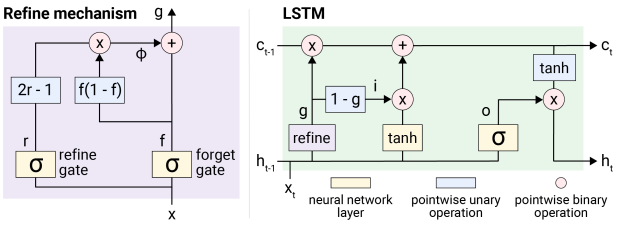


Figure 1. **Refine mechanism.** The refine mechanism improves flow of gradients through a saturating gate f. As f saturates, its gradient vanishes, and its value is unlikely to change (see Figure 4). The refine gate r is used to produce a bounded additive term ϕ that may push f lower or higher as necessary. The resulting effective gate g can achieve values closer to 0 and 1 and can change even when f is stuck. We apply it to the forget gate of an LSTM. The g is then used in place of f in the state update (5).

3. Our Proposed Gating Mechanisms

We present two solutions that work in tandem to address the previously described issues. The first is the **refine gate**, which allows for better gradient flow by reparameterizing a saturating gate, for example, the forget gate. The second is **uniform gate initialization**, which ensures a diverse range of gate values are captured at the start of training, which allows a recurrent model to have a multi-scale representation of an input sequence at initialization.

3.1. Refine Gate

Formally, the full mechanism of the refine gate as applied to gated recurrent models is defined in equations (9)-(11). Note that it is an isolated change where the forget gate f_t is modified to get the effective forget gate in (10) before applying the the standard update (1). Figure 1 illustrates the refine gate in an LSTM cell. Figure 3 illustrates how the refine gate r_t is defined and how it changes the forget gate f_t to produce an effective gate g_t. The refine gate allows the effective gate g to reach much higher and lower activations than the

[1] In this work, we use "gate" to alternatively refer to a $[0,1]$-valued function or the value ("activation") of that function.

constituent gates f and r, bypassing the saturating gradient problem. For example, this allows the effective forget gate to reach $g = 0.99$ when the forget gate is only $f = 0.9$.

Finally, to simplify comparisons and ensure that we always use the same number of parameters as the standard gates, when using the refine gate we tie the input gate to the effective forget gate, $i_t = 1 - g_t$.[2] However, we emphasize that these techniques can be applied to any gate (or more broadly, any bounded function) to improve initialization distribution and help optimization. For example, our methods can be combined in different ways in recurrent models, e.g. an independent input gate can be modified with its own refine gate.

$$r_t = \sigma(\mathcal{P}_r(x_t, h_{t-1})), \tag{9}$$

$$g_t = r_t \cdot (1-(1-f_t)^2) + (1-r_t) \cdot f_t^2, \tag{10}$$

$$c_t = g_t c_{t-1} + (1-g_t) u_t. \tag{11}$$

3.2. Uniform Gate Initialization

Standard initialization schemes for the gates can prevent the learning of long-term temporal correlations (Tallec & Ollivier, 2018). For example, supposing that a unit in the cell state has constant forget gate value f_t, then the contribution of an input x_t in k time steps will decay by $(f_t)^k$. This gives the unit an effective *decay period* or *characteristic timescale* of $O(\frac{1}{1-f_t})$.[3] Standard initialization of linear layers $\mathcal{L}$ sets the bias term to 0, which causes the forget gate values (2) to concentrate around 0.5. A common trick of setting the forget gate bias to $b_f = 1.0$ (Jozefowicz et al., 2015) does increase the value of the decay period to $\frac{1}{1-\sigma(1.0)} \approx 3.7$. However, this is still relatively small and may hinder the model from learning dependencies at varying timescales easily.

We instead propose to directly control the distribution of forget gates, and hence the corresponding distribution of decay periods. In particular, we propose to simply initialize the value of the forget gate activations f_t according to a uniform distribution $\mathcal{U}(0,1)$[4],

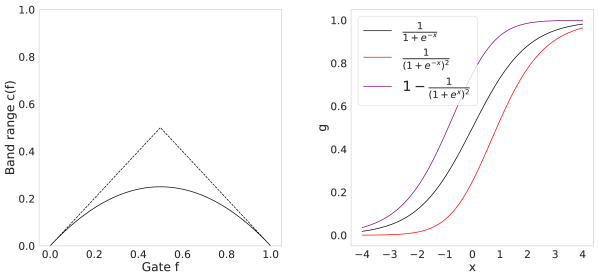
$$b_f \sim \sigma^{-1}(\mathcal{U}[\epsilon, 1-\epsilon]). \tag{12}$$

An important difference between UGI and standard or other (e.g. Tallec & Ollivier, 2018) initializations is that negative forget biases are allowed. The effect of UGI is that all timescales are covered, from units with very high forget activations remembering information (nearly) indefinitely, to those with low activations focusing solely on the incoming input. Additionally, it introduces no additional parameters; it even can have less hyperparameters than the standard gate initialization, which sometimes tunes the forget bias b_f (Jozefowicz et al., 2015). Appendix B.2 and B.3 further discuss the theoretical effects of UGI on timescales.

