IMPROVING THE GATING MECHANISM OF RECURRENT NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

In this work, we revisit the gating mechanisms widely used in various recurrent and feedforward networks such as LSTMs, GRUs, or highway networks. These gates are meant to control information flow, allowing gradients to better propagate back in time for recurrent models. However, to propagate gradients over very long temporal windows, they need to operate close to their saturation regime. We propose two independent and synergistic modifications to the standard gating mechanism that are easy to implement, introduce no additional hyper-parameters, and are aimed at improving learnability of the gates when they are close to saturation. Our proposals are theoretically justified, and we show a generic framework that encompasses other recently proposed gating mechanisms such as chrono-initialization (Tallec & Ollivier, 2018) and master gates (Shen et al., 2018). We perform systematic analyses and ablation studies on the proposed improvements and evaluate our method on a wide range of applications including synthetic memorization tasks, sequential image classification, language modeling, and reinforcement learning. Empirically, our proposed gating mechanisms robustly increase the performance of recurrent models such as LSTMs, especially on tasks requiring long temporal dependencies.

1 INTRODUCTION

Recurrent neural networks are a standard machine learning tool for interacting with sequential data. Broadly speaking, RNNs represent a learned model which can be used to sweep over input data x_t to produce a sequence of recurrent states h_t. Information is passed between iterations of this mapping by passing the recurrent state h_t into the subsequent mapping. At a high level we can then view RNNs as a neural network architecture in which each sequential application of the network computes the mapping $(x_t,h_{t-1})\mapsto h_t$.

This mapping is typically optimized by defining a supervised loss on the outputs h_t (frequently via another mapping which produces predictions y_t from the recurrent state) and using backpropagation through time (BPTT) to compute gradients with respect to the parameters of the RNN. When optimized in this fashion, however, these networks are often subject to the "vanishing gradient problem" wherein changes to the parameters that control long term dependencies become vanishingly small (see e.g. Pascanu et al., 2013). In order to attack this problem *gating* mechanisms were introduced. These have proven proven crucial to the success of RNNs, leading to popular instantiations such as Long Short-Term Memory (LSTM; Hochreiter & Schmidhuber, 1997) networks and Gated Recurrent Units (GRU; Chung et al., 2014). Furthermore, it has been shown that the use of gates is fundamental from a theoretical perspective by modeling a dynamical system that is invariant to changes in time scale (Tallec & Ollivier, 2018).

However, although the addition of gates helps enormously in alleviating some optimization problems, it has been observed that the basic gated recurrence equation alone is still insufficient for capturing different types of dependencies—especially so for problems with long timescales. These problems have traditionally been addressed by introducing new components to the basic RNN setup. Some techniques include: controlling the stability of the update function u (Arjovsky et al., 2016; Li et al., 2018a), stacking layers in a hierarchy (Chung et al., 2016), adding skip connections and dilations (Koutnik et al., 2014; Chang et al., 2017), using an external memory (Graves et al., 2014; Weston et al., 2014; Wayne et al., 2018; Gulcehre et al., 2017), auxiliary semi-supervision (Trinh et al., 2018), and more. However, these approaches have not been widely adopted over the standard LSTM as they are often specialized for certain tasks, are not as robust, and introduce additional complexity. Recently the transformer model has been successful in many applications areas such as NLP (Radford et al., 2019; Dai et al., 2019). However, recurrent neural networks are still used to a large degree due their faster inference without the need to maintain the entire sequence in memory.

We address two particular problems with the standard gating mechanism of recurrent models. Gating works by employing a *forget gate* and *input gate*, sometimes tied to be one minus the other, whose values are in $(0,1)$. They control how fast information is forgotten or allowed into the memory cell. Removing the forget gate, hence forcing the model not to forget, results in a corruption of the memory, whereas all inputs get added on top

of each other making it impossible to recover them. Making effective use of the forget gate is crucial to have a well behaved RNN (Gers et al., 1999). Particularly, when the gates are tied, their behaviour is that of a low-pass filter, where the gate decides the time-scale to which the unit will respond. Typical initialization of the gates are relatively concentrated, meaning that initially the neural network has a limited range of timescales. Our first proposal, that we refer to as *uniform gate initialization* (Section 2.1), is to directly initialize the activations of these gates from a distribution that captures a wider spread of dependency lengths. This allows gradients to flow better through time, with an implicit trade-off between precision and how far in the past they reach.

However, while initially the gates should span multiple timescales by filtering the input signal, after training their behaviour should converge to be close to binary. Particularly, if the gates are binary the amount of noise or smoothing can be reduced, and the model can respond to discrete events. But learning gates that effectively operate in their saturation regime is difficult because of vanishing gradients. We derive a different formulation of the gates that uses an auxiliary *refine gate* to modulate the main gate. This allows for a wider range of activation without having gradients vanish as fast, leaving the behaviour of the gate to still be learnable. We show theoretically and empirically the implication of our formulation. Combining these two modifications yields our proposed model, which we call the UR gating mechanism.

We empirically evaluate our modifications to several alternative changes from the literature on several benchmarks for recurrent models. These benchmarks include synthetic long-term dependency tasks, sequential pixel-level image classification, language modeling, program execution, and reinforcement learning. We find that our proposed gating mechanism frequently improves significantly over well-tuned vanilla LSTMs, especially on long-term dependency tasks. Additionally, as a modification of only the gating component, the alteration of the gating mechanism are minimal, lightweight and do not introduce any new hyper-parameters.

2 Gated Recurrent Neural Networks

The use of *forget gates* in RNNs has proven to be an incredibly important mechanism used to avoid the "vanishing gradient" problem (Bengio et al., 1994; Hochreiter et al., 2001). However, their use does introduce additional complexity due to the tendency of learned gates to saturate, which in turn leads to memory only over limited timescale ranges. In this work we introduce an alternative formulation for the gating mechanism which we will show is able avoid this issue. While we will focus primarily on the LSTM, due to its status as the most popular recurrent model, these techniques can be used in any model that makes similar use of forget gates.

A typical LSTM is an RNN whose state is represented by a tuple (h_t, c_t) consisting of the recurrent state and a cell state. Given inputs x_t and the previous state-tuple, a typical LSTM computes the forget, input, and output gates f_t, i_t, and o_t. These gates are used in order to produce the following cell and recurrent states c_t and h_t as described in equations (1–5). Here we have introduced $\mathcal{L}(...)$ to correspond to a parameterized linear projection of its inputs. $\sigma(\cdot)$ refers to the commonly used sigmoid activation functions for the gates and biases for the forget, input and output gates are b_f, b_i, b_o. Intuitively the forget gate represents the propensity of the cell state to forget information from previous iterations and the input gate correspondingly controls the ability of the cell state to incorporate new information from the current input. Similarly the output gate controls the inclusion of information in the recurrent state. As originally proposed by Hochreiter & Schmidhuber (1997) the cell state of an LSTM serves as an "error carousel" that allows the gradients to propagate through time without vanishing due to the interaction between the input and forget gates.

$$f_t = \sigma(\mathcal{L}_i(x_t, h_{t-1}) + b_f) \tag{1}$$

$$i_t = \sigma(\mathcal{L}_f(x_t, h_{t-1}) + b_i) \tag{2}$$

$$o_t = \sigma(\mathcal{L}_o(x_t, h_{t-1}) + b_o) \tag{3}$$

$$c_t = f_t c_{t-1} + i_t \tanh(\mathcal{L}_c(x_t, h_{t-1}) + b_c) \tag{4}$$

$$h_t = o_t \tanh(c_t) \tag{5}$$

Although the gates of the LSTM were initially motivated as a binary mechanism, switching on or off to allow information and gradients to pass through, in reality, they are frequently not close to 0 or 1. However, learning to keep information in the memory for long time delays necessarily requires that the values of these gates have saturated values. This can be seen from Equation 4, where a value of f_t near zero is required to maintain the value of the cell state. Due to the well-known "saturating property" of the sigmoid activation function, however, we can see that values near the extrema result in extremely small gradients. This results in two difficult problems for gradient-based learning of the gates. First, it effectively places a threshold on the activations of gates because they stop changing once they attain a relatively large or small value—which can prove problematic for recurrent models that may need to learn extremely long dependencies requiring very high activations. Secondly, this behavior also makes it very unlikely that such saturation once obtained can be be unlearned (Gulcehre et al., 2016).

In what follows, we propose two solutions which work in tandem to address these issues. The first relies on the *uniform* initialization of the bias term within the gates, and the second proposes an alternative *refine* gate formulation which avoids the saturation problem. The main method we consider in this paper simply combines these two independent ideas, resulting in what we refer to as **UR-**gates. As we will see, both improvements

are simple conceptually and in implementation, hyperparameter-free, and improve upon previously proposed solutions in practice. See Figure 1 for an illustration of our proposed LSTM modification.

2.1 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 decayed by the rate of f_t^k. This gives the unit an effective *decay period* or *characteristic timescale* of $O(\frac{1}{1-f_t})$. Initializations of the linear gate layers often set the bias term to 0, which causes the forget gate values (1) to concentrate around $1/2$. A common trick of setting the forget gate bias to $b_f = 1.0$ 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 moreover fixed, hindering the model from easily learning dependencies at varying timescales.

