GATO: GATES ARE NOT THE ONLY OPTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent Neural Networks (RNNs) facilitate prediction and generation of structured temporal data such as text and sound. However, training RNNs is hard. Vanishing gradients cause difficulties for learning long-range dependencies. Hidden states can explode for long sequences and send unbounded gradients to model parameters, even when hidden-to-hidden Jacobians are bounded. Models like the LSTM and GRU use gates to bound their hidden state, but most choices of gating functions lead to saturating gradients that contribute to, instead of alleviate, vanishing gradients. Moreover, performance of these models is not robust across random initializations. In this work, we specify desiderata for sequence models. We develop one model that satisfies them and that is capable of learning long-term dependencies, called GATO. GATO is constructed so that part of its hidden state does not have vanishing gradients, regardless of sequence length. We study GATO on copying and arithmetic tasks with long dependencies and on modeling intensive care unit and language data. Training GATO is more stable across random seeds and learning rates than GRUs and LSTMs. GATO solves these tasks using an order of magnitude fewer parameters.

1 INTRODUCTION

RNNs allow us to model structured data like text, image, and sound from domains such as art, healthcare, and planetary sciences. These models form the basis for tasks such as generation and prediction. Modeling joint distributions of sequences facilitates the generation of images (Oord et al., 2016), videos (Denton & Fergus, 2018), drawings (Ha & Eck, 2017), and jazz music (Huang et al., 2019). Modeling conditional distributions enables applications like predicting patient outcomes in medical centers with electronic health records (Che et al., 2018). In all of these cases, the data generation process may involve dependencies that range over hundreds of time steps (e.g. number of video frames or health over a lifetime) where early events may determine the distribution of later events (Kamiya et al., 2015; Denton & Fergus, 2018).

The basic RNN (Elman, 1990) suffers from vanishing and exploding gradient problems (Bengio et al., 1994) that cause difficulties for learning long-range dependencies. The problems stem from extreme eigenvalues of parameter matrices in the RNN. Such models can also suffer from unbounded growth or shrinkage of the hidden state over long sequence lengths. This can happen even when the hidden-to-hidden Jacobian does not vanish or explode and contributes additional difficulties for learning recurrent and decoder parameters. Careful parameterization and initialization of RNNs is crucial for their success in learning long-range dependencies using gradient-based optimization.

The majority of proposed solutions make uses of RNNs with gated units, notably LSTMs (Hochreiter & Schmidhuber, 1997) and GRUs (Cho et al., 2014). These models have been the basis for many state-of-the-art empirical results. Gated architectures address gradient issues by bounding the hidden state and by designing the recurrence to copy the state forward in time. The GRU updates its state with a convex combination of bounded updates and the current state. The LSTM uses an auxiliary cell state with linear recursive updates to copy the hidden state forward. However, gated models are difficult to optimize in practice, in part due to the saturating gradients of gating non-linearities.

An alternative to gated models is to restrict hidden transformations to be unitary (Arjovsky et al., 2016; Jing et al., 2017; Wisdom et al., 2016). Unitary matrices can be safely raised to powers without vanishing gradients since they have modulus 1 eigenvalues. But, these approaches often impose computational overhead to ensure the constraint and require non-trivial parameterizations.

We propose a sequence model called GATO. The key insight in GATO is that only part of the hidden state must have non-vanishing gradients for states in the far past to influence the current state. GATO splits its hidden state into two channels. At each time step, each channel is updated by a function of the input and current hidden state. To ensure that one of the channels will not have vanishing gradients, GATO limits the functions for both channels to only depend on one of the channels. GATO then uses non-saturating functions, such as the cosine, to bound the hidden state upon input to the decoder to stabilize gradients for model parameters. We use a simple version of GATO where the dimensions within the channels do not interact and show that it can model arbitrary conditional distributions through memorization.

We study GATO on two tasks that involve copying and adding on long sequences. GATO solves the copying task while the others do not. On adding, the GRU experiences difficulty as sequence length increases while GATO does not. We also test GATO on health and language data. Using a critical care database collected in Boston (Johnson et al., 2016), GATO predicts patient length-of-stay in the intensive care unit more accurately than GRUs and LSTMs across seeds and learning rates. GATO achieves lower held-out perplexity on language modeling using the Penn Treebank dataset. We explore this in a setting with no methods such as dropout or gradient clipping. GATO solves all of these tasks using an order of magnitude fewer parameters than other models. GATO is stable to train across random initializations compared to GRUs and LSTMs.