[2] In our experiments, we found that tying input/forget gates makes negligible difference on downstream performance, consistent with previous findings in the literature (Greff et al., 2016; Melis et al., 2017).

[3] This corresponds to the number of timesteps it takes to decay by $1/e$.

[4] Since $\sigma^{-1}(0) = -\inf$, we use the standard practice of thresholding with a small ϵ for stability.



(a) (b)

Figure 2. **The adjustment function.** (a) An adjustment function $\alpha(f_t)$ satisfying natural properties is chosen to define a band within which the forget gate is refined. (b) The forget gate $f_t(x)$ is conventionally defined with the sigmoid function (black). The refine gate interpolates around the original gate f_t to yield an effective gate g_t within the upper and lower curves, $g_t \in f_t \pm \alpha(f_t)$.

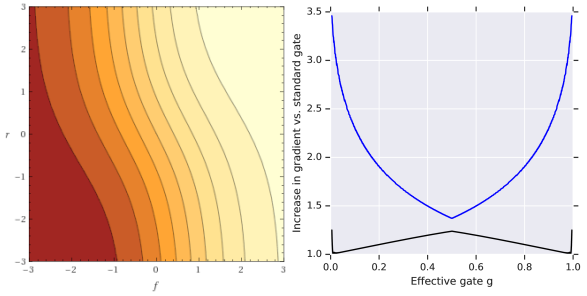
3.3. The URLSTM

The URLSTM requires two small modifications to the vanilla LSTM. First, we present the way the biases of forget gates are initialized in Equation (12) with UGI. Second, the modifications on the standard LSTM equations to compute the refine and effective forget gates are presented in Equations (9)-(11). However, we note that these methods can be used to modify any gate (or more generally, bounded function) in any model. In this context, the URLSTM is simply defined by applying UGI and a refine gate r on the original forget gate f to create an effective forget gate g (Equation (10)). This effective gate is then used in the cell state update (11). Empirically, these small modifications to an LSTM are enough to allow it to achieve nearly binary activations and solve difficult memory problems (Figure 5).

3.4. A Formal Treatment of Refine Gates

Given a gate $f = \sigma(\mathcal{P}_f(x)) \in [0,1]$, the refine gate is an independent gate $r = \sigma(\mathcal{P}_r(x))$ that modulates f to produce a value $g \in [0,1]$ which will be used in place of f downstream. It is motivated by considering how to modify the output of a gate f in a way that promotes gradient-based learning, derived below.

An additive adjustment A root cause of the saturation problem is that the gradient ∇f of a gate can be written solely as a function of the activation value as $f(1-f)$, decays rapidly as f approaches to 0 or 1. Thus when the activation f is past a certain upper or lower threshold,



(a) (b)

Figure 3. **Refine gate activations and gradients.**: (a) Contours of the effective gate g_t as a function of the forget and refine gates f_t, r_t. High effective activations can be achieved with more modest f_t,r_t values. (b) The gradient ∇g_t as a function of effective gate activation g_t. [Black, blue]: Lower and upper bounds on the ratio of the gradient with a refine gate vs. the gradient of a standard gate. For activation values near the extremes, the refine gate can significantly increase the gradients.

learning effectively stops. This problem cannot be fully addressed only by modifying the input of the sigmoid, as in UGI and other techniques, as the gradient will still vanish by backpropagating through the activation function.

Therefore to better control activations near the saturating regime, instead of changing the input to the sigmoid in $f = \sigma(\mathcal{P}(x))$, we consider modifying the output. Modifying the gate with a multiplicative interaction can have unstable learning dynamics since when the gates have very small values, the multiplicative factor may need to be very large to avoid the gradients of the gates shrinking to zero. As a result, we consider adjusting f with an input-dependent additive update $\phi(f,x)$ for some function ϕ, to create an effective gate $g = f + \phi(f,x)$ that will be used in place of f downstream such as in the main state update (1). This sort of additive ("residual") connection is a common technique to increase gradient flow, and indeed was the motivation of the LSTM additive gated update (1) itself (Hochreiter & Schmidhuber, 1997).

Choosing the adjustment function ϕ Although there might be many choices that seem plausible for choosing an appropriate additive update ϕ, we first identify the desirable properties of such a function and then discuss how our refine gate mechanism satisfies those properties.

The desired properties of ϕ emerge considering the applications of the gating mechanisms in recurrent models:

- **Boundedness**: After the additive updates, the activations still need to be bounded between 0 and 1.
- **Symmetricity**: The resulting gating framework should be symmetric around 0, as sigmoid does.
- **Smoothness**: The refining mechanism should be differentiable, since we will be using backpropagation and gradient based optimization methods.