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 gates (f_t) according to a uniform distribution $\mathcal{U}(0,1)$, which we will refer to as Uniform Gate Initialization (UGI). This can be implemented by sampling the forget (and input) gate bias according to the following equations:

$$b_f = \sigma^{-1}(\mathcal{U}[\epsilon, 1-\epsilon]) \qquad b_i = -b_f \tag{6}$$

(see Appendix C.5 for ϵ). For example, an illustrative difference between UGI and standard initialization or other proposed initializations (e.g. Tallec & Ollivier, 2018) is that negative 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; in fact it can have fewer parameters than standard gate initialization, as the forget bias b_f is sometimes treated as a hyperparameter.

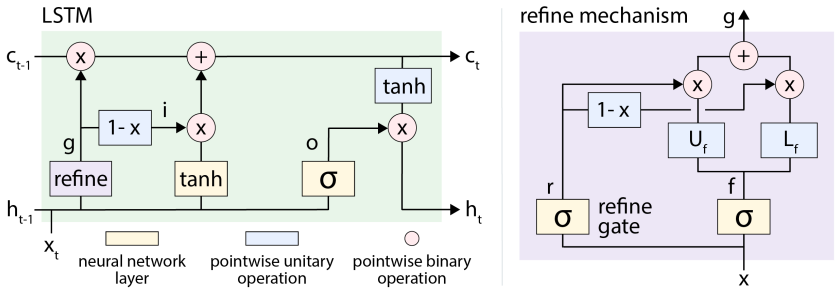


Figure 1: **LSTM with refine mechanism** The refine mechanism replaces the forget gate and input gate in the standard LSTM. The refine mechanism contains a forget gate f_t and refine gate r_t, which interpolates between an upperbound U_{f_t} and lowerbound L_{f_t} function of the forget gate. The resulting effective forget gate g_t is then used in place of f_t in the state update (4).

2.2 THE REFINE GATE

The root of the saturation problem is that the gradient of the gates, e.g. $f_t = \sigma(\mathcal{L}(x_t) + b)$, which can be written solely as a function of the activation value as $f_t(1-f_t)$, decays rapidly as f_t approaches 0 or 1. Thus when the activation f_t is past a certain upper or lower threshold, learning effectively stops. This can be problematic for a technique such as UGI, which although is useful for creating very small and large activations immediately, those gates will be stuck in the saturated regime. This problem cannot be fully addressed only by modifying the input to the sigmoid, as is common, as the gradient will still vanish by backpropagating through the nonlinearity.

Therefore to better control activations near the saturating regime, instead of changing the input to the sigmoid gate $f_t = \sigma(\mathcal{L}(x_t) + b)$, we consider modifying the output. In particular, we propose to adjust f_t with an input-dependent update to create an effective gate $g_t = f_t + \phi(f_t, x_t)$, which will be used in the final state update (12). 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 itself (Hochreiter & Schmidhuber, 1997).

Although many choices seem plausible for selecting the function ϕ, we reason from desirable properties of the final effective gate g_t to deduce a principled update ϕ. First, note that f_t might want to be increased or decreased, regardless of what its value is. For example, given a large activation f_t near saturation, it may need to be even higher to address long-term dependencies; alternatively, if it is too high by initialization or needs to unlearn previous behavior, it may need to decrease. Therefore, we would like to create an effective gate g_t within $f_t \pm c$ for some c. Furthermore c should have the following natural properties:

Proposition 1. *The additive adjustment range c should satisfy **Validity**: The allowable adjustment range $c=c(f_t)$ is a function of f_t, as g_t must lie in $(0,1)$. In particular, $c(f_t)\leq\min(f_t,1-f_t)$. **Symmetry**: Since 0 and 1 are completely symmetrical in the gating framework, $c(f_t)=c(1-f_t)$. **Differentiability**: $c(f_t)$ will be used in backpropagation, requiring $c\in C^1(\mathbb{R})$.*

Figure 2a illustrates the general appearance of $c(f_t)$ based on these properties. In particular, they imply respectively that $c'(0)\leq 1$ and $c'(1)\geq -1$, $c'(f_t)=-c'(1-f_t)$, and c' is continuous. The simplest such function satisfying these is the linear $c'(f_t)=1-2f_t$, yielding $c(f_t)=f_t-f_t^2=f_t(1-f_t)$.

Given such a $c(f_t)$, we recall that the goal is to produce an effective gate $g_t\in[f_t-c(f_t),f_t\ +\ c(f_t)]$ (Figure 2b) using some input-dependent update $g_t=f_t+\phi(f_t,x_t)$. Our final observation is that the simplest such function of this form is $\phi(f_t,x_t)=c(f_t)\psi(f_t,x_t)$ for some $[-1,\ 1]$-valued function ψ. This leads to $\phi(f_t,x_t)=c(f_t)(2r_t-1)$ for another gate $r_t=\sigma(\mathcal{L}(x_t))$. The full update is given in Equation (7),

$$g_t=f_t+c(f_t)(2r_t-1)=f_t+f_t(1-f_t)(2r_t-1)=(1-r_t)\cdot f_t^2+r_t\cdot(1-(1-f_t)^2) \tag{7}$$

Equation (7) has the elegant interpretation that the gate r linearly interpolates between the lower band $f_t-c(f_t)=f_t^2$ and the symmetric upper band $f_t+c(f_t)=1-(1-f_t)^2$ (Figure 2b). In other words, the forget gate f_t is the coarse-grained determinant of the effective gate g_t, while the refine gate r_t "refines" it. This allows the effective gate g_t to reach much higher and lower activations than the constituent sigmoid gates f_t and r_t (Figure 2c), bypassing the saturating gradient problem. For example, this allows the effective forget gate to reach $g_t=0.99$ when the forget gate is only $f_t=0.9$.

As a aside, we note that instead of choosing a single function $c(f_t)$ defining the allowable range for g_t, we could have defined a separate upper and lower band $g_t\in[f_t-c_l(f_t),f_t+c_u(f_t)]$. However, tying $c_l=c_u$ has the additional intuitive property that when the refine gate is centered ($r_t=1/2$), the effective gate $g_t=1/2(f_t-c(f_t))+1/2(f_t+c(f_t))=f_t$ is equal to the forget gate.

Formally, the full mechanism of the refine gate is defined in Equation (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, $i_t=1-\hat{f}_t$. However, we note that our methods can be combined in different ways; for example, the input gate can also be modified with its own refine gate.

$$f_t=\sigma(\mathcal{L}_f(x_t,h_{t-1})+b_f) \tag{8}$$

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

$$\hat{f}_t=r_t\cdot(1-(1-f_t)^2)+(1-r_t)\cdot f_t^2 \tag{10}$$

$$=2r_t\cdot f_t+(1-2r_t)\cdot f_t^2 \tag{11}$$

$$c_t=\hat{f}_t c_{t-1}+i_t\tanh(\mathcal{L}_f(x_t,h_{t-1})+b_c), \tag{12}$$

Figure 2b illustrates how the refine gate r_t changes the effective gate f_t to produce an effective gate g_t within a band. In particular, Figure 2d considers the magnitude of the gradient $\|\nabla_x g_t\|$ as a function of the activation g_t. For example, this function is $g_t(1-g_t)$ when $g_t=\sigma(x_t)$ is the basic gate. However, this gradient increases substantially when the refine gate is introduced. Figure 2d plots the minimum and maximum multiplicative increase over the case of a standard gate, illustrating how the inclusion of a refine gate promotes better gradient flow and optimization – increasingly so in the saturating regime near $g_t=\{0,1\}$.

2.3 Effect on timescales

Finally, we remark that although we have been working directly with gate activation values g_t, it is illustrative to reason with their characteristic timescales $1/(1-g_t)$ instead, whence both UGI and refine gate have clean interpretations. First, UGI is equivalent to initializing the decay period from a particular heavy-tailed distribution, in contrast to the standard initialization with any fixed bias b which has a fixed decay period $(1-\sigma(b))^{-1}$.

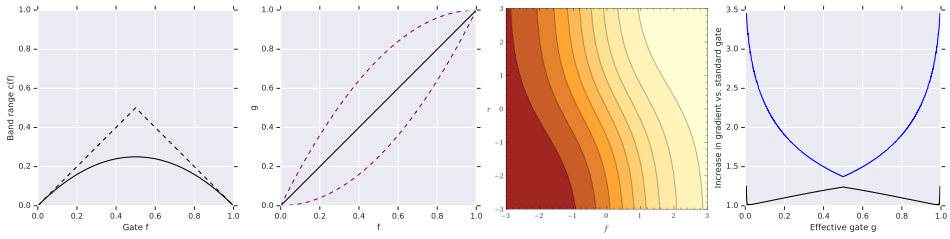
Proposition 2. *UGI is equivalent to to sampling the decay period $D=1/(1-f_t)$ from a distribution with density proportional to $\mathbb{P}(D=x)\propto\frac{d}{dx}(1-1/x)=x^{-2}$, i.e. a Pareto$(\alpha=2)$ distribution.*

On the other hand, for any forget gate activation f_t with timescale $D=1/(1-f_t)$, the refine gate fine-tunes it within $1/(1-f_t^2)=1/(1-f_t)(1+f_t)$ and $1/(1-f_t)^2$.