2 Background on Sequence Models

Sequence modeling targets learning the sequence of conditional distributions that constitute the joint. Let $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T)$ be a sequence of T observations with joint distribution $Q(\mathbf{x})$. We consider the standard probabilistic model $p_\theta(\mathbf{x})$ with parameter θ, where each sequence element depends on all previous elements. Let $\mathcal{L}_t$ at timestep t be the negative conditional log probability of the next sequence element according to the model:

$$p_\theta(\mathbf{x}) = p_\theta(\mathbf{x}_1) \prod_{t=1}^{T-1} p_\theta(\mathbf{x}_{t+1}|\mathbf{x}_{\leq t}); \qquad \mathcal{L}_t(\theta) = -\log p_\theta(\mathbf{x}_{t+1}|\mathbf{x}_{\leq t}); \qquad \mathcal{L}(\theta) = \sum_t \mathcal{L}_t(\theta).$$

To minimize $\mathcal{L}(\theta)$ on samples from Q, we need expressive p_θ to capture complicated conditionals.

2.1 Sequence Modeling with Neural Networks

Sequence models often represent conditionals using a hidden state that evolves as a sequence of data is observed. Using a function D to map from states to distribution parameters, we define the conditional probability $p(\mathbf{x}_{t+1}|\mathbf{x}_{\leq t})$ with parameters $D(\mathbf{h}_t)$. To represent the true conditionals of Q, $\mathbf{h}_t$ must summarize the information in $\mathbf{x}_1 \ldots \mathbf{x}_t$ useful for predicting $\mathbf{x}_{>t}$. Typically, D is chosen to be a feed-forward neural network. Let $\mathbf{U}$ be hidden-to-hidden parameters, $\mathbf{W}$ be data-to-hidden parameters, and σ be an element-wise non-linearity. RNNs proceed by parameterizing the dynamics of the hidden state via the following discrete-time recurrence:

$$\mathbf{h}_{t+1} = \sigma(\mathbf{U}\mathbf{h}_t + \mathbf{W}\mathbf{x}_{t+1}). \tag{1}$$

Though RNNs can represent a broad class of sequences (Schäfer & Zimmermann, 2006; Siegelmann & Sontag, 1995), they face problems in modeling long sequences where an early part of the sequence influences a later part. On one hand, modeling such data requires the use of powerful functions, but on the other, gradient-based optimization of such functions is unstable and requires manual tuning.

2.2 RNNs Have Problems

Vanishing and Exploding Gradients. Consider two states $\mathbf{h}_t$ and $\mathbf{h}_T$ with times $t \ll T$. Exploding gradients occur when h_T changes dramatically for small changes in h_t, while vanishing gradients occur when the influence of h_t on h_T decreases exponentially with gap $T - t$.

Consider the RNN as in Equation (1) and recall that σ is an element-wise non-linearity. Let $\mathcal{L}_T$ be the loss at time T. Let $\mathbf{D}_t = \texttt{diag}(\sigma'(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t))$ be the Jacobian for σ. Suppose that λ is a lower-bound on the absolute-value of the Jacobian $\mathbf{D}$'s entries. Then if the smallest singular value of

$\mathbf{U}$ is larger than $1/\lambda$, $\partial \mathbf{h}_T / \partial \mathbf{h}_t$ blows up exponentially as time gap $T - t$ increases. Suppose instead that λ is an upper-bound on the absolute-value of $\mathbf{D}$'s entries. Then if the largest singular value of $\mathbf{U}$ is smaller than $1/\lambda$, $\partial \mathbf{h}_T / \partial \mathbf{h}_t$ decays exponentially with time gap $T - t$. The Jacobian $\partial \mathbf{h}_T / \partial \mathbf{h}_t$ involved in $\mathcal{L}_T / \partial \mathbf{U}$ is

$$\frac{\partial \mathcal{L}_T}{\partial \mathbf{U}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{U}},$$

and can stop losses $\mathcal{L}_T$ from backpropagating to $\mathbf{U}$. These issues cause short-term dependencies to be favored over long-term ones (Bengio et al., 1994; Chung et al., 2014).