Let us note that, f_t may need to be either increased or decreased, regardless of what value it has. This is because the gradients through the gates can vanish either when the activations get closer to 0 or 1. Therefore, an additive update to f should create an *effective gate activation* g_t in the range $f_t \pm \alpha$ for some α. We assume that the allowed adjustment range, $\alpha = \alpha(f_t)$, needs to be a function of f to keep the g between 0 and 1. Since 0 and 1 are symmetrical in the gating framework, our adjustment rate should also satisfy $\alpha(f) = \alpha(1-f)$.

Figure 2a illustrates the general appearance of $\alpha(f)$ based on aforementioned properties. According to the *Boundedness* property, the adjustment rate should be be upper-bounded by $\min(f, 1-f)$ to ensure that $g \in f \pm \alpha(f)$ is bounded between 0 and 1. As a consequence of this property, its derivatives should also satisfy, $\alpha'(0) \leq 1$ and $\alpha'(1) \geq -1$. Symetricity also implies $\alpha'(f) = -\alpha'(1-f)$, and smoothness implies α' is continuous. The simplest such function satisfying all these properties is the linear $\alpha'(f) = 1 - 2f$, yielding to our choice of adjustment function, $\alpha(f) = f - f^2 = f(1-f)$. However, when f is bounded between 0 and 1, $\alpha(f)$ will be positive.

Recall that the goal is to produce an effective activation $g = f + \phi(f,x)$ such that $g \in f \pm \alpha(f)$ (Figure 2b) given. Our final observation is that the simplest such function ϕ satisfying this is $\phi(f,x) = \alpha(f)\psi(f,x)$ where $\psi(f,x) \in [-1,1]$ decides the sign of adjustment, and it can also change its magnitude as well. The standard method of defining $[-1,1]$-valued differentiable functions is achieved by using a $\tanh$ non-linearity, and this leads to $\phi(f,x) = \alpha(f)(2r-1)$ for another gate $r = \sigma(\mathcal{P}(x))$. The full refine update equation can be given as in Equation (13),

$$\begin{aligned} g &= f + \alpha(f)(2r-1) = f + f(1-f)(2r-1) \\ &= (1-r)\cdot f^2 + r\cdot(1-(1-f)^2) \end{aligned} \tag{13}$$

Equation (13) has the elegant interpretation that the gate r linearly interpolates between the lower band $f - \alpha(f) = f^2$ and the symmetric upper band $f + \alpha(f) = 1 - (1-f)^2$ (Figure 2b). In other words, the original gate f is the coarse-grained determinant of the effective gate g, while the gate r "refines" it.

4. Related Gating Mechanisms

We highlight a few recent works that also propose small gate changes to address problems of long-term or variable-length dependencies. Like ours, they can be applied to any gated update equation.

Tallec & Ollivier (2018) suggest an initialization strategy to capture long-term dependencies on the order of T_{max}, by sampling the gate biases from $b_f \sim \log \mathcal{U}(1, T_{max} - 1)$. Although similar to UGI in definition, *chrono initialization*

(CI) has critical differences in the timescales captured, for example, by using an explicit timescale parameter and having no negative biases. Due to its relation to UGI, we provide a more detailed comparison in Appendix B.3. As mentioned in Section 3.4, techniques such as these that only modify the input to a sigmoid gate do not adequately address the saturation problem.

The Ordered Neuron (ON) LSTM introduced by (Shen et al., 2018) aims to induce an ordering over the units in the hidden states such that "higher-level" neurons retain information for longer and capture higher-level information. We highlight this work due to its recent success in NLP, and also because its novelties can be factored into introducing two mechanisms which only affect the forget and input gates, namely (i) the $\mathrm{cumax} := \mathrm{cumsum} \circ \mathrm{softmax}$ activation function which creates a monotonically increasing vector in $[0,1]$, and (ii) a pair of "master gates" which are ordered by cumax and fine-tuned with another pair of gates.

We observe that these are related to our techniques in that one controls the distribution of a gate activation, and the other is an auxiliary gate with modulating behavior. Despite its important novelties, we find that the ON-LSTM has drawbacks, including speed and scaling issues of its gates. We provide the formal definition and detailed analysis of the ON-LSTM in Appendix B.4. For example, we comment on how UGI can also be motivated as a faster approximation of the cumax activation. We also flesh out a deeper relationship between the master and refine gates and show how they can be interchanged for each other.

We include a more thorough overview of other related works on RNNs in Appendix B.1. These methods are mostly orthogonal to the isolated gate changes considered here and are not analyzed. We note that an important drawback common to all other approaches is the introduction of substantial hyperparameters in the form of constants, training protocol, and significant architectural changes. For example, even for chrono initialization, one of the less intrusive proposals, we experimentally find it to be particularly sensitive to the hyperparameter T_{max} (Section 5).