Proposition 3. *Given a forget gate activation with timescale D, the refine gate creates an effective forget gate with timescale in $(D/2,\ D^2)$.*

2.4 Related Work

Many previous works have observed that a basic LSTM cannot solve simple memorization tasks (Section 3.1) and offer various approaches to address the problem of long-term temporal dependencies. In order to allow the gates to operate in saturated regimes, Gulcehre et al. (2016) proposed to use piece-wise linear functions with noise. However, this introduces an additional hyperparameter and training dynamics can still be unstable if the noise injected to the activations is large. Li et al. (2018b), on the other hand, relies on the Gumbel trick (Maddison et al., 2016; Jang et al., 2016), a technique for learning discrete variables within a neural



(a) (b) (c) (d)

Figure 2: **Refine gate** in action: (a) [Solid] A function $c(f_t)$ satisfying Prop. 1 is chosen to define a band within which the forget gate is refined. (b) The effective gate g_t as a function of the forget gate f_t, which is conventionally the identity function (solid line). The refine gate interpolates around f_t to yield an effective forget gate within the boundary curves (dashed purple), $g_t \in (f_t - c(f_t), f_t + c(f_t))$. (c) 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. (d) 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 when using a refine gate vs. without.

network, to train LSTM models with discrete gates. This sort of approach can suffer from several issues, such as gradient estimation bias and unstable training dynamics from the discrepancy between training and inference; limited expressivity due to discrete instead of continuous gates; and an additional temperature hyperparameter that needs to be tuned explicitly.

Alternatively, gates can be removed entirely if strong constraints are imposed on other parts of the model. (Li et al., 2018a) use diagonal weight matrices and require stacked RNN layers to combine information between hidden units. A long line of work has investigated the use of identity or orthogonal initializations and constraints on the recurrent weights to control multiplicative gradients unrolled through time (Le et al., 2015; Arjovsky et al., 2016; Henaff et al., 2016). Chandar et al. (2019) proposed a recurrent neural network architecture that can use non-saturating activation functions without any gating mechanism. However, although these gate-less techniques can be used to alleviate the vanishing gradient problem with RNNs, unbounded activation functions can cause less stable learning dynamics and exploding gradients.

Finally, the activation of a sigmoid gate can be adjusted by modifying the inputs. One way to produce more binary activations is by dividing the input by a temperature parameter, as mentioned above. Alternatively, 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 has key differences in the timescales captured, for example by using an explicit timescale parameter and having no negative biases. Due to its similarity to UGI, we provide a more detailed comparison in Appendix A.1. However, as mentioned in Section 2.2, techniques such as these that only modify the input to a sigmoid gate do not fully address the saturation problem due to the fact gradients can still approach zero in the saturated regime.

Additionally, an important drawback common to the approaches outlined above is the introduction of substantial hyperparameters in the form of constants, training protocol (e.g. annealing), 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 3).

The ON-LSTM introduced by Shen et al. (2018) is an LSTM variant which 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. They rely on cumax, defined as $\mathrm{cumsum} \circ \mathrm{softmax}$, which creates a monotonically increasing vector in $[0,1]$, instead of σ. If applied to a forget gate (1), this induces a soft hierarchy on the cell state such that higher-ranking units forget slower. Second, the ON-LSTM creates an additional pair of auxiliary "master gates" defined with cumax, which are combined with the original forget and input f,i to define an effective forget and input gate $\hat{f}$,$\hat{i}$. Despite introducing these important novelties, we believe that the ON-LSTM has a few issues, including speed/stability issues with the cumax function, suboptimal magnitudes of the effective forget/input gates stemming from the master gates, and additional parameters or hyperparameters. We provide a detailed analysis and discussion of the ON-LSTM in Appendix A.2.

2.5 Gate ablations

Based on these insights, we propose several other gating variants to isolate the effects of different gating components. We primarily evaluate on the two main axes of variation mentioned previously, namely, activation+initialization combinations (constant bias σ, chrono, UGI) and auxilliary gates (master, refine). We

describe a few ablations in particular. Precise definitions for each of these are given in Appendix A.2. **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 auxilliary master gates and define f,i (1),(2) using the cumax activation function. **UM-: UGI master gates**. This variant of the ON-LSTM ablates the cumax operation on the master gates, replacing it with a sigmoid activation and UGI. **OR-: Refine instead of master**. A final variant of the ON- ablates the master gate, replacing it with the refine gate. In this formulation, as with our 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 summarizes the gating modifications we consider and their naming conventions. Note that we also denote the ON-LSTM method as "OM-LSTM" (M for master) for mnemonic ease. Finally, we remark that all methods here are controlled with the same number of parameters as the standard LSTM, aside from the OM-LSTM and UM-LSTM which use an additional $\frac{1}{2C}$-fraction parameters where C is the downsize factor (Appendix A.2).

Table 1: Summary of gating mechanisms considered in this work. (top) Existing approaches. (middle) Our proposed method. (bottom) Ablations of our method with different components.

Name	Gate Mechanism
LSTM	Standard LSTM with forget gate bias initialized to 1.0
C-LSTM	Chrono gate initialization
OM-LSTM	Ordered Neuron LSTM, uses master gates with cumax activation
UR-LSTM	Uniform forget gate initialization (tied input gate) and refine gate
U-LSTM	Uniform gate initialization
O-LSTM	cumax activation on forget/input gates
UM-LSTM	Master gates with uniform gate initialization
OR-LSTM	cumax activation on forget gate (tied input gate) and refine gate

3 Experiments

We evaluate our proposed gating mechanisms on synthetic memory tasks, pixel-by-pixel image classification, language modeling, and reinforcement learning. We provide further results on program execution task in the Appendix D.2. We first perform full ablations of the gating variants described in Section 2.5 on the simpler synthetic and sequential image classification tasks. Using conclusions drawn from these about the effects of various gating components, we evaluate the best candidates on the other tasks.

All of these benchmarks have prior work with strong recurrent core baselines, from which we used the same models and protocol, changing only the gating mechanism. We make a note of any important experimental details for each task, and full protocols and details are given in Appendix C. Generally, all models used are single layer LSTMs trained by Adam with default parameters. Potential hyperparameter sweeps were done only over hidden size or learning rate – other hyperparameters are taken directly from previous baselines for each task except when noted. When chrono-initialization is compared, we chose to set T_{max} proportional to the hidden size. This heuristic uses the intuition that if dependencies of length T are present then so should dependencies of all lengths $\leq T$ for reasonable tasks, and the amount of information that can be remembered can be proportional to the number of hidden units.

3.1 Synthetic 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. In the Copy task, a sequence of $N+2$ 0 digits is generated 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. Models are trained using the cross-entropy loss (Appendix C.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 the first half of the sequence and i_1 in the second half, 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.

Figure 3 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-**LSTM. 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 or the refine gate. The U-LSTM is able to make progress, but converges slower as it suffers from gate saturation without the refine gate. The vanilla LSTM makes no progress. The OM-LSTM

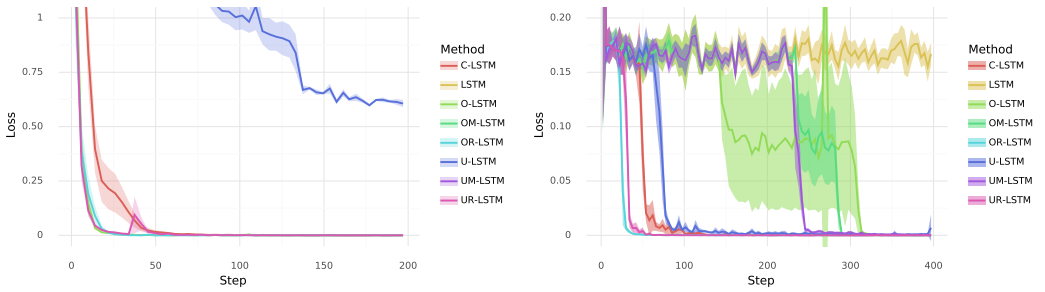


Figure 3: (Left) Copy task length 500 (Right) Adding task length 2000. Every method besides the LSTM solves the Adding task, with Refine gates faster than Master and Ordered faster than chrono or uniform gate initialization. The only methods capable of solving copy are OR-,UR-,O-, and C-LSTM models.

and UM-LSTM also get stuck at the baseline loss, despite the OM-LSTM's cumax activation, which we hypothesize is due to the suboptimal magnitudes of the gates at initialization (Appendix A.2). On the Adding task, every method besides the basic LSTM is able to eventually solve it. The refine gate variants OR- and UR-LSTM are fastest, followed by the initialization variants C- and U-LSTM, followed by the cumax-gated O-LSTM, and finally the master gate variants OM-, UM-LSTM. The refine gate variants are also significantly more stable than the others.