Exploding Forward Computation. The hidden state can also grow without bound or vanish for long sequences in the hundreds of steps. Consider the following recurrence:

$$\mathbf{z} = \texttt{tanh}(\mathbf{W}^{(1)} \mathbf{x}_{t+1} + \mathbf{U} \mathbf{h}_t); \qquad \mathbf{h}_{t+1} = \mathbf{z} \odot \mathbf{h}_t + (1 - \mathbf{z}) \odot \texttt{tanh}(\mathbf{W}^{(2)} \mathbf{h}_t). \tag{2}$$

If $\mathbf{h}$ is ever greater than 1 in any dimension, the state will grow over a long sequence under a variety of conditions on $\mathbf{W}^{(1)}$ and $\mathbf{W}^{(2)}$. This can cause gradient problems for $\mathbf{U}$ and is discussed in Section 3.

2.3 Long Short-Term Memory and Gated Recurrent Units

To improve gradients, Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) and Gated Recurrent Units (GRU) (Cho et al., 2014) augment RNNs with additional computation that helps bound the state as well as reduce gradient vanishing. The LSTM retains a cell state that copies the hidden state forward in time. The GRU uses a convex combination of the current state and a bounded proposal as in Equation (2). In this setting, the hidden state is usually initialized to $\mathbf{0}$ and the GRU equation keeps the state bounded. See Appendix D for more details.

This variety of neural sequence models has produced state-of-the-art performance in machine translation (Sutskever et al., 2014; Luong et al., 2015) and image captioning (Xu et al., 2015). However, these models are difficult to optimize. Practitioners must experiment with optimization techinques including gradient clipping, randomly dropping-out hidden state updates (Krueger et al., 2016), and even training discriminators to keep generated and data sequences close (Lamb et al., 2016). Empirical investigation into the behavior of the LSTM gates has also shown that forget and input gate values tend toward 0.5, contradicting explicit interpretations as gates (Li et al., 2018b). Recent analysis has shown that LSTMs still have gradient issues that cause them to favor learning short-term dependencies in data (Kanuparthi et al., 2019). See Appendix F for a brief discussion.

3 Desiderata for Sequence Models

We now develop criteria for building sequence models and consider several important gradients.

Hidden-to-Hidden gradients. For sequence models to learn long-term dependencies, part of the hidden state at time t needs to have non-decaying influence on states at later times T. Therefore only some, not all, of the hidden units need to avoid vanishing gradients. Our first criterion is:

1. Part (a block) of the Jacobian $\partial \mathbf{h}_T / \partial \mathbf{h}_t$ does not depend on time gap $T - t$.

Decoder Gradients. Let γ be a non-linear function and consider a loss at time t defined through a linear multi-output decoder $D(\mathbf{h}_t) = \mathbf{W} \gamma(\mathbf{h}_t)$. Then the gradient of the loss with respect to $\mathbf{W}$ is:

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_t}{\partial D} \frac{\partial D}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}_t}{\partial D} \gamma(\mathbf{h}_t)^\top$$

To ensure that $||\partial \mathcal{L}_t / \partial \mathbf{W}||$ does not become too large, one property that γ needs to have is:

2. γ is bounded to ensure that decoder gradients $\partial D / \partial \mathbf{W}$ do not grow with time.

Loss-to-Hidden Gradients. $\partial \mathcal{L}_t / \partial \mathbf{h}_t$ for one time step t also depends on the choice of γ. It is defined through the decoder by

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{h}_t} = \frac{\partial \mathcal{L}_t}{\partial D} \frac{\partial D}{\partial \mathbf{h}_t} = \left(\mathbf{W}^\top \frac{\partial \mathcal{L}_t}{\partial D} \right) \odot \gamma'(\mathbf{h}_t)$$

This means that:

3. γ's derivative γ' evaluated at $\mathbf{h}_t$ should not converge to $\mathbf{0}$ as $||\mathbf{h}_t||$ grows.
4. The magnitude $||\gamma'||$ evaluated at $\mathbf{h}_t$ is bounded even when $||\mathbf{h}_t||$ increases.