4.1. Gate Ablations

Our insights about previous work with related gate components allow us to perform extensive ablations of our contributions. We observe two independent axes of variation, namely, activation function/initialization (cumax, constant bias sigmoid, CI, UGI) and auxiliary modulating gates (master, refine), where different components can be replaced with each other. Therefore we propose several other gate combinations to isolate the effects of different gating mechanisms. We summarize a few ablations here; precise details are given in Appendix B.5. **O-**: Ordered gates. A natural simplification of the main idea of ON-LSTM, while keeping the hierarchical bias on the forget activations, is to simply drop the auxiliary master gates and define f_t, i_t (2)-(3) using the cumax activation function. **UM**: UGI master gates. This variant of the ON-LSTM's gates ablates the cumax operation on the master gates, replacing it with a sigmoid activation and UGI which maintains the same initial distribution on the activation values. **OR**: Refine instead of master. A final variant in between the UR gates and the ON-LSTM's gates combines cumax with refine gates. In this formulation, as in UR gates, the refine gate modifies the forget gate and the input gate is tied to the effective forget gate. The forget gate is ordered using cumax.

Table 1. **Summary of gate ablations.** Summary of gating mechanisms considered in this work as applied to the forget/input gates of recurrent models. Some of these ablations correspond to previous work. -- standard LSTMs, C- (Tallec & Ollivier, 2018), and OM (Shen et al., 2018)

Name	Initialization/Activation	Auxiliary Gate
`--`	Standard initialization	N/A
`C-`	Chrono initialization	N/A
`O-`	cumax activation	N/A
`U``-`	Uniform initialization	N/A
`-`**`R`**	Standard initialization	Refine gate
`OM`	cumax activation	Master gate
`U``M`	Uniform initialization	Master gate
`O`**`R`**	cumax activation	Refine gate
`UR`	Uniform initialization	Refine gate

Table 1 summarizes the gating modifications we consider and their naming conventions. Note that we also denote the ON-LSTM method as **OM** for mnemonic ease. Finally, we remark that all methods here are controlled with the same number of parameters as the standard LSTM, aside from OM and UM which use an additional $\frac{1}{2C}$-fraction parameters where C is the downsize factor on the master gates (Appendix B.4). $C = 1$ unless noted otherwise.

5. Experiments

We first perform full ablations of the gating variants (Section 4.1) on two common benchmarks for testing memory models: synthetic memory tasks and pixel-by-pixel image classification tasks. We then evaluate our main method on important applications for recurrent models including language modeling and reinforcement learning, comparing against baselines from literature where appropriate.

The main claims we evaluate for each gating component are (i) the refine gate is more effective than alternatives (the master gate, or no auxiliary gate), and (ii) UGI is more effective than standard initialization for sigmoid gates. In particular, we expect the *R gate to be more effective than *M or *- for any primary gate *, and we expect U* to be better than -* and comparable to O* for any auxiliary gate *.

The standard LSTM (--) uses forget bias 1.0 (Section

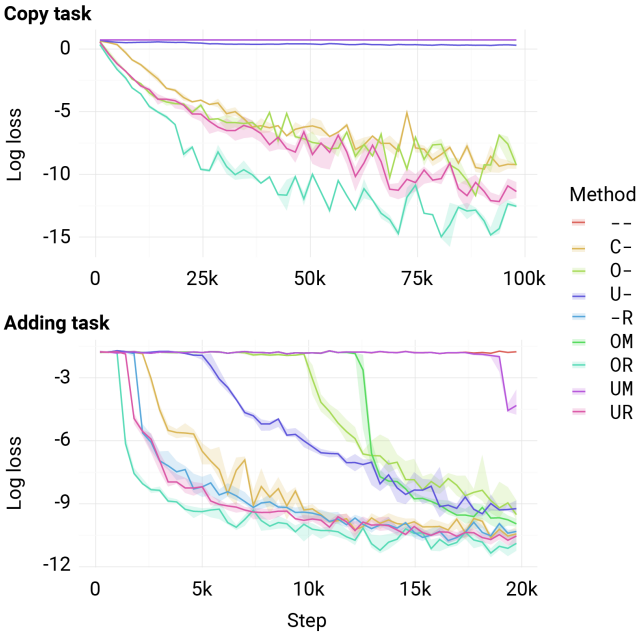


Figure 4. **Performance on synthetic memory:** *Copy task* using sequences of length 500. Several methods including standard gates fail to make any progress (overlapping flat curves at baseline). Note that methods that combine the refine gate with a range of gate values (OR, UR) perform best. But the refine gate on its own does not perform well. *Adding task* using sequences of length 2000. Most methods eventually make progress, but again methods that combine the refine gate with a range of gate values (OR, UR) perform best.

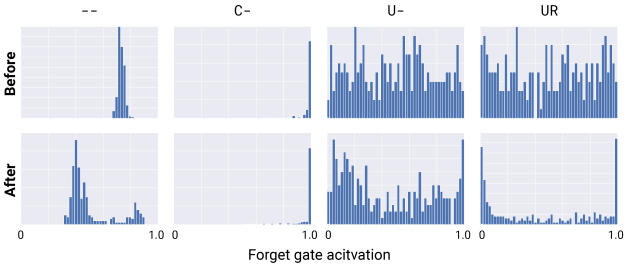


Figure 5. **Distribution of forget gate activations before and after training**. For the Copy task. We show the distribution of activations f_t for four methods: -- cannot learn large enough f_t and makes no progress on the task. C- initializes with extremal activations which barely change during training. U- makes progress by encouraging a range of forget gate values, but this distribution does not change significantly during training due to saturation. UR starts with the same distribution as U- but is able to learn extreme gate values, which allows it to access the distal inputs, as necessary for this task. Appendix E.1 shows a reverse task where UR is able to un-learn from a saturated regime.