Figure 4 shows the distributions of forget gate activations of sigmoid-activation methods, before and after training on the Copy task. Note that the C-LSTM and vanilla LSTM stay close to initialization (near 1 for C-LSTM, and $\sigma(1.0) \approx 0.73$ for vanilla). Thus the C-LSTM solves the task as it has artificially high gate values by construction while the LSTM cannot learn any high-valued gates. On the other hand, the U-LSTM makes slow progress at pushing gate values toward 1. Finally, the UR-LSTM learns a bimodal distribution of forget gate activations, tending toward either 1 or 0, with a large fraction of the neurons able to get close to perfect remembering as is required by the task.

The C-LSTM has extreme gate initializations by construction, which is well suited for this task. This raises the question: what happens if the initialization distribution does not match the task at hand; could the gates learn back to a more moderate regime? We point out that such a phenomenon could occur non-pathologically on more complex setups, such as a scenario where a model trains to remember on a Copy-like task and then needs to "unlearn" as part of a meta-learning or continual learning setup. In Appendix D.1, we consider such a synthetic scenario and experimentally show that the addition of a refine gate helps models train much faster while in a saturated regime with extremal activations. We also point to the poor performance of C- outside of synthetic memory tasks when using our high hyperparameter-free initialization as more evidence that it is very difficult for standard gates to unlearn undesired saturated behavior.

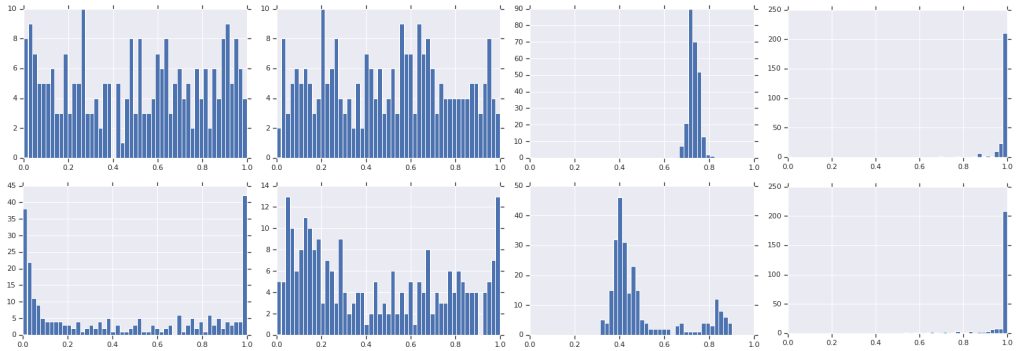
3.2 Pixel-by-pixel image classification

These tasks involve feeding a recurrent model the pixels of an image in a scanline order before producing a classification label. We test on the sequential MNIST (sMNIST), permuted MNIST (pMNIST) (Le et al., 2015), and sequential CIFAR-10 (sCIFAR) tasks. For this task, single-layer LSTMs with 512 hidden units are used as the base model. Each LSTM method was ran with a learning rate sweep with 3 seeds each. The best validation score found over any run is reported in the first two columns of Table 2.[1] We find in general that all methods are able to improve over the vanilla LSTM. However, the differences become even more pronounced when stability is considered. Although Table 2 reports the best validation accuracies found on any run, we found that most methods were quite unstable. Asterisks are marked next to a score denoting how many of the 3 seeds diverged, for the learning rate that score was found at.

Conversely, Figure 5 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.

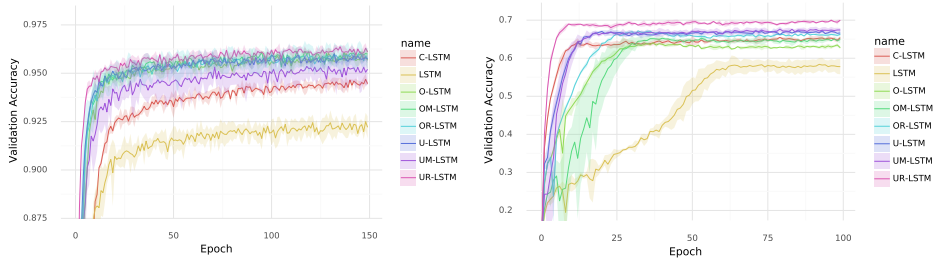
As our methods comprise simple changes to the gates (1)-(2) only, they combine effectively with other techniques developed for recurrent models. Table 2 also reports scores when the same gating mechanisms are applied to the GRU model instead of the LSTM, where similar trends hold across the gating variants. In particular, UR-GRU is the only method that is able to stably attain good performance. As another

[1] sMNIST is not included here as it is too easy, making it difficult to draw conclusions.



(a) UR-LSTM (b) U-LSTM (c) LSTM (d) C-LSTM

Figure 4: Histograms of f_t activations (avg over time and batch) before and after training on Copy (y-axis independently scaled). (Top/Bottom) Activations before/after training. C-LSTM initializes with nearly pure forget gates which barely change during training. Standard LSTM initialization cannot learn large enough f_t and makes no progress on the task. U-LSTM makes progress by encouraging a range of forget gate values, but this distribution does not change significantly during training due to saturation. UR-LSTM starts with the same distribution, but is able to learn extremal gate values. Complementary to here when learning large activations is necessary, Appendix D.1 shows a reverse task where the UR-LSTM is able to un-learn from a saturated regime.



(a) Permuted MNIST (b) Sequential CIFAR-10

Figure 5: Learning curves with deviations for different gating methods on pixel image classification, at the best stable learning rate. Error regions show the 95% confidence interval for the mean reward across three seeds.

example, applying regularization techniques designed for vanilla LSTMs to the alternative gates improve their performance even more. Table 3 compares the test accuracy of the UR gating mechanism at 2048 hidden units against other models. The addition of a generic regularization technique—we chose Zoneout (Krueger et al., 2016) with default hyperparameters ($z_c = 0.5$, $z_h = 0.05$)—continued improving the UR-LSTM, outperforming even non-recurrent models on sequential MNIST and CIFAR-10.

Table 2: Validation accuracies on pixel image classification. Asterisks denote divergent runs at the learning rate the best validation score was found at.

	-	C-	U-	O-	OM-	OR-	U-	UR-
pMNIST	95.53^{*}	95.11^{**}	96.26	96.24	96.37^{*}	96.67^{*}	96.21^{*}	96.59
sCIFAR	64.06^{**}	67.11^{***}	68.24^{*}	67.66^{*}	67.58^{*}	70.90^{*}	68.33	70.39
sCIFAR (GRU)	71.72^{**}	65.44	71.58^{*}	33.09^{***}	71.42^{**}	11.81^{***}	69.59^{**}	72.10

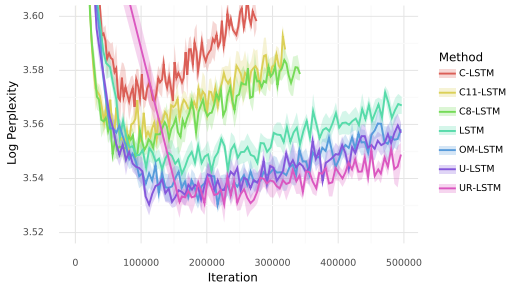
From Sections 3.1 and 3.2, we draw a few conclusions about the comparative performance of different gate modifications. Both C- and U- initialization strategies by themselves broadly improve on the LSTM when long-term dependencies are present, with chrono initialization particularly good at memory tasks. The ordered gate O- alone is better than OM- on memorization tasks, but slightly worse on images, and both are noticeably worse than OR-. We find that the ordered methods O-, OM-, OR- using cumax activation have good performance and convergence speed but are often unstable, and additionally have speed drawbacks,

Table 3: Test acc. on pixel-by-pixel image classification benchmarks. Top: LSTM baselines and variants. Middle: Non-recurrent sequence models with global receptive field. Bottom: Our methods.

Model	sMNIST	pMNIST	sCIFAR
LSTM (ours)	98.9	93.00	60.68
Dilated GRU (Chang et al., 2017)	99.0	94.6	-
IndRNN (Li et al., 2018a)	99.0	96.0	-
r-LSTM (2-Layer with Auxiliary Loss) (Trinh et al., 2018)	98.4	95.2	72.2
Transformer (Trinh et al., 2018)	98.9	97.9	62.2
Temporal convolution network (Bai et al., 2018a)	99.0	97.2	
TrellisNet (Bai et al., 2018b)	99.20	**98.13**	73.42
UR-LSTM	**99.28**	96.96	71.00
UR-LSTM + Zoneout (Krueger et al., 2016)	99.21	97.58	74.34
GRU + Zoneout	99.27	96.51	**74.4**

Model	Valid.	Test
LSTM	34.3	35.8
C-LSTM	35.0	36.4
C-LSTM (T=8)	34.3	36.1
C-LSTM (T=11)	34.6	35.8
OM-LSTM	34.0	34.7
U-LSTM	33.8	34.9
UR-LSTM	**33.6**	**34.6**

(a) Perplexities on the WikiText-103 dataset.