These criteria require bounded, differentiable functions whose gradients do not go to $\mathbf{0}$ as their input increases. This rules out traditional choices of non-linear functions that saturate, such as sigmoid and `tanh`, and functions that grow such as `SoftPlus` and `ReLU`. Periodic functions such as `cos` and `sin` meet these criteria.

Forward Computation. In tasks such as language modeling, the hidden state can be processed for thousands of steps through the corpus. Let $\mathbf{r}_t$ denote the part of $\mathbf{h}_t$ involved in the recurrence to compute $\mathbf{h}_{t+1}$. Then if $||\mathbf{r}_t||$ is large at time t during training, $||\partial \mathcal{L}_{t+1} / \partial \mathbf{U}||$ will be large and can cause additional issues in learning recurrent parameters $\mathbf{U}$.

5. $||\mathbf{r}_t||$ in general should not become large as t increases.

4 GATO

We propose a new sequence model that meets the desiderata called GATO. GATO is capable of learning long-range dependencies and is stable to train, exhibiting little variance across learning rates and initializations. GATO requires fewer parameters to perform well on tasks such as copying, arithimetic, classification, and language modeling. We describe the model and analyze its gradients.

4.1 GATO MODEL

Building on the ideas in Section 3, GATO partitions the RNN hidden state into two parts, one of which is guaranteed to have good long-term gradients by construction. The updates of the hidden state impose a structure on the hidden-to-hidden Jacobian wherein a sub-matrix is the identity $\mathbf{I}$. This means that parts of $\partial \mathbf{h}_T / \partial \mathbf{h}_t$ do not dependent on $T - t$ and therefore do not vanish with time.

Partitioned Hidden State. The hidden state $\mathbf{h}_t$ has two channels $\mathbf{r}_t$ and $\mathbf{s}_t$. At each time step, both channels are updated by a non-linear function of the previous state. To ensure that one part of the state, $\mathbf{s}_t$, has time-independent hidden-to-hidden gradients, the updates do not depend on $\mathbf{s}$. Let f_ϕ and g_ψ be non-linear functions with parameters ϕ and ψ. GATO's hidden state evolves as:

$$\mathbf{r}_{t+1} = f_\phi(\mathbf{r}_t, \mathbf{x}_{t+1}); \qquad \mathbf{s}_{t+1} = \mathbf{s}_t + g_\psi(\mathbf{r}_t, \mathbf{x}_{t+1}); \qquad \mathbf{h}_{t+1} = [\mathbf{r}_{t+1}, \mathbf{s}_{t+1}].$$

This is an RNN where the hidden state update is only a function of half of the state, and where half of the updates are defined by residual connections (He et al., 2016; Wang & Tian, 2016).

4.2 GATO HAS GOOD GRADIENTS

Hidden-to-Hidden Gradients. As shown in Section 2.2, RNNs suffer from vanishing gradients. We show that the GATO hidden state recurrence does not have vanishing gradients for part of the hidden state. First consider $\partial \mathbf{s}_T / \partial \mathbf{s}_t$ and recall the evolution of $\mathbf{r}$ and $\mathbf{s}$:

$$\mathbf{r}_T = f_\phi(f_\phi(...f_\phi(\mathbf{r}_t, \mathbf{x}_{t+1}))); \quad \mathbf{s}_T = \mathbf{s}_t + \sum_{k=t}^{T-1} g_\psi(\mathbf{r}_k, \mathbf{x}_{k+1}).$$

The right hand side for $\mathbf{r}_T$ and the second term of the right hand side for $\mathbf{s}_T$ do not depend on $\mathbf{s}_t$, so $\partial \mathbf{s}_T / \partial \mathbf{s}_t$ is the identity matrix $\mathbf{I}$. Next, $\partial \mathbf{r}_T / \partial \mathbf{s}_t = \mathbf{0}$ because $\mathbf{s}$ is not included among the arguments to the update $f_\phi(\mathbf{r}_t, \mathbf{x}_t)$. The whole Jacobian matrix of hidden states $\mathbf{h}_T$ with respect to $\mathbf{h}_t$ is:

