A. Pseudocode

We show how the gated update in a typical LSTM implementation can be easily replaced by UR- gates.

The following snippets show pseudocode for the the gated state updates for a vanilla LSTM model (top) and UR-LSTM (bottom).

```
forget_bias = 1.0 # hyperparameter
...
f, i, u, o = Linear(x, prev_hidden)
f_ = sigmoid(f + forget_bias)
i_ = sigmoid(i)
next_cell = f_ * prev_cell + i_ * tanh(u)
next_hidden = sigmoid(o) * tanh(next_cell)
```

Listing 1: LSTM

```
# Initialization
u = np.random.uniform(low=1/hidden_size,
                      high=1-1/hidden_size,
                      size=hidden_size)
forget_bias = -np.log(1/u-1)
...
# Recurrent update
f, r, u, o = Linear(x, prev_hidden)
f_ = sigmoid(f + forget_bias)
r_ = sigmoid(r - forget_bias)
g = 2*r_*f_ + (1-2*r_)*f_**2
next_cell = g * prev_cell + (1-g) * tanh(u)
next_hidden = sigmoid(o) * tanh(next_cell)
```

Listing 2: UR-LSTM

B. Further discussion on related methods

Section 4 briefly introduced chrono initialization (Tallec & Ollivier, 2018) and the ON-LSTM (Shen et al., 2018), closely related methods that modify the gating mechanism of LSTMs. We provide more detailed discussion on these in Sections B.3 and B.4 respectively. Section B.1 has a more thorough overview of related work on recurrent neural networks that address long-term dependencies or saturating gates.

B.1. Related Work

Several methods exist for addressing gate saturation or allowing more binary activations. Gulcehre et al. (2016) proposed to use piece-wise linear functions with noise in order to allow the gates to operate in saturated regimes. Li et al. (2018b) instead use the Gumbel trick (Maddison et al., 2016; Jang et al., 2016), a technique for learning discrete variables within a neural network, to train LSTM models with discrete gates. These stochastic approaches can suffer from issues such as gradient estimation bias, unstable training, and limited expressivity from discrete instead of continuous gates. Additionally they require more involved training protocols with 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 another RNN architecture using additive state updates and non-saturating activation functions instead of gates. 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.

As mentioned, a particular consequence of the inability of gates to approach extrema is that gated recurrent models struggle to capture very long dependencies. These problems have traditionally been addressed by introducing new components to the basic RNN setup. Some techniques include 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 important and commonly used due their faster inference without the need to maintain the entire sequence in memory. We emphasize that the vast majority of proposed RNN changes are completely orthogonal to the simple gate improvements in this work, and we do not focus on them.

A few other recurrent cores that use the basic gated update (1) but use more sophisticated update functions u include the GRU, Reconstructive Memory Agent (RMA; Hung et al., 2018), and Relational Memory Core (RMC; Santoro et al., 2018), which we consider in our experiments.

B.2. Effect of proposed methods on timescales

We briefly review the connection between our methods and the effective timescales that gated RNNs capture. Recall that Section 3.2 defines the *characteristic timescale* of a neuron with forget activation f_t as $1/(1-f_t)$, which would be the number of timesteps it takes to decay that neuron by a constant.

The fundamental principle of gated RNNs is that the activations of the gates affects the timescales that the model can address; for example, forget gate activations near 1.0 are necessary to capture long-term dependencies.

Thus, although our methods were defined in terms of activations g_t, it is illustrative to reason with their characteristic timescales $1/(1-g_t)$ instead, whence both UGI and refine gate also have clean interpretations.

First, UGI is equivalent to initializing the decay period from a particular heavy-tailed distribution, in contrast to standard initialization with a fixed decay period $(1-\sigma(b_f))^{-1}$.

Proposition 1. *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 between $D=1/(1-f_t^2)=1/(1-f_t)(1+f_t)$ and $1/(1-f_t)^2$.

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

B.3. Chrono Initialization

The chrono initialization

$$b_f \sim \log(\mathcal{U}([1, T_{max}-1])) \tag{14}$$
$$b_i = -b_f. \tag{15}$$

was the first to explicitly attempt to initialize the activation of gates across a distributional range. It was motivated by matching the gate activations to the desired timescales.

They also elucidate the benefits of tying the input and forget gates, leading to the simple trick (15) for approximating tying the gates at initialization, which we borrow for UGI. (We remark that perfect tied initialization can be accomplished by fully tying the linear maps $\mathcal{L}_f, \mathcal{L}_i$, but (15) is a good approximation.)