2.2). When chrono initialization is used and not explicitly tuned, we set T_{max} to be proportional to the hidden size. This heuristic uses the intuition that if dependencies of length T exist, then so should dependencies of all lengths $\leq T$. Moreover, the amount of information that can be remembered is proportional to the number of hidden units.

All of our benchmarks have prior work with recurrent baselines, from which we used the same models, protocol, and hyperparameters whenever possible, changing only the gating mechanism without doing any additional tuning for the refine gating mechanisms. Full protocols and details for all experiments are given in Appendix D.

5.1. Synthetic Memory Tasks

Our first set of experiments is on synthetic memory tasks (Hochreiter & Schmidhuber, 1997; Arjovsky et al., 2016) that are known to be hard for standard LSTMs to solve. For these tasks, we used single layer models with 256 hidden units, trained using Adam with learning rate 10^{-3}.

Copy task. The input is a sequence of $N+20$ digits where the first 10 tokens $(a_0,a_1,...,a_9)$ are randomly chosen from $\{1,...,8\}$, the middle N tokens are set to 0, and the last ten tokens are 9. The goal of the recurrent model is to output $(a_0,...,a_9)$ in order on the last 10 time steps, whenever the cue token 9 is presented. We trained our models using cross-entropy with baseline loss $\log(8)$ (Appendix D.1).

Adding task. The input consists of two sequences: 1. N numbers $(a_0,...,a_{N-1})$ sampled independently from $\mathcal{U}[0,1]$ 2. an index $i_0 \in [0, N/2)$ and $i_1 \in [N/2, N)$, together encoded as a two-hot sequence. The target output is $a_{i_0}+a_{i_1}$ and models are evaluated by the mean squared error with baseline loss $1/6$.

Figure 4 shows the loss of various methods on the Copy and Adding tasks. The only gate combinations capable of solving Copy completely are OR, UR, O-, and C-. This confirms the mechanism of their gates: these are the only methods capable of producing high enough forget gate values either through the cumax non-linearity, the refine gate, or extremely high forget biases. U- is the only other method able to make progress, but converges slower as it suffers from gate saturation without the refine gate. -- makes no progress. OM and UM also get stuck at the baseline loss, despite OM's cumax activation, which we hypothesize is due to the suboptimal magnitudes of the gates at initialization (Appendix B.4). On the Adding task, every method besides -- is able to eventually solve it, with all refine gate variants fastest.

Figure 5 shows the distributions of forget gate activations of sigmoid-activation methods, before and after training on the Copy task. It shows that activations near 1.0 are important for a model's ability to make progress or solve this task, and that adding the refine gate makes this significantly easier.

5.2. Pixel-by-pixel Image Classification

These tasks involve feeding a recurrent model the pixels of an image in a scanline order before producing a classification

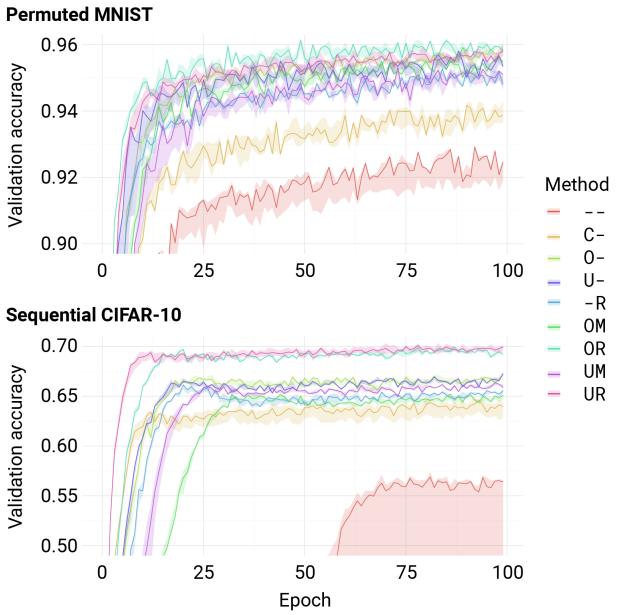


Figure 6. **Performance on pixel-by-pixel image classification.** Performance is consistent with synthetic tasks. -- performs the worst. Other gating variants improve performance. Note that methods that combine the refine gate with a range of gate values (OR, UR) perform best.