(b) Validation learning curves, illustrating training speed and generalization (i.e. overfitting) behavior.

preventing us from using them in later tasks. U- by itself is better than any O- method on images and Adding, but worse on Copy. Overall, we confirm that the refine gate has better convergence and stability than the master gate, and UGI is generally better than cumax aside from hard memorization tasks.

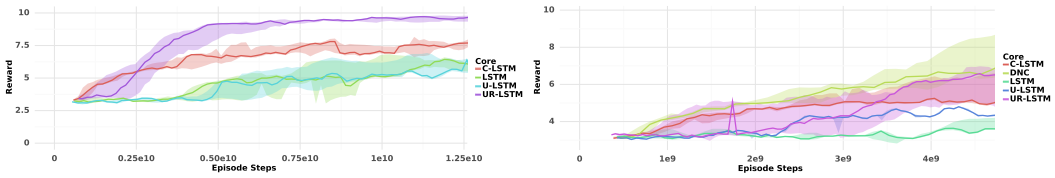
3.3 Language Modeling

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. In particular, we evaluate our gate modifications against the exact hyperparameters of a state-of-the-art LSTM baseline (Rae et al., 2018) without additional tuning (Appendix C). We compare to other gating baselines, in particular OM-LSTM, as it was designed for this domain (Shen et al., 2018) and chrono initialization, which addresses dependencies of a particular timescale, versus 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 and meant to mimic fixed biases of about 1.0 and 2.0 respectively (Tallec & Ollivier, 2018).

Table 6a shows Validation and Test set perplexities for various models. We find that the OM-LSTM, U-LSTM, and UR-LSTM all robustly improve over the standard LSTM with no additional tuning. In addition, although the OM-LSTM was designed to capture the hierarchical nature of language using the cumax activation, it does not perform better than the U-LSTM and UR-LSTM. The chrono initialization with our default initialization strategy is far too large. While manually tweaking the T_{max} hyperparameter improves its performance, it is still far from the methods that have uniform gate initialization. We attribute this to the nature of language having dependencies on multiple widely-varying timescales, and that uniform gate initialization is enough to capture these without resorting to strictly enforced hierarchies such as in OM-LSTM.

3.4 Reinforcement Learning

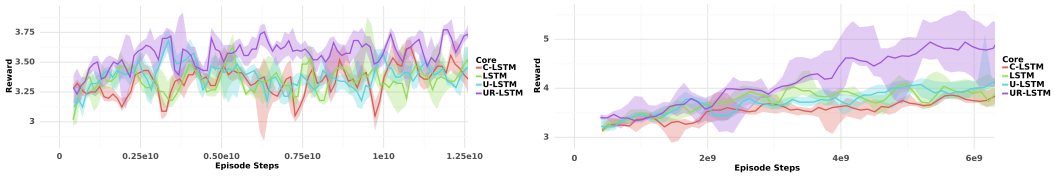
The memory is a core to many partially observable reinforcement learning tasks since the agent can observe only part of the environment once at a time. However, designing memory architectures for reinforcement learning problems have been a challenging task (Oh et al., 2016; Wayne et al., 2018). We investigated the performance of different gating mechanisms and initializations in reinforcement learning tasks to evaluated on Passive and Active matching tasks for the discount factor of $\gamma = 0.96$. In Figure 7, we showed the results of different models on the Passive matching task where UR-LSTM outperforms other models. In Figure 11, we showed the results of our models on the Active Matching tasks.



(a) Passive Match without Distractor Rewards

(b) Passive Match with Distractor Rewards

Figure 7: We have borrowed the passive matching tasks from the (Hung et al., 2018) and run an A3C agent (Mnih et al., 2016) with the LSTM, C-LSTM, U-LSTM and UR-LSTM as recurrent policy cores. We have noticed that UR-LSTM substantially outperformed other cores on the Passive Match without Distractor Rewards task and matched the performance of the DNC on the Passive Match with Distractor Rewards task. We show the mean and the confidence intervals for 5 seeds.



(a) Active Match with Distractors: DNC model

(b) Active Match with Distractors: RMA model

Figure 8: The UR gates improve other models from (Hung et al., 2018) on the Active Match task.

Hung et al. (2018) evaluated agents with different types of recurrent cores: the basic LSTM, the DNC (an LSTM with memory), and the RMA. On the full Active Match task, only the RMA agent was able to even learn better-than-chance behavior. They also considered ablations of the task with distractors removed. We found that the UR- gates substantially improved the performance of the basic LSTM core on both Passive Match and Active Match tasks, with or without distractor rewards. On the difficult Active Match task, it was the only method to achieve better than random behavior.

Although the DNC and RMA are both explicit memory architectures designed for long-term credit assignment, we considered whether their performance could also be improved with our gate improvements. Figure 8 shows the performance of the DNC and RMA core with the gate modifications. We observe that the UR-DNC is the only DNC core to achieve better than chance accuracy. In Appendix D.3, we additionally show the performance of the UR-LSTM on Active Match which nearly matches the basic RMA (8b, green). Finally, the UR-RMA is noticeably better than the other RMA cores.

4 Discussion

In this work we introduce, analyze, and evaluate several modifications to the ubiquitous gating mechanism that appears in recurrent neural networks. We describe theoretically-justified 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 method 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. We point out that the performance on these tasks can perhaps be further improved with dedicated tuning of the new methods, and also that many combinations were left unexplored. Finally, we emphasize that these improvements are completely 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

Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pp. 1120–1128, 2016.

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

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Trellis networks for sequence modeling. *arXiv preprint arXiv:1810.06682*, 2018b.

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

Sarath Chandar, Chinnadhurai Sankar, Eugene Vorontsov, Samira Ebrahimi Kahou, and Yoshua Bengio. Towards non-saturating recurrent units for modelling long-term dependencies. *arXiv preprint arXiv:1902.06704*, 2019.

Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 77–87, 2017.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. 1999.

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

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. Noisy activation functions. In *International conference on machine learning*, pp. 3059–3068, 2016.

Caglar Gulcehre, Sarath Chandar, and Yoshua Bengio. Memory augmented neural networks with wormhole connections. *arXiv preprint arXiv:1701.08718*, 2017.

Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. *arXiv preprint arXiv:1602.06662*, 2016.

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

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

Chia-Chun Hung, Timothy Lillicrap, Josh Abramson, Yan Wu, Mehdi Mirza, Federico Carnevale, Arun Ahuja, and Greg Wayne. Optimizing agent behavior over long time scales by transporting value. *arXiv preprint arXiv:1810.06721*, 2018.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

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

Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Chris Pal. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016.

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.

Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. 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.

Zhuohan Li, Di He, Fei Tian, Wei Chen, Tao Qin, Liwei Wang, and Tie-Yan Liu. Towards binary-valued gates for robust lstm training. *arXiv preprint arXiv:1806.02988*, 2018b.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128*, 2016.

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.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019.

Jack W Rae, Chris Dyer, Peter Dayan, and Timothy P Lillicrap. Fast parametric learning with activation memorization. *arXiv preprint arXiv:1803.10049*, 2018.

Adam Santoro, Ryan Faulkner, David Raposo, Jack Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy Lillicrap. Relational recurrent neural networks. In *Advances in Neural Information Processing Systems*, pp. 7299–7310, 2018.

Yikang Shen, Shawn Tan, Alessandro Sordoni, and Aaron Courville. Ordered neurons: Integrating tree structures into recurrent neural networks. *arXiv preprint arXiv:1810.09536*, 2018.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

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

Trieu H Trinh, Andrew M Dai, Minh-Thang Luong, and Quoc V Le. Learning longer-term dependencies in rnns with auxiliary losses. *arXiv preprint arXiv:1803.00144*, 2018.

Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, Jack Rae, Piotr Mirowski, Joel Z Leibo, Adam Santoro, et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *arXiv preprint arXiv:1410.3916*, 2014.

Wojciech Zaremba and Ilya Sutskever. Learning to execute. *arXiv preprint arXiv:1410.4615*, 2014.

A Further discussion on related methods

A.1 Chrono Initialization

We find that chrono initialization

$$b_f \sim \log(\mathcal{U}([1, T_{max} - 1])) \tag{13}$$

$$b_i = -b_f. \tag{14}$$

has two main drawbacks. First, it is sensitive to the hyperparameter T_{max} which represents the maximum potential length of dependencies. For example, Tallec & Ollivier (2018) set this parameter according to a different protocol for every task, with values ranging from 8 to 2000. Our experiments used a hyperparameter-free method to initialize T_{max} (Section 3), and we found that chrono initialization generally severely

over-emphasizes long-term dependencies if T_{max} is not carefully controlled. A different workaround suggested by Tallec & Ollivier (2018) is to sample from $\mathbb{P}(T=k) \propto \frac{1}{k\log^2(k+1)}$ and setting $b_f = \log(T)$. Note that such an initialization would be almost equivalent to sampling the decay period from the distribution with density $\mathbb{P}(x) \propto (x\log^2 x)^{-1}$ (since the decay period is $(1-f)^{-1} = 1+\exp(b_f)$). This parameter-free initialization is thus similar in spirit to the uniform gate initialization (Proposition 2), but from a much heavier-tailed distribution that emphasizes very long-term dependencies.