However, the main drawback of CI is that the initialization distribution is too heavily biased toward large terms. This leads to empirical consequences such as difficult tuning (due to most units starting in the saturation regime, requiring different learning rates) and high sensitivity to the hyperparameter T_{max} that 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 5), 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 $\mathbb{P}(D=x) \propto (x \log^2 x)^{-1}$ (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 1), but from a much heavier-tailed distribution that emphasizes very long-term dependencies.

These interpretations suggest that it is plausible to define a family of Pareto-like distributions from which to draw the initial 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.

In general, our experimental recommendation for CI is that it can be better than standard initialization or UGI when certain conditions are met (tasks with long dependencies and nearly fixed-length sequences as in Sections 5.1, 5.4) and/or when it can be explicitly tuned (both the hyperparameter T_{max}, as well as the learning rate to compensate for almost all units starting in saturation). Otherwise, we recommend UGI or standard initialization. We found no scenarios where it outperformed UR- gates.

B.4. ON-LSTM

In this section we elaborate on the connection between the mechanism of (Shen et al., 2018) and our methods. We define the full ON-LSTM and show how its gating mechanisms can be improved. For example, there is a remarkable connection between its master gates and our refine gates – independently of the derivation of refine gates in Section 3.4, we show how a specific way of fixing the normalization of master gates becomes equivalent to a single refine gate.

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

$$\tilde{f}_t = \mathrm{cumax}(\mathcal{L}_{\tilde{f}}(x_t, h_{t-1})) \tag{16}$$
$$\tilde{i}_t = 1 - \mathrm{cumax}(\mathcal{L}_{\tilde{i}}(x_t, h_{t-1})). \tag{17}$$

These combine with an independent pair of forget and input gates f_t, i_t, meant to control fine-grained behavior, to create an effective forget/input gate $\hat{f}_t, \hat{i}_t$ which are used to update

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)	$\mathbb{P}(D=x) \propto \frac{1}{x \log^2 x}$
Uniform gate initialization	$\mathbb{P}(D=x) \propto \frac{1}{x^2}$
cumax activation	$\mathbb{P}(D=x) \propto \frac{1}{x^2}$

the state (equation (1) or (5)).

$$\omega_t = \tilde{f}_t \circ \tilde{i}_t \tag{18}$$

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

$$\hat{i}_t = i_t \circ \omega_t + (\tilde{i}_t - \omega_t). \tag{20}$$

As mentioned in Section B.1, this model modifies the standard forget/input gates in two main ways, namely ordering the gates via the cumax activation, and supplying an auxiliary set of gates controlling fine-grained behavior. Both of these are important novelties and together allow recurrent models to better capture tree structures.

However, the UGI and refine gate can be viewed as improvements over each of these, respectively, demonstrated both theoretically (below) and empirically (Sections 5 and E.3), even on tasks involving hierarchical sequences.

Ordered gates Despite having the same parameter count and asymptotic efficiency as standard sigmoid gates, cumax gates seem noticeably slower and less stable in practice for large hidden sizes. Additionally, using auxiliary 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.

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 (equation (12)).

However, we believe that the cumax activation is still valuable in many situations if speed and instability are not issues. These include when the hidden size is small, when extremal gate activations are desired, or when ordering needs to be strictly enforced to induce explicit hierarchical structure. For example, Section (5.1) shows that they can solve hard memory tasks by themselves.

Master gates We observe that the magnitudes of master gates are suboptimally normalized. 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[5]

$$\begin{aligned} f_t &= \sigma(\mathcal{L}_f(x_t, h_{t-1})) \\ u_t &= \tanh(\mathcal{L}_u(x_t, h_{t-1})) \\ h_t &= f_t h_{t-1} + (1-f_t) u_t. \end{aligned}$$

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 often enforced, for example in the most popular LSTM variant, the GRU (Cho et al., 2014), or our UR- gates. 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 conventional wisdom that the input and forget gates should sum to close to 1.