label. We test on the sequential MNIST (sMNIST), permuted MNIST (pMNIST) (Le et al., 2015), and sequential CIFAR-10 (sCIFAR-10) tasks. Each LSTM method was ran with a learning rate sweep with 3 seeds each. We found that many methods were quite unstable, with multiple seeds diverging. Figure 6 shows the accuracy curves of each method at their best *stable* learning rate. The basic LSTM is noticeably worse than all of the others. This suggests that any of the gate modifications, whether better initialization, cumax non-linearity, or master or refine gates, are better than standard gates especially when long-term dependencies are present. Additionally, the uniform gate initialization methods are generally better than the ordered and chrono initialization, and the refine gate performs better than the master gate. Table 2 compares the test accuracy of our main model against other models from the literature. In addition, we tried variants of GRUs and the addition of a generic regularization technique—we chose Zoneout (Krueger et al., 2016) with default hyperparameters ($z_c = 0.5$, $z_h = 0.05$). This combination even outperformed non-recurrent models on sequential MNIST and CIFAR-10.

From Sections 5.1 and 5.2, we draw a few conclusions about the comparative performance of different gate modifications. First, the refine gate is consistently better than comparable master gates. C- solves the synthetic memory tasks but is worse than any other variant outside of those. We find ordered (cumax) gates to be effective, but speed issues prevent us from using them in more complicated tasks. UR gates are consistently among the best performing and most stable.

Table 2. **Comparison to prior methods for pixel-by-pixel image classification.** Test acc. on pixel-by-pixel image classification benchmarks. Top: Recurrent baselines and variants. Middle: Non-recurrent sequence models with global receptive field. r-LSTM has 2-layers with an auxiliary loss. Bottom: Our methods.

Method	sMNIST	pMNIST	sCIFAR-10
LSTM (ours)	98.9	95.11	63.01
Dilated GRU (Chang et al., 2017)	99.0	94.6	-
IndRNN (Li et al., 2018a)	99.0	96.0	-
r-LSTM (Trinh et al., 2018)	98.4	95.2	72.2
Transformer (Trinh et al., 2018)	98.9	97.9	62.2
Temporal ConvNet (Bai et al., 2018a)	99.0	97.2	-
TrellisNet (Bai et al., 2018b)	99.20	**98.13**	73.42
URLSTM	**99.28**	96.96	71.00
URLSTM + Zoneout (Krueger et al., 2016)	99.21	97.58	74.34
URGRU + Zoneout	99.27	96.51	**74.4**

Table 3. **Language modelling results.** Perplexities on the WikiText-103 dataset.

Method	Valid	Test
--	34.3	35.8
C-	35.0	36.4
C- $T_{max} = 8$	34.3	36.1
C- $T_{max} = 11$	34.6	35.8
OM	34.0	34.7
U-	33.8	34.9
UR	**33.6**	**34.6**

5.3. Language Modeling

We consider word-level language modeling on the WikiText-103 dataset, where (i) the dependency lengths are much shorter than in the synthetic tasks, (ii) language has an implicit hierarchical structure and timescales of varying lengths. We evaluate our gate modifications against the exact hyperparameters of a SOTA LSTM-based baseline (Rae et al., 2018) without additional tuning (Appendix D). Additionally, we compare against ON-LSTM, which was designed for this domain (Shen et al., 2018), and chrono initialization, which addresses dependencies of a particular timescale as opposed to timescale-agnostic UGI methods. In addition to our default hyperparameter-free initialization, we tested models with the chrono hyperparameter T_{max} manually set to 8 and 11, values previously used for language modeling to mimic fixed biases of about 1.0 and 2.0 respectively (Tallec & Ollivier, 2018).

Table 3 shows Validation and Test set perplexities for various models. We find that OM, U-, and UR improve over -- with no additional tuning. However, although OM was designed to capture the hierarchical nature of language with the cumax activation, it does not perform better than U- and UR. Appendix D, Figure 11 additionally shows validation perplexity curves, which indicate that UR overfits less than the other methods.

The chrono initialization using our aforementioned initial-

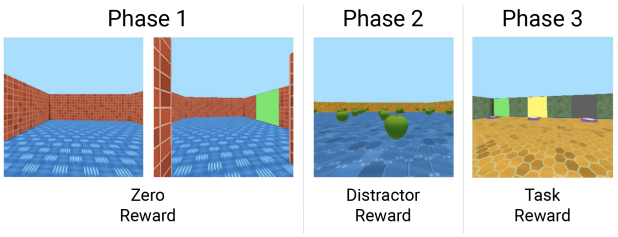


Figure 7. **Active match.** Hung et al. (2018). The agent navigates a 3D world using observations from a first person camera. The task has three phases. In phase 1, the agent must search for a colored cue. In phase 2, the agent is exposed to apples which give distractor rewards. In phase 3, the agent must correctly recall the color of the cue and pick the sensor near the corresponding color to receive the task reward. An episode lasts between 450 and 600 steps, requiring long-term memory and credit assignment.