$$\frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} = \begin{bmatrix} \frac{\partial \mathbf{s}_T}{\partial \mathbf{s}_t} & \frac{\partial \mathbf{s}_T}{\partial \mathbf{r}_t} \\ \frac{\partial \mathbf{r}_T}{\partial \mathbf{s}_t} & \frac{\partial \mathbf{r}_T}{\partial \mathbf{r}_t} \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \sum_{k=t}^{T-1} \frac{\partial g_\psi(\mathbf{r}_k, \mathbf{x}_{k+1})}{\partial \mathbf{r}_k} \frac{\partial \mathbf{r}_k}{\partial \mathbf{r}_t} \\ \mathbf{0} & \prod_{\ell=t}^{T-1} \frac{\partial f_\phi(\mathbf{r}_\ell, \mathbf{x}_{\ell+1})}{\partial \mathbf{r}_\ell} \end{bmatrix}. \tag{3}$$

Since $\partial \mathbf{s}_T / \partial \mathbf{s}_t = \mathbf{I}$ does not depend on time difference $T - t$, part of the hidden state, $\mathbf{s}_T$, retains its influence from $\mathbf{s}_t$ (criterion 1). Though $\partial \mathbf{s}_T / \partial \mathbf{r}_t$ and $\partial \mathbf{r}_T / \partial \mathbf{r}_t$ are complicated, $\partial \mathbf{s}_T / \partial \mathbf{s}_t$ is guaranteed not to vanish so gradients can be propagated back through time. In Appendix A, we explore the effect of removing this feature and find that model performance suffers.

Decoder and Loss-to-Hidden Gradients. We have just shown that the hidden state channel $\mathbf{s}_t$'s gradient does not vanish. As discussed in Section 3, there are additional optimization issues that can be introduced by the hidden state. We focus on $\mathbf{s}_t$. In the case of loss defined through decoder $D(\mathbf{s}_t) = \mathbf{W}\gamma(\mathbf{s}_t)$, γ must be chosen to ensure that $||\partial \mathcal{L}_t / \partial \mathbf{W}||$ and $||\partial \mathcal{L}_T / \partial \mathbf{s}_t||$ do not shrink or grow, regardless of $||\mathbf{s}_t||$. In the experiments, GATO ensures this by using $\gamma(\mathbf{s}_t) = \texttt{cos}(\mathbf{s}_t)$.

4.3 Non-Interacting Hidden Units

Consider a version of GATO where interactions are limited to pairs of units. Let j index pairs of hidden units and let the hidden state size be $2J$. Then the updates for each pair $(r^{(j)}, s^{(j)})$ are:

$$r_{t+1}^{(j)} = f_\phi^{(j)}(r_t^{(j)}, \mathbf{x}_{t+1}); \qquad s_{t+1}^{(j)} = s_t^{(j)} + g_\psi^{(j)}(r_t^{(j)}, \mathbf{x}_{t+1}). \tag{4}$$

Non-interacting units in RNN hidden states have been explored previously (Li et al., 2018a). We show that this version of GATO can represent arbitrary conditionals by memorization. Suppose we have a sequence model of scalar data with conditionals defined via hidden states $p(x_{t+1}|x_{\leq t}) = p(x_{t+1}; D(\mathbf{s}_t, \mathbf{r}_t))$. For this model to be able to represent arbitrary conditionals, the hidden state needs to store $\mathbf{x}_{\leq t}$. One way for the version of GATO in Equation (4) to memorize its inputs $\mathbf{x}$ would be for each $r^{(j)}$ to learn how to count and each function $g_\phi^{(j)}$ to store its one-dimensional input x in $s^{(j)}$ when the count $r^{(j)}$ equals j. A powerful decoder can then read the memorized sequence from the hidden state and compute a conditional distribution. This construction requires the size J to be at least the length of the sequence T.

4.4 Two Examples of Non-Interacting GATO

Let $\mathbf{x}_t \in \mathbb{R}^D$ denote the input at time t. Let $\mathbf{h}_t = [\mathbf{r}_t, \mathbf{s}_t]$ denote the hidden state with $\mathbf{r}_t, \mathbf{s}_t \in \mathbb{R}^J$. Let $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{J \times D}$ and $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^J$ be parameters. Let σ be the sigmoid. Let $\lambda \in \mathbb{R}$.