Proposition 3. *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 (19) and (20) 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 (16), (17) sum 1 in expectation at initialization, as do the original forget and input gates. Looking at individual units in the ordered master gates, we

[5] In the literature, this is called the JANET (van der Westhuizen & Lasenby, 2018), which is also equivalent to the GRU without a reset gate (Chung et al., 2014), or a recurrent highway network with depth $L = 1$ (Zilly et al., 2017).

have $\mathbb{E}\hat{f}^{(j)}=\frac{j}{n}$,$\mathbb{E}\hat{i}^{(j)}=1-\frac{j}{n}$. Thus the above simplifies to

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

The gate normalization can be fixed by re-scaling equations (19) and (20). It turns out that tying the master gates and re-scaling is exactly equivalent to the mechanism of a refine gate. In this equivalence, the role of the master and forget gates of the ON-LSTM are played by our forget and refine gate respectively.

Proposition 4. *Suppose the master gates $\tilde{f}_t$,$\tilde{i}_t$ are tied and the equations* (19)-(20) *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 (19)-(20) are modified with an extra coefficient (rescaling in bold):

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

Now we have

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

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 (23) can be rewritten as follows:

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

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. It can be shown that in this case, the effective input gate $\hat{i}_t$ (24) is also defined through a refine gate mechanism, where $\tilde{i}_t=1-\tilde{f}_t$ is refined by i_t:

$$\hat{i}=i\cdot(1-(1-\tilde{i}))^2+(1-i)\cdot\tilde{i}^2.$$

Based on our experimental findings, in general we would recommend the refine gate in place of the master gate.

B.5. Gate ablation details

For clarity, we formally define the gate ablations considered which mix and match different gate components.

We remark that other combinations are possible, for example combining CI with either auxiliary gate type, which would lead to CR- or CM- gates. Alternatively, the master or refine gates could be defined using different activation and initialization strategies. We chose not to consider these methods due to lack of interpretation and theoretical soundness.

O- This ablation uses the cumax activation to order the forget/input gates and has no auxiliary gates.

$$f_t=\mathrm{cumax}(\mathcal{L}_f(x_t,h_{t-1})) \tag{25}$$
$$i_t=1-\mathrm{cumax}(\mathcal{L}_i(x_t,h_{t-1})). \tag{26}$$

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 (16), (17) are replaced with

$$u=\mathcal{U}(0,1) \tag{27}$$
$$b_f=\sigma^{-1}(u) \tag{28}$$
$$\tilde{f}_t=\sigma(\mathcal{L}_{\tilde{f}}(x_t,h_{t-1})+b_f) \tag{29}$$
$$\tilde{i}_t=\sigma(\mathcal{L}_{\tilde{i}}(x_t,h_{t-1})-b_f) \tag{30}$$

Equations (18)-(20) are then used to define effective gates $\hat{f}_t$,$\hat{i}_t$ which are used in the gated update (1) or (5).

OR- This ablation combines ordered main gates with an auxilliary refine gate.

$$\tilde{f}_t=\mathrm{cumax}(\mathcal{L}_{\tilde{f}}(x_t,h_{t-1})+b_f) \tag{31}$$
$$r_t=\sigma(\mathcal{L}_r(x_t,h_{t-1})+b_r) \tag{32}$$
$$g_t=r_t\cdot(1-(1-f_t)^2)+(1-r_t)\cdot f_t^2 \tag{33}$$
$$i_t=1-g_t \tag{34}$$

g_t,i_t are used as the effective forget and input gates.

C. Analysis Details

The gradient analysis in Figure 3 was constructed as follows. Let f, r, g be the forget, refine, and effective gates

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

Letting x, y be the pre-activations of the sigmoids on f and r, the gradient of g can be calculated as

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

Substituting the relation

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

this reduces to the Equation 35,

$$\begin{aligned}\|\nabla g\|^2 = &((g-f^2)(1-2f) + 2f^2(1-f))^2 \\ &+ (g-f^2)^2\left(1 - \frac{g-f^2}{2f(1-f)}\right)^2. \end{aligned} \tag{35}$$

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 3b. This was performed numerically.

D. Experimental Details

To normalize the number of parameters used for models using master gates, i.e. the OM- and UM- gating mechanisms, we used a downsize factor on the main gates (see Section B.4). This was set to $C = 16$ for the synthetic and image classification tasks, and $C = 32$ for the language modeling and program execution tasks which used larger hidden sizes.

D.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. Gradients were clipped at 1.0.

The training data consisted of randomly generated sequences for every minibatch rather than iterating through a fixed dataset. Each method ran 3 seeds, with the same training data for every method.

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.

For Figure 4, the log loss curves show the median of 3 seeds, and the error bars indicate 60% confidence.