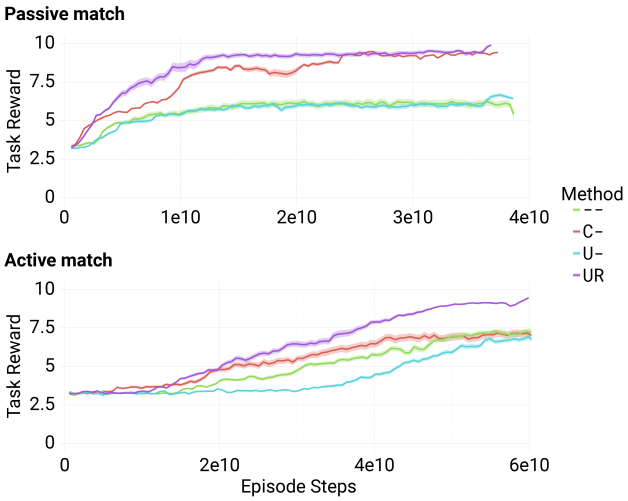


Figure 8. **Performance on reinforcement learning tasks that require memory.** We evaluated the image matching tasks from Hung et al. (2018), which test memorization and credit assignment, using an A3C agent (Mnih et al., 2016) with an LSTM policy core. We observe that general trends from the synthetic tasks (Section (5.1)) transfer to this reinforcement learning setting.

ization strategy makes biases far too large. While manually tweaking the T_{max} hyperparameter helps, it is still far from any UGI-based methods. We attribute these observations to the nature of language having dependencies on multiple widely-varying timescales, and that UGI is enough to capture these without resorting to strictly enforced hierarchies such as in OM.

5.4. Reinforcement Learning Memory Tasks

In most partially observable reinforcement learning (RL) tasks, the agent can observe only part of the environment at a time and thus requires a memory to summarize what it has seen previously. However, designing memory architectures for reinforcement learning problems has been a challenging task (Oh et al., 2016; Wayne et al., 2018). Many memory architectures for RL use an LSTM component to summarize what an agent has seen.

We investigated if changing the gates of these LSTMs can improve the performance of RL agents, especially on difficult tasks involving memory and long-term credit assignment. We chose the **Passive match** and **Active match** tasks from Hung et al. (2018) using A3C agents (Mnih et al., 2016). See Figure 7 for a description of Active match. Passive match is similar, except the agent always starts facing the colored cue. As a result, Passive Match only tests long term memory, not long-term credit assignment. Only the final task reward is reported.

Hung et al. (2018) evaluated agents with different recurrent cores: basic LSTM, LSTM+Mem (an LSTM with memory), and RMA (which also uses an LSTM core), and found the standard LSTM was not able to solve these tasks. We modified the LSTM agent with our gate mechanisms. Figure 8 shows the results of different methods on the Passive match and Active match tasks with distractors. These tasks are structurally similar to the synthetic tasks (Sec. 5.1) requiring retrieval of a memory over hundreds of steps to solve the task, and we found that those trends largely transferred to the RL setting even with several additional confounders present such as agents learning via RL algorithms, being required to learn relevant features from pixels rather than being given the relevant tokens, and being required to explore in the Active Match case.

We found that the UR gates substantially improved the performance of the basic LSTM on both Passive Match and Active Match tasks with distractor rewards. The URLSTM was the was the only method able to get near optimal performance on both tasks, and achieved similar final performance to the LSTM+Mem and RMA agents reported in (Hung et al., 2018).

5.5. Additional Results and Experimental Conclusions

Appendix (E.1) shows an additional synthetic experiment investigating the effect of refine gates on saturation. Appendix (E.3) has results on a program execution task, which is interesting for having explicit long and variable-length dependencies and hierarchical structure. It additionally shows another very different gated recurrent model where the UR gates provide consistent improvement.

Finally, we would like to comment on the longevity of the LSTM, which for example was frequently found to outperform newer competitors when better tuned (Melis et al., 2017; Merity, 2019). Although many improvements have been suggested over the years, none have been proven to be as robust as the LSTM across an enormously diverse range of sequence modeling tasks. By experimentally starting from

well-tuned LSTM baselines, we believe our simple isolated gate modifications to actually be robust improvements. In Appendix B.3 and B.4, we offer a few conclusions for the practitioner about the other gate components considered based on our experimental experience.

6. Discussion

In this work, we introduce and evaluate several modifications to the ubiquitous gating mechanism that appears in recurrent neural networks. We describe methods that improve on the standard gating method by alleviating problems with initialization and optimization. The mechanisms considered include changes on independent axes, namely initialization/activations and auxiliary gates, and we perform extensive ablations on our improvements with previously considered modifications. Our main gate model robustly improves on standard gates across many different tasks and recurrent cores, while requiring less tuning. Finally, we emphasize that these improvements are entirely independent of the large body of research on neural network architectures that use gates, and hope that these insights can be applied to improve machine learning models at large.

References

Arjovsky, M., Shah, A., and Bengio, Y. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pp. 1120–1128, 2016.

Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Bai, S., Kolter, J. Z., and Koltun, V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018a.