Furthermore, under this interpretation, it is plausible to define a family of Pareto-like distributions from which to draw the decay periods from, with this distribution treated as a hyperparameter. However, with no additional prior information on the task, we believe the uniform gate initialization to be the best candidate, as it 1. is a simple distribution with easy implementation, 2. has characteristic timescale distributed as an intermediate balance between the heavy-tailed chrono initialization and sharply decaying standard initialization, and 3. is similar to the ON-LSTM's cumax activation, in particular matching the initialization distribution of the cumax activation.

Table 4 summarizes the decay period distributions at initialization using different activations and initialization strategies.

Table 4: Distribution of the decay period $D = (1-f)^{-1}$ using different initialization strategies.

Initialization method	Timescale distribution
Constant bias $b_f = b$	$\mathbb{P}(D=x) \propto \mathbf{1}\{x = 1+e^b\}$
Chrono initialization (known timescale T_{max})	$\mathbb{P}(D=x) \propto \mathbf{1}\{x \in [2, T_{max}]\}$
Chrono initialization (unknown timescale)	$\mathbb{P}(D=x) \propto \frac{1}{x\log^2 x}$
cumax activation	$\mathbb{P}(D=x) \propto \frac{1}{x^2}$
Uniform gate initialization	$\mathbb{P}(D=x) \propto \frac{1}{x^2}$

A.2 ON-LSTM

In this section we elaborate on the connection between the master gates of Shen et al. (2018) and our refine gates.

First, we formally define the full ON-LSTM. The master gates are a cumax-activation gate

$$\tilde{f}_t = \mathrm{cumax}(W_{\tilde{f}} x_t + U_{\tilde{f}} h_{t-1} + b_{\tilde{f}}) \tag{15}$$

$$\tilde{i}_t = 1 - \mathrm{cumax}(W_{\tilde{i}} x_t + U_{\tilde{i}} h_{t-1} + b_{\tilde{i}}), \tag{16}$$

which modify an independent pair of forget and input gates f_t, i_t to create an effective forget/input gate $\hat{f}_t, \hat{i}_t$ which are used to update the cell state.

$$\omega_t = \tilde{f}_t \circ \tilde{i}_t \tag{17}$$

$$\hat{f}_t = f_t \circ \omega_t + (\tilde{f}_t - \omega_t) \tag{18}$$

$$\hat{i}_t = i_t \circ \omega_t + (\tilde{i}_t - \omega_t) \tag{19}$$

$$\tag{20}$$

We describe a few potential drawbacks of the ON-LSTM. First, despite having the same parameters and asymptotic efficiency as standard sigmoid gates, cumax gates turn out to be significantly slower and less stable. Second, the addition of the master gates creates additional parameters compared to the basic LSTM. Shen et al. (2018) alleviated both of these problems by defining a "downsize" operation, whereby neurons are grouped in chunks of size C, each of which share the same master gate values. However, this also creates an additional hyperparameter.

Finally, the magnitudes of the gates are suboptimally normalized. Conventional wisdom suggests that the effective forget and input gates should sum to about 1, which induces an elegant differential equation interpretation for recurrent models (Tallec & Ollivier, 2018).[2] However, the gates of the ON-LSTM are too small – summing to about $5/6$ are initialization. We show this in Appendix A.2 and provide further discussion.

[2]Indeed, some variants of the LSTM such as the GRU enforce this property via weight-tying, as do our own refine gate methods.

The speed and stability issues can be fixed by just using the sigmoid non-linearity instead of cumax. To recover the most important properties of the cumax—activations at multiple timescales—the equivalent sigmoid gate can be initialized so as to match the distribution of cumax gates at initialization. This is just uniform gate initialization (6).

Finally, the gate normalization can be fixed by re-scaling equations (18)-(19). It turns out that this is exactly equivalent to the refine gate mechanism (Appendix A.2).[3]

Thus, the uniform gate initialization + refine gate LSTM can also be viewed as an improved version of the ON-LSTM gating mechanism that fixes problems with speed and stability, tuning, and initialization.

A nice interpretation of gated recurrent models shows that they are a discretization of a continuous differential equation. This leads to the leaky RNN model $h_{t+1}=(1-\alpha)h_t+\alpha u_t$, where u_t is the update to the model such as $\tanh(W_x x_t+W_h h_t+b)$. Learning α as a function of the current time step leads to the simplest gated recurrent model (this is extremely close to the GRU but without the reset gate. It is also equivalent to a Recurrent Highway Network of depth 1). Tallec & Ollivier (2018) show that this exactly corresponds to the discretization of a differential equation that is invariant to *time warpings* and time rescalings. In the context of the LSTM, this interpretation requires the values of the forget and input gates to be tied so that $f_t+i_t=1$. This weight-tying is sometimes used, for example in the most popular LSTM variant, the GRU (Cho et al., 2014). In a large-scale LSTM architecture search, it was found that removing the input gate was not significantly detrimental (Greff et al., 2016).

However, the ON-LSTM does not satisfy this common wisdom that the input and forget gates should sum to close to 1.

Proposition 4. *At initialization, the expected value of the average effective forget gate activation $\hat{f}_t$ is $5/6$.*

Let us consider the sum of the effective forget and input gates at initialization. Adding equations (18) and (19) yields

$$\begin{aligned}\hat{f}_t+\hat{i}_t&=(f_t+i_t)\circ\omega_t+(\tilde{f}_t+\tilde{i}_t-2\omega_t)\\&=\tilde{f}_t+\tilde{i}_t+(f_t+i_t-2)\circ\omega_t.\end{aligned}$$

Note that the master gates (15), (16) sum 1 in expectation at initialization, as do the original forget and input gates. Looking at individual units in the ordered master gates, we have $\mathbb{E}\hat{f}^j=\frac{j}{n}$,$\mathbb{E}\hat{i}^j=1-\frac{j}{n}$. Thus the above simplifies to

$$\begin{aligned}\mathbb{E}[\hat{f}_t+\hat{i}_t]&=1-\mathbb{E}\omega_t\\\mathbb{E}[\hat{f}_t^j+\hat{i}_t^j]&=1-\frac{j}{n}(1-\frac{j}{n})\\\mathbb{E}[\mathbb{E}_{j=0..n}\hat{f}_t^j+\hat{i}_t^j]&=1-\int_0^1 x\,dx+\int_0^1 x^2\,dx\\&=\frac{5}{6}.\end{aligned}$$

Proposition 5. *Suppose the master gates $\tilde{f}_t$,$\tilde{i}_t$ are tied and the equations* (18)-(19) *defining the effective gates $\hat{f}_t$,$\hat{i}_t$ are rescaled such as to ensure $\mathbb{E}[\hat{f}_t+\hat{i}_t]=1$ at initialization. The resulting gate mechanism is exactly equivalent to that of the refine gate.*

Consider the following set of equations where the master gates are tied ($\tilde{f}_t+\tilde{i}_t=1$,$f_t+i_t=1$) and (18)-(19) are modified with an extra coefficient (rescaling in bold):

$$\tilde{i}_t=1-\tilde{f}_t \tag{21}$$
$$i_t=1-f_t \tag{22}$$
$$\omega_t=\tilde{f}_t\cdot\tilde{i}_t \tag{23}$$
$$\hat{f}_t=\mathbf{2}\cdot f_t\cdot\omega_t+(\tilde{f}_t-\omega_t) \tag{24}$$
$$\hat{i}_t=\mathbf{2}\cdot i_t\cdot\omega_t+(\tilde{i}_t-\omega_t) \tag{25}$$

Now we have

$$\begin{aligned}\hat{f}_t+\hat{i}_t&=\tilde{f}_t+\tilde{i}_t+2(f_t+i_t-1)\cdot\omega_t\\&=1+2(f_t+i_t-1)\cdot\omega_t\end{aligned}$$

[3] In this equivalence, the role of the master and forget gates of the ON-LSTM are played by our forget and refine gate respectively.

which has the correct scaling, i.e. $\mathbb{E}[\hat{f}_t+\hat{i}_t]=1$ at initialization assuming that $\mathbb{E}[f_t+i_t]=1$ at initialization.

But (24) can be rewritten as follows:

$$\begin{aligned}\hat{f}&=2\cdot f\cdot\omega+(\tilde{f}-\omega)\\&=2\cdot f\cdot\tilde{f}\cdot(1-\tilde{f})+(\tilde{f}-\tilde{f}\cdot(1-\tilde{f}))\\&=2f\cdot\tilde{f}-2f\cdot\tilde{f}^2+\tilde{f}^2\\&=f\cdot2\tilde{f}-f\cdot\tilde{f}^2-f\cdot\tilde{f}^2+\tilde{f}^2\\&=f\cdot(1-(1-\tilde{f}))^2+(1-f)\cdot\tilde{f}^2.\end{aligned}$$

This is equivalent to the refine gate, where the master gate plays the role of the forget gate and the forget gate plays the role of the refine gate. Note that in this case, the effective input gate $\hat{i}_t$ is also defined through a refine gate mechanism, where i_t is refined by $\tilde{i}_t=1-\tilde{f}_t$.