For Figure 5, each histogram represents the distribution of forget gate values of the hidden units (of which there are 256). 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, to produce the average activation value for every unit.

D.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. All training was performed with the Adam optimizer, batch size 50, and gradients clipped at 1.0. MNIST trained for 150 epochs, CIFAR-10 used 100 epochs over the training set.

Table 5 All models (LSTM and GRU) used hidden state size 512. Learning rate swept in $\{2e-4, 5e-4, 1e-3, 2e-3\}$ with three seeds each.

Table 5 reports the highest validation score found. The GRU model swept over learning rates $\{2e-4, 5e-4\}$; all methods were unstable at higher learning rates.

Figure 6 shows the median validation accuracy with quartiles (25/75% confidence intervals) over the seeds, for the best-performing stable learning rate (i.e. the one with highest average validation score on the final epoch). This was generally $5e-4$ or $1e-3$, with refine gate variants tending to allow higher learning rates.

Table 2 The UR-LSTM and UR-GRU 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

Table 5. **Gate ablations on pixel-by-pixel image classification.** Validation accuracies on pixel image classification. Asterisks denote divergent runs at the learning rate the best validation score was found at.

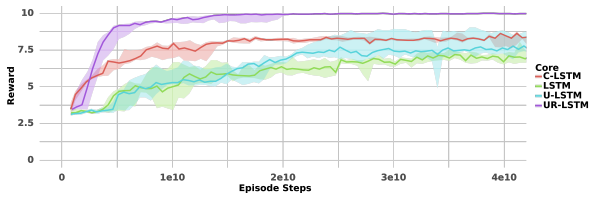
Gating Method	-	C-	O-	U-	R-	OM-	OR-	UM-	UR-
pMNIST	94.77**	94.69	96.17	96.05	95.84*	95.98	96.40	95.50	96.43
sCIFAR	63.24**	65.60	67.78	67.63	71.85*	67.73*	70.41	67.29*	71.05
sCIFAR (GRU)	71.30*	64.61	69.81**	70.10	70.74*	70.20*	71.40**	69.17*	71.04

0.5 was also applied to the output classification hidden layer.

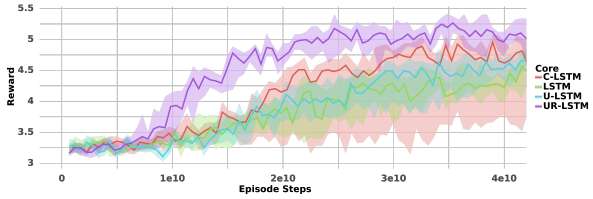
D.3. Language Modeling

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, **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.

Figure 11 shows smoothed validation perplexity curves showing the 95% confidence intervals over the last 1% of data.



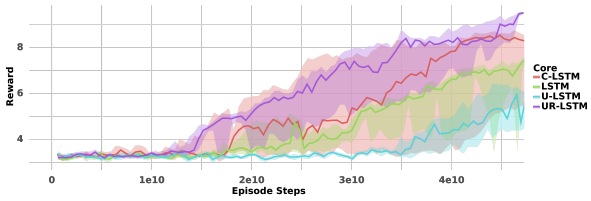
(a) Passive Match without Distractor Rewards



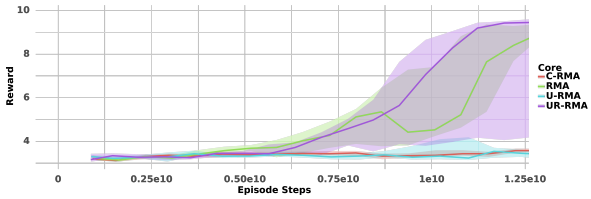
(b) Active Match without Distractor Rewards

Figure 9. **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.

Reinforcement Learning The Active Match and Passive Match tasks were borrowed from Hung et al. (2018) with the same settings. For Figures 9 and 13, the discount factor in the environment was set to $\gamma = .96$. For Figure 10, the discount factor was $\gamma = .998$. Figure 13 corresponds to the full Active Match task in Hung et al. (2018), while Figure 10 is their version with small distractor rewards where the apples in the distractor phase give 1 instead of 5 reward.



(a) Active Match with Distractor Rewards - LSTM



(b) Active Match with Distractor Rewards - RMA

Figure 10. The addition of distractor rewards changes the task and relative performance of different gating mechanisms. For both LSTM and RMA recurrent cores, the UR- gates still perform best.