Bai, S., Kolter, J. Z., and Koltun, V. Trellis networks for sequence modeling. *arXiv preprint arXiv:1810.06682*, 2018b.

Bengio, Y., Simard, P., Frasconi, P., et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Chandar, S., Sankar, C., Vorontsov, E., Kahou, S. E., and Bengio, Y. Towards non-saturating recurrent units for modelling long-term dependencies. *arXiv preprint arXiv:1902.06704*, 2019.

Chang, S., Zhang, Y., Han, W., Yu, M., Guo, X., Tan, W., Cui, X., Witbrock, M., Hasegawa-Johnson, M. A., and Huang, T. S. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 77–87, 2017.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Chung, J., Ahn, S., and Bengio, Y. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

Dai, Z., Yang, Z., Yang, Y., Cohen, W. W., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pp. 1406–1415, 2018.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

Gulcehre, C., Moczulski, M., Denil, M., and Bengio, Y. Noisy activation functions. In *International conference on machine learning*, pp. 3059–3068, 2016.

Gulcehre, C., Chandar, S., and Bengio, Y. Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*, 2017.

Henaff, M., Szlam, A., and LeCun, Y. Recurrent orthogonal networks and long-memory tasks. *arXiv preprint arXiv:1602.06662*, 2016.

Hochreiter, S. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91 (1), 1991.

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

Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

Hung, C.-C., Lillicrap, T., Abramson, J., Wu, Y., Mirza, M., Carnevale, F., Ahuja, A., and Wayne, G. Optimizing agent behavior over long time scales by transporting value. *arXiv preprint arXiv:1810.06721*, 2018.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with Gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Jozefowicz, R., Zaremba, W., and Sutskever, I. An empirical exploration of recurrent network architectures. In *International Conference on Machine Learning*, pp. 2342–2350, 2015.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *The International Conference on Learning Representations (ICLR)*, 2018.

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

Kočiskỳ, T., Schwarz, J., Blunsom, P., Dyer, C., Hermann, K. M., Melis, G., and Grefenstette, E. The NarrativeQA reading comprehension challenge. *Transactions of the Association for Computational Linguistics*, 6:317–328, 2018.

Koutnik, J., Greff, K., Gomez, F., and Schmidhuber, J. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

Krueger, D., Maharaj, T., Kramár, J., Pezeshki, M., Ballas, N., Ke, N. R., Goyal, A., Bengio, Y., Courville, A., and Pal, C. Zoneout: Regularizing RNNs by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

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

Li, S., Li, W., Cook, C., Zhu, C., and Gao, Y. Independently recurrent neural network (IndRNN): Building a longer and deeper RNN. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5457–5466, 2018a.

Li, Z., He, D., Tian, F., Chen, W., Qin, T., Wang, L., and Liu, T.-Y. Towards binary-valued gates for robust LSTM training. *arXiv preprint arXiv:1806.02988*, 2018b.

Maddison, C. J., Mnih, A., and Teh, Y. W. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. *arXiv preprint arXiv:1707.05589*, 2017.

Merity, S. Single headed attention rnn: Stop thinking with your head. *arXiv preprint arXiv:1911.11423*, 2019.

Merity, S., Xiong, C., Bradbury, J., and Socher, R. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Oh, J., Chockalingam, V., Singh, S., and Lee, H. Control of memory, active perception, and action in Minecraft. *arXiv preprint arXiv:1605.09128*, 2016.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

Rae, J. W., Dyer, C., Dayan, P., and Lillicrap, T. P. Fast parametric learning with activation memorization. *arXiv preprint arXiv:1803.10049*, 2018.

Santoro, A., Faulkner, R., Raposo, D., Rae, J., Chrzanowski, M., Weber, T., Wierstra, D., Vinyals, O., Pascanu, R., and Lillicrap, T. Relational recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 7299–7310, 2018.

Shen, Y., Tan, S., Sordoni, A., and Courville, A. Ordered neurons: Integrating tree structures into recurrent neural networks. *arXiv preprint arXiv:1810.09536*, 2018.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

Tallec, C. and Ollivier, Y. Can recurrent neural networks warp time? *arXiv preprint arXiv:1804.11188*, 2018.

Trinh, T. H., Dai, A. M., Luong, M.-T., and Le, Q. V. Learning longer-term dependencies in RNNs with auxiliary losses. *arXiv preprint arXiv:1803.00144*, 2018.

van der Westhuizen, J. and Lasenby, J. The unreasonable effectiveness of the forget gate. *arXiv preprint arXiv:1804.04849*, 2018.

Wayne, G., Hung, C.-C., Amos, D., Mirza, M., Ahuja, A., Grabska-Barwinska, A., Rae, J., Mirowski, P., Leibo, J. Z., Santoro, A., et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.

Weston, J., Chopra, S., and Bordes, A. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Zaremba, W. and Sutskever, I. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

Zilly, J. G., Srivastava, R. K., Koutník, J., and Schmidhuber, J. Recurrent highway networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 4189–4198. JMLR. org, 2017.