One layer GATO.

$$\begin{aligned} \mathbf{s}_{t+1} &= \mathbf{s}_t + \texttt{SoftPlus}(\mathbf{A}\mathbf{x}_{t+1} + \mathbf{a} \odot \mathbf{r}_t); \\ \mathbf{r}_{t+1} &= \lambda \cdot \sigma(\mathbf{B}\mathbf{x}_{t+1} + \mathbf{b} \odot \mathbf{r}_t) \odot \mathbf{r}_t + \texttt{tanh}(\mathbf{C}\mathbf{x}_{t+1} + \mathbf{c} \odot \mathbf{r}_t). \end{aligned} \tag{5}$$

Two layer GATO. Let $\texttt{F}(\mathbf{r}, \mathbf{x})$ denote a neural network with inputs $\mathbf{r}, \mathbf{x}$ such that each output dimension F_j is a one hidden layer neural network with $\texttt{ReLU}$ activation and hidden size k that only depends on r_j and $\mathbf{x}$. Then two-layer GATO is defined as:

$$\begin{aligned} \mathbf{s}_{t+1} &= \mathbf{s}_t + \texttt{SoftPlus}(\texttt{F}(\mathbf{x}_{t+1}, \mathbf{r}_t)); \\ \mathbf{r}_{t+1} &= \lambda \cdot \sigma(\mathbf{A}\mathbf{x}_{t+1} + \mathbf{a} \odot \mathbf{r}) \odot \mathbf{r}_t + \texttt{tanh}(\mathbf{B}\mathbf{x}_{t+1} + \mathbf{b} \odot \mathbf{r}). \end{aligned} \tag{6}$$

In both models, setting λ to be a number less than one helps meet criterion 5 in Section 3.

5 Experiments

We study GATO on two standard sequence tasks, copying and performing arithmetic on long sequences (Hochreiter & Schmidhuber, 1997; Arjovsky et al., 2016). We then use GATO for hospital length of stay prediction using electronic health records and on language modeling. We compare against the GRU and LSTM. We run experiments across three random seeds and choices of learning rates. We explore this in a setting with no gradient clipping or dropout.

5.1 Model and Baseline Specifications

For baselines, we use PyTorch modules `nn.GRU` and `nn.LSTM` (Paszke et al., 2017). For GATO, we choose $\lambda = 0.7$. In our experiments, we find $\lambda = 0.5$ performed similar to 0.7. We give exact numbers of RNN parameters for each experiment. Embedding and decoder parameters are constant across models and excluded in counts. More details including initialization are described in Appendix B. All models are trained across three random seeds for all tasks. The error bands in all plots represent the minimum and maximum across seeds with a bold line representing the mean.

5.2 Experiment 1: Long Copy Task

Task Description. We model sequences of the form `ABA`, where `A` is a sequence of length 20 with tokens uniformly drawn from $\{1, \ldots, 10\}$ and `B` is 100 constant `BLANK` tokens. The model must predict the second `A` as a copy of the first. The hidden state must memorize the first 20 tokens and keep them in memory while processing the 100 `BLANK`'s. This is a harder variant of the copy task in (Arjovsky et al., 2016). We do not use a special token to mark the beginning of the second `A` region.

Experiment Setup. All models use an input embedding size of 4, recurrent hidden size of 1024, and a fully-connected decoder with one hidden layer of size 256. We use two layer GATO. GATO has $138,752$ parameters,while the GRU and LSTM have $3,118,080$ and $4,222,976$, respectively. We draw new sequences from the data distribution for each step of training. We evaluate on a held-out set of 1000 sequences. We let each model see 1 million training points.

Results. We show the results in Figure 1. For the LSTM and GRU, the probability of the tokens to be copied is near chance (0.10) across the three seeds. GATO learns the task with at least .50 probability averaged across the held-out set for all tokens to be copied.