A.3 Gate ablation details

O- A natural simplification of the main idea of ON-LSTM, while keeping the hierarchical bias on the forget activations, is to simply drop the auxilliary gates and define the original forget and input gates (1),(2) using the cumax activation function.

$$f_t=\mathrm{cumax}(W_{xf}x_t+W_{hf}h_{t-1}+b_f) \tag{26}$$

$$i_t=1-\mathrm{cumax}(W_{xi}x_t+W_{hi}h_{t-1}+b_i). \tag{27}$$

We note that one difficulty with this in practice is the reliance on the expensive cumax, and hypothesize that this is perhaps the ON-LSTM's original motivation for the second set of gates combined with downsizing.

UM- This variant of the ON-LSTM ablates the cumax operation on the master gates, replacing it with a sigmoid activation initialized with UGI. Equations (15), (16) are replaced with

$$u=\mathcal{U}(0,1) \tag{28}$$

$$b=\sigma^{-1}(u) \tag{29}$$

$$\tilde{f}_t=\sigma(W_{x\tilde{f}}x_t+W_{h\tilde{f}}h_{t-1}+b) \tag{30}$$

$$\tilde{i}_t=\sigma(W_{x\tilde{i}}x_t+W_{h\tilde{i}}h_{t-1}-b) \tag{31}$$

OR- A final variant of the ON- ablates the master gate, replacing it with the refine gate. In this formulation, as with our 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.

B Analysis Details

The gradient analysis in Figure 2 was constructed as follows. Let f,r,g be the forget, refine, and effective gates

$$g=2rf+(1-2r)f^2.$$

Then

$$\begin{aligned}\nabla_x g&=2rf(1-f)+(1-2r)(2f)(f(1-f))=2f(1-f)[r+(1-2r)f]\\\nabla_y g&=2fr(1-r)+(-2f^2)r(1-r)=2fr(1-r)(1-f)\\\|\nabla g\|^2&=[2f(1-f)]^2\left[(r+f-2fr)^2+r^2(1-r)^2\right].\end{aligned}$$

Substituting the relation

$$r=\frac{g-f^2}{2f(1-f)},$$

this reduces to

$$\begin{aligned}\|\nabla g\|^2&=[2f(1-f)]^2\left[\left(\frac{g-f^2+2f^2(1-f)-2f(g-f^2)}{2f(1-f)}\right)^2+\frac{(g-f^2)^2(2f(1-f)-g+f^2)^2}{(2f(1-f))^4}\right]\\&=((g-f^2)(1-2f)+2f^2(1-f))^2+(g-f^2)^2\left(1-\frac{g-f^2}{2f(1-f)}\right)^2.\end{aligned}$$

Given the constraint $f^2\leq g\leq 1-(1-f)^2$, this function can be minimized and maximized in terms of g to produce the upper and lower bounds in Figure 2d.

C EXPERIMENTAL DETAILS

C.1 SYNTHETIC TASKS

All models consisted of single layer LSTMs with 256 hidden units, trained with the Adam optimizer Kingma & Ba (2014) with learning rate 1e-3.

Our version of the Copy task is a very minor variant of other versions reported in the literature, with the main difference being that the loss is considered only over the last 10 output tokens which need to be memorized. This normalizes the loss so that losses approaching 0 indicate true progress. In contrast, this task is usually defined with the model being required to output a dummy token at the first $N+10$ steps, meaning it can be hard to evaluate performance since low average losses simply indicate that the model learns to output the dummy token.

Figure 3 Error regions show the 95% confidence interval for the mean reward across three seeds.

Figure 4 Each histogram represents the distribution of the `hidden size` number of forget gate units. The values are created by averaging units over time and samples, i.e., reducing a minibatch of forget gate activations of shape `(batch size, sequence length, hidden size)` over the first two diensions.

C.2 IMAGE CLASSIFICATION

All models used a single hidden layer recurrent network (LSTM or GRU). Inputs x to the model were given in batches as a sequence of shape `(sequence length, num channels)`, (e.g. $(1024,3)$ for CIFAR-10), by flattening the input image left-to-right, top-to-bottom. The outputs of the model of shape `(sequence length, hidden size)` were processed independently with a single ReLU hidden layer of size 256 before the final fully-connected layer outputting softmax logits.

Table 2 LSTM models (first two columns) used hidden state size 512. Learning rate swept in $\{2e{-}4,5e{-}4,1e{-}3,2e{-}3\}$ with three seeds each. Batch size 50. MNIST trained for 150 epochs, CIFAR-10 used 100 epochs over the training set.

Table 2 reports the highest validation score found. Based on LSTM results, the GRU model had hidden size 1024 and swept over learning rates $\{5e{-}4,1e{-}3\}$. It is likely that a wider range would have yielded more consistent results.

Figure 5 shows the average validation accuracy (with confidence intervals) over the seeds, for the best-performing stable learning rate.

Table 3 The uniform gate initialization with refine gate LSTM used 1024 hidden units for the sequential and permuted MNIST task, and 2048 hidden units for the sequential CIFAR task.

The vanilla LSTM baseline used 512 hidden units for MNIST and 1024 for CIFAR. Larger hidden sizes were found to be unstable.

Zoneout parameters were fixed to reasonable default settings based on Krueger et al. (2016), which are $z_c{=}0.5, z_h{=}0.05$ for LSTM and $z{=}0.1$ for GRU. When zoneout was used, standard Dropout (Srivastava et al., 2014) with probability 0.5 was also applied to the output classification hidden layer.

C.3 LANGUAGE MODELING

Hyperparameters are taken from Rae et al. (2018) tuned for the vanilla LSTM, which consist of (chosen parameter bolded out of sweep): $\{1,2\}$ LSTM layer, $\{0.0,0.1,0.2,\mathbf{0.3}\}$ embedding dropout, {yes,**no**} layer norm, and {**shared**,not shared} input/output embedding parameters. Our only divergence is using a hidden size of 3072 instead of 2048, which we found improved the performance of the vanilla LSTM. Training was performed with Adam at learning rate 1e-3, gradients clipped to 0.1, sequence length 128, and batch size 128 on TPU. The LSTM state was reset between article boundaries.

C.4 PROGRAM EVALUATION

Protocol was taken from Santoro et al. (2018) with minor changes to the hyperparameter search. All models were trained with the Adam optimizer, the *Mix* curriculum strategy from Zaremba & Sutskever (2014), and batch size 128.

RMC: The RMC models used a fixed memory slot size of 512 and swept over $\{2,4\}$ memories and $\{2,4\}$ attention heads for a total memory size of 1024 or 2048. They were trained for 2e5 iterations.

LSTM: Instead of two-layer LSTMs with sweeps over skip connections and output concatenation, single-layer LSTMs of size 1024 or 2048 were used. Learning rate was swept in {5e-4, 1e-3}, and models were trained for 5e5 iterations. Note that training was still faster than the RMC models despite the greater number of iterations.

C.5 ADDITIONAL DETAILS

Implementation Details The inverse sigmoid function (6) can be unstable if the input is too close to $\{0,1\}$. Uniform gate initialization was instead implemented by sampling from the distribution $\mathcal{U}[1/d,1-1/d]$ instead of $\mathcal{U}[0,1]$, where d is the hidden size, to avoid any potential numerical edge cases. This choice is justified by the fact that with perfect uniform sampling, the expected smallest and largest samples would be $1/(d+1)$ and $1-1/(d+1)$.

For distributional initialization strategies, a trainable bias vector was sampled independently from the chosen distribution (i.e. equation (14) or (6)) and added/subtracted to the forget and input gate ((1)-(2)) before the non-linearity. Additionally, each linear model such as $W_{xf}x_t+W_{hf}h_{t-1}$ had its own trainable bias vector, effectively doubling the learning rate on the pre-activation bias terms on the forget and input gates. This was an artifact of implementation and not intended to affect performance.

Permuted image classification In an effort to standardize the permutation used in the Permuted MNIST benchmark, we use a particular deterministic permutation rather than a random one. After flattening the input image into a one-dimensional sequence, we apply the *bit reversal* permutation. This permutation sends the index i to the index j such that j's binary representation is the reverse of i's binary representation. The intuition is that if two indices i,i' are close, they must differ in their lower-order bits. Then the bit-reversed indices will be far apart. Therefore the bit-reversal permutation destroys spatial and temporal locality, which is desirable for these sequence classification tasks meant to test long-range dependencies rather than local structure.