Figure 8 used 5 seeds per method.

D.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.

D.5. Additional Details

Implementation Details The inverse sigmoid function (12) 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 (12)) and added/subtracted to the forget and input gate ((2)-(3)) 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.

The refine gate update equation (10) can instead be implemented as

$$\begin{aligned} g_t &= r_t \cdot (1-(1-f_t)^2) + (1-r_t)\cdot f_t^2 \\ &= 2r_t \cdot f_t + (1-2r_t)\cdot f_t^2 \end{aligned}$$

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)
```

Listing 3: Bit-reversal permutation for permuted MNIST.

E. Additional Experiments

E.1. Synthetic Forgetting

Figure 5 on the Copy task demonstrates that extremal gate activations are necessary to solve the task, and initializing the activations near 1.0 is helpful.

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.

Here, 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.

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 5.1) of length 500, hidden size 64, learning rate 1e-4. The R-LSTM is able to solve the task, while the LSTM is stuck after 1e4 iterations.

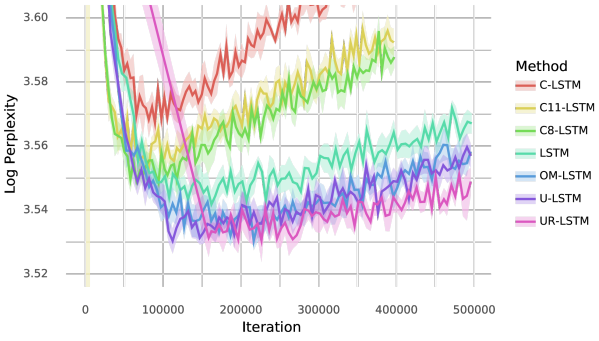
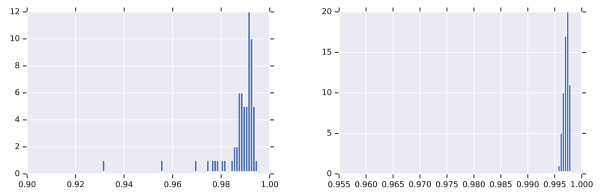


Figure 11. Validation learning curves, illustrating training speed and generalization (i.e. overfitting) behavior.

E.2. Reinforcement Learning