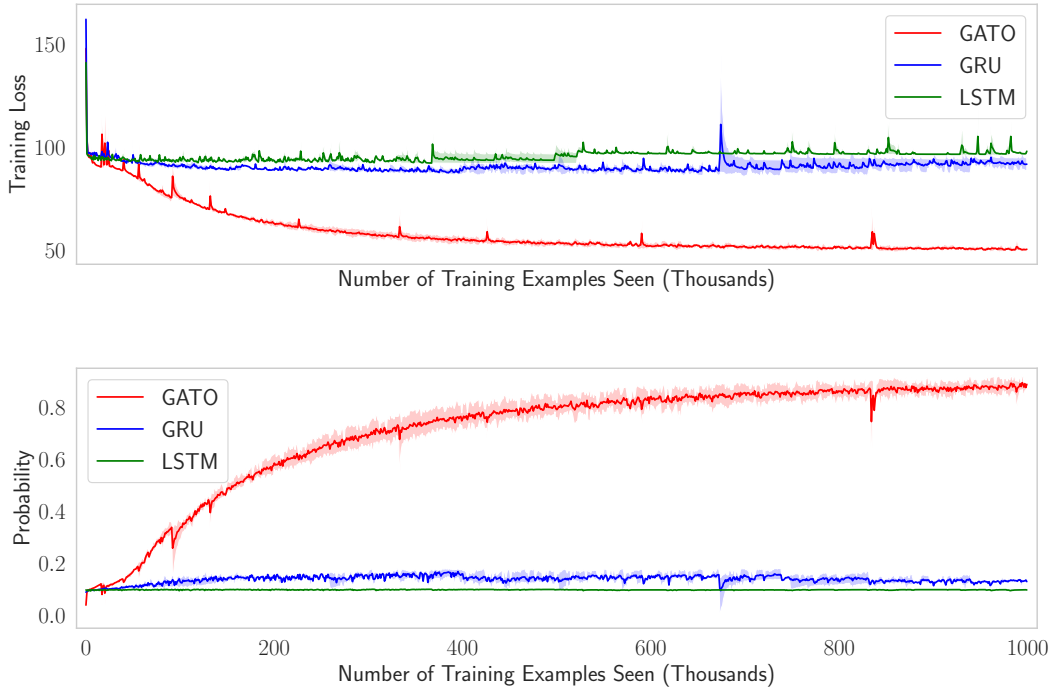

Figure 1: **GATO learns to copy long sequences with an order of magnitude fewer parameters than the GRU and LSTM.** The GRU and LSTM output probabilities near chance for the copied tokens. Top: Training Loss. Bottom: Held Out Average Probability of Copied Tokens.

5.3 EXPERIMENT 2: ADDING PROBLEM

Task Description. This experiment follows the adding task described in (Hochreiter & Schmidhuber, 1997). In this task, the input consists of two sequences of length T. Each element of the first sequence is sampled from the uniform distribution $\mathcal{U}[0, 1]$. The second sequence plays the role of an indicator. It is all 0's except for two 1's. The location of the first 1 is sampled uniformly from the first half of the sequence, while the location of the second 1 is sampled uniformly from the second half of the sequence. The task is to predict the sum of the two numbers in the first sequence whose locations correspond to the locations of the 1's in the second sequence. The mean squared error of predicting the expectation, 1, is 0.167. This forms the baseline.

Experiment Setup. We compare GATO, GRU and LSTM with the baseline and we use two layer GATO. Loss is measured by mean squared error. We do not use embeddings for this task, so the input size at each timestep is 2 (uniform variate and indicator). All models use recurrent hidden size 512 and a decoder with one hidden layer of size 256. For recurrent parameters, GATO has 43,264 parameters while the GRU and LSTM have 792,576 and 1,056,768 respectively. To make a prediction, we apply the decoder to the RNN hidden state after processing the sequence. We sample new data points at each step of training. We evaluate on a held-out set of 1000 examples. We run experiment with sequence lengths $T = 100, 200, 400, 750$.

Results. Our results are shown in Figure 2. After training with 200,000 examples, the mean squared error of the LSTM was still around the baseline. For short sequence lengths, both the GRU and GATO learned the task. As the sequence length increases, the GRU needs more time to learn the task. At the longest length $T = 750$, random initializations of the GRU fail to beat the baseline.