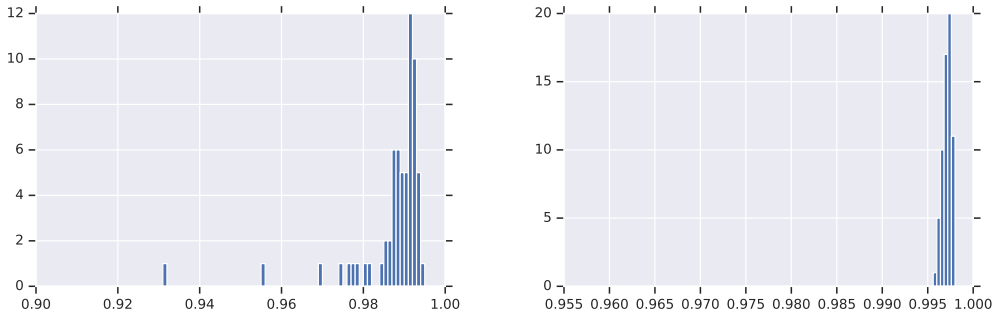
```
def bitreversal_po2(n):
    m = int(math.log(n) / math.log(2))
    perm = np.arange(n).reshape(n, 1)
    for i in range(m):
        n1 = perm.shape[0] // 2
        perm = np.hstack((perm[:n1], perm[n1:]))
    return perm.squeeze(0)
def bitreversal_permutation(n):
  m = int(math.ceil(math.log(n) / math.log(2)))
  N = 1 << m
  perm = bitreversal_po2(N)
  return np.extract(perm < n, perm)
```

D ADDITIONAL EXPERIMENTS

D.1 SYNTHETIC FORGETTING

One of the original motivations of

Figure 4 illustrates how the refine gate significantly helps gates learn extreme activations, which is normally difficult because of gate saturation. Here we empirically demonstrate the opposite phenomenon: that if gates are too extreme and need to be regressed (perhaps due to needing to "unlearn" previous weights after distributional input shift, or simply due to initialization).



(a) UR-LSTM (b) LSTM

Figure 9: Distribution of forget gate activations after extremal initialization, and training on the Adding task. The UR-LSTM is able to learn much faster in this saturated gate regime while the LSTM does not solve the task. The smallest forget unit for the UR-LSTM after training has characteristic timescale over an order of magnitude smaller than that of the LSTM.

For this experiment, we initialize the biases of the gates extremely high (effective forget activation $\approx\sigma(6)$. We then consider the Adding task (Section 3.1 of length 500, hidden size 64, learning rate 1e-4. The UR-LSTM is able to solve the task, while the LSTM is stuck after 1e4 iterations.

D.2 Program Execution

The Learning to Execute (Zaremba & Sutskever, 2014) dataset consists of algorithmic snippets from a programming language of pseudo-code. An input is a program from this language presented one character at a time, and the target output is a numeric sequence of characters representing the execution output of the program. There are three categories of tasks: Addition, Control, and Program, with distinctive types of input programs. We use the most difficult setting from Zaremba & Sutskever (2014), which uses the parameters `nesting=4, length=9`, referring to the nesting depth of control structure and base length of numeric literals, respectively. Examples of input programs are shown in previous works (Zaremba & Sutskever, 2014; Santoro et al., 2018).

We are interested in this task for several reasons. First, we are interested in comparing against the C- and OM-gate methods, because

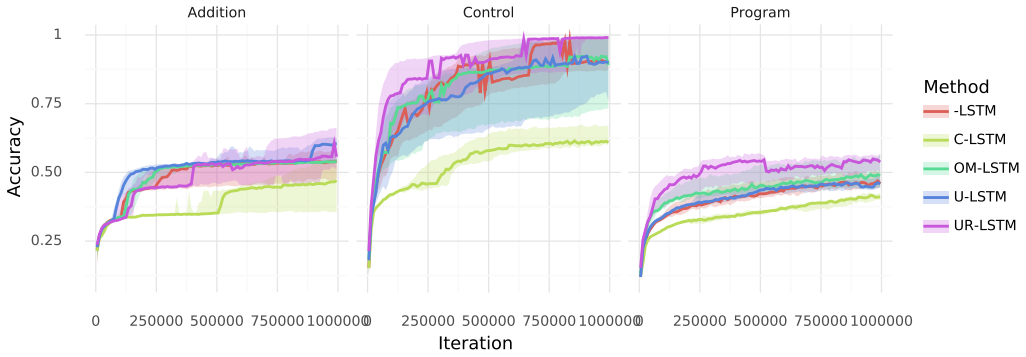
- The maximum sequence length is fairly long (several hundred tokens), meaning our T_{max} heuristic for C-gates is within the right order of magnitude of dependency lengths.
- The task has highly variable sequence lengths, wherein the standard training procedure randomly samples inputs of varying lengths (called the "Mix" curriculum in Zaremba & Sutskever (2014)). Additionally, the Control and Program tasks contain complex control flow and nested structure. They are thus a measure of a sequence model's ability to model dependencies of differing lengths, as well as hierarchical information. Thus we are interested in comparing the effects of UGI methods, as well as the full OM- gates which are designed for hierarchical structures (Shen et al., 2018).

Finally, this task has prior work using a different type of recurrent core, the Relational Memory Core (RMC), that we also use as a baseline to evaluate our gates on different models Santoro et al. (2018).

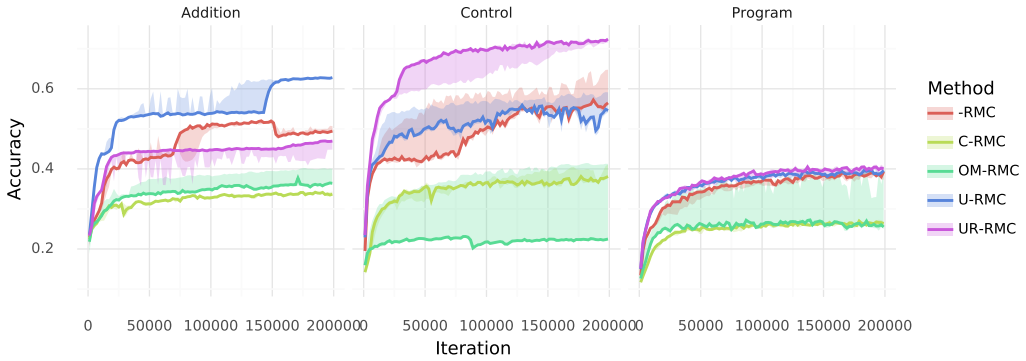
Training curves are shown in Figure 10, which plots the median accuracy with confidence intervals. We point out a few observations. First, despite having a T_{max} value on the right order of magnitude, the C- gated methods have very poor performance across the board, reaffirming the chrono initialization's high sensitivity to this hyperparameter.

Second, the U-LSTM and U-RMC are the best methods on the Addition task. Additionally, the UR-RMC vs. RMC on Addition is one of the very few tasks we have found where a generic substitution of the UR-gate does not improve on the basic gate. We have not investigated what property of this task caused these phenomena.

Aside from the U-LSTM on addition, the UR-LSTM outperforms all other LSTM cores. The UR-RMC is also the best core on both Control and Program, the tasks involving hierarchical inputs and longer dependencies. For the most part, the improved mechanisms of the UR- gates seem to transfer to this recurrent core as



(a) LSTM - Program Evaluation (nesting=4, length=9)



(b) RMC - Program Evaluation (nesting=4, length=9)

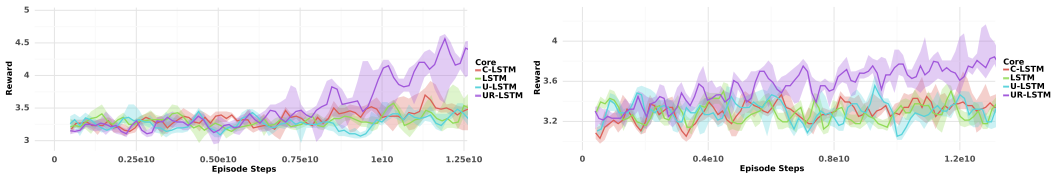
Figure 10: Program Execution evaluation accuracies.

well. We highlight that this is not true of similar gating mechanisms. In particular, the OM-LSTM, which is supposed to model hierarchies, has good performance on Control and Program as expected (although not better than the UR-LSTM). However, the OM- gates' performance plummets when transferred to the RMC core.

Interestingly, the -LSTM cores are consistently better than the -RMC versions, contrary to previous findings on easier versions of this task using similar protocol and hyperparameters Santoro et al. (2018). We did not explore different hyperparameter regimes on this more difficult setting.

D.3 Reinforcement Learning

Figures 7 and 8 evaluated our gating methods with the LSTM model on the Passive Match task, and with the DNC and RMA model on the Active Match task from Hung et al. (2018). We additionally ran the agents with gate modifications on the LSTM core on the Active Match task, with and without distractor rewards. Learning curves are shown in Figure 11. Similarly to the other results, the UR- gated core is noticeably better than the others, which are not noticeably better than random chance.



(a) Active Match without Distractor Rewards

(b) Active Match with Distractor Rewards

Figure 11: We experimented with the active matching tasks proposed in (Hung et al., 2018) and ran an A3C agent (Mnih et al., 2016) with the LSTM, C-LSTM, U-LSTM and UR-LSTM as recurrent policy cores. The UR-LSTM substantially outperformed other cores on the Active Match without Distractor Rewards. We show the mean and the confidence intervals for 5 seeds.