Figures 9 and 10 evaluated our gating methods with the LSTM and RMA models on the Passive Match and Active Match tasks, with and without distractors. We additionally ran the agents on an even harder version of the Active Match task with larger distractor rewards (the full Active Match from Hung et al. (2018)). Learning curves are shown in Figure 13. Similarly to the other results, the UR- gated core is noticeably better than the others. For the DNC model, it is the only one that performs better than random chance.



(a) R-LSTM (b) LSTM

Figure 12. 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.

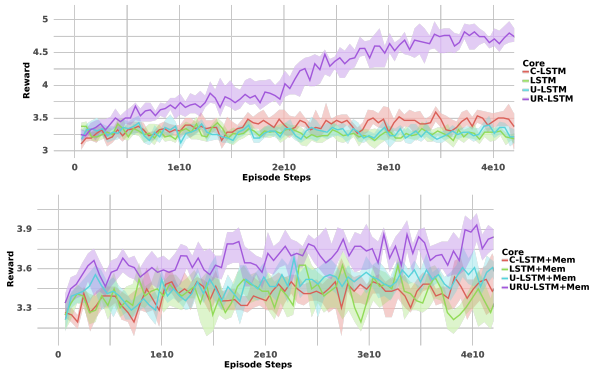


Figure 13. The full Active Match task with large distractor rewards, using agents with LSTM or DNC recurrent cores.

E.3. 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).

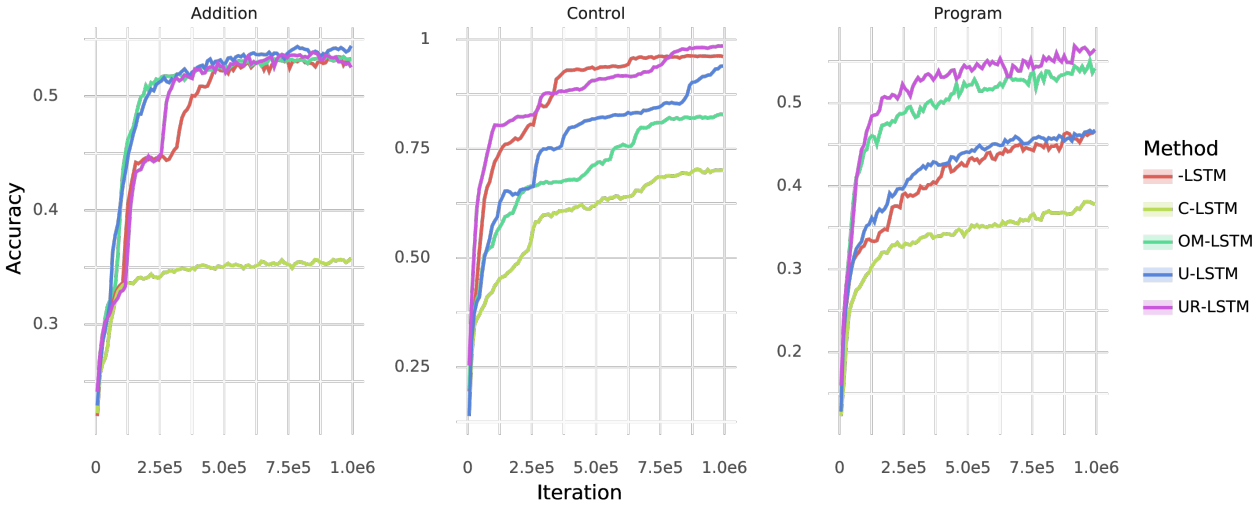
Finally, this task has prior work using a different type of recurrent core, the Relational Memory Core (RMC), that we also use as a baseline to evaluate our gates on different models (Santoro et al., 2018). Both the LSTM and RMC were found to outperform other recurrent baselines such as the Differential Neural Computer (DNC) and EntNet.

Training curves are shown in Figure 14, 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.

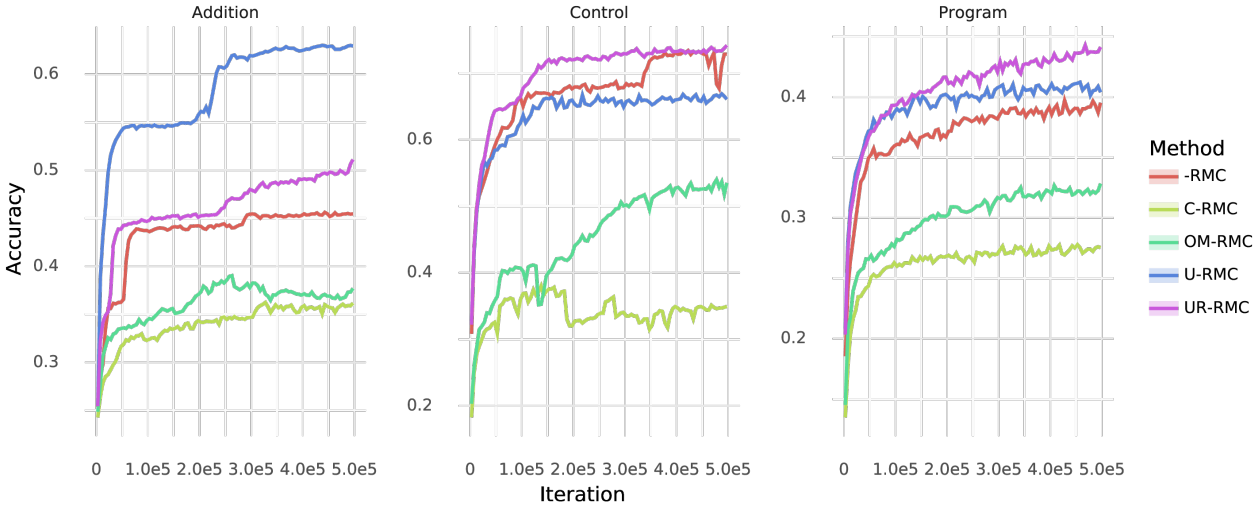
Second, the U-LSTM and U-RMC are the best methods on the Addition task. Additionally, the UR-RMC vs. RMC on Addition is one of the very few tasks we have found where a generic substitution of the UR- gate does not improve on the basic gate. We have not investigated what property of this task caused these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



(a) LSTM - Learning to Execute (nesting=4, length=9)



(b) RMC - Learning to Execute (nesting=4, length=9)

Figure 14. Program Execution evaluation accuracies.