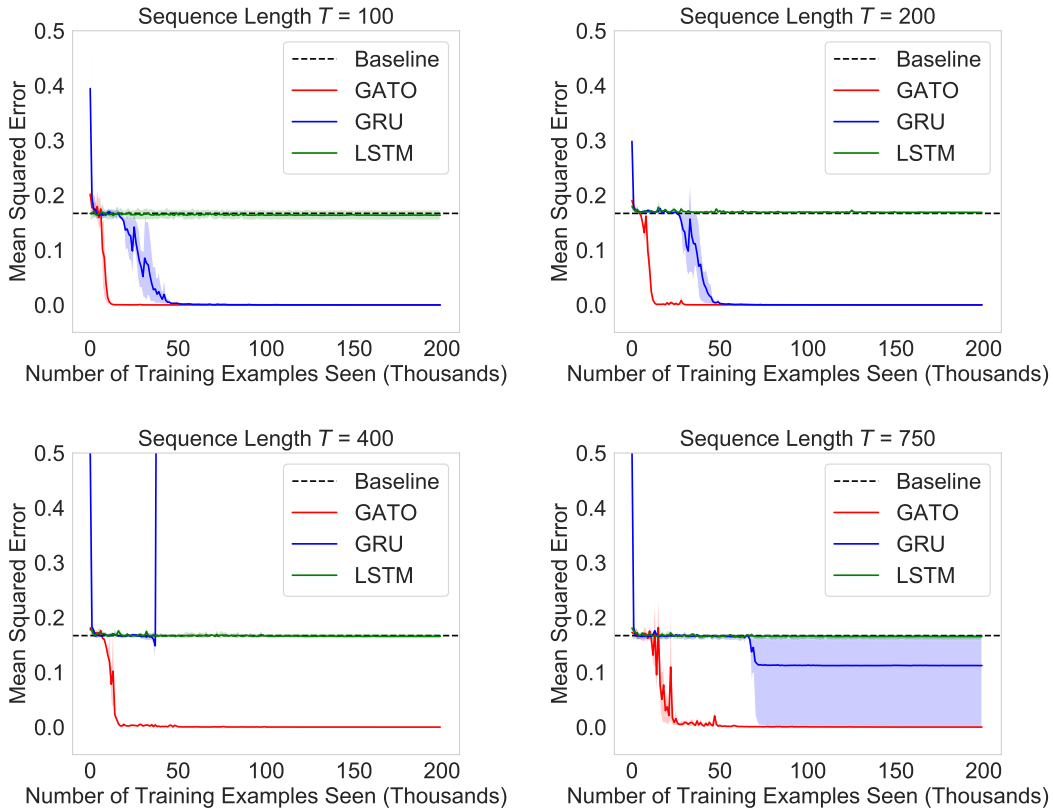

Figure 2: **GATO is more robust across different seeds and sequence lengths.** We show the mean squared errors of the adding problem for sequence lengths $T = 100, 200, 400, 750$ over number of training examples. LSTM failed to learn the task. For length $T = 400$, the GRU numerically failed on one seed (NAN error). We denote this by infinite error.

Table 1: **GATO beats LSTM and GRU on test accuracy with both learning rates 4e-3 and 1e-4.** We show the accuracy of length of stay prediction averaged across seeds. GATO at either learning rate has higher accuracy than GRU and LSTM. GATO has 40 times fewer parameters.

Model	# Parameters	Test Accuracy, LR 4e-3	Test Accuracy, LR 1e-4
LSTM	1,103,872	0.6503	0.5694
GRU	827,904	0.5192	0.5691
GATO	**20,736**	**0.6934**	**0.6460**

Table 2: **GATO competes with the LSTM and GRU on test perplexity scores.** We calculate the perplexity averaged over three seeds on Penn Treebank language modeling. GATO gets similar perplexity with 6 times fewer parameters.

Model	# Parameters	Test Perplexity Score
LSTM	10,150,400	122.86
GRU	7,612,800	115.78
GATO	**1,273,350**	**112.85**

5.4 Experiment 3: Length of Stay in the Intensive Care Unit

Task Description. This experiment uses MIMIC-III, a Boston-based critical care database. The task is to predict if a patient stays more than 1 week in the intensive care unit using the first 48 hours of their vitals. There are 53,211 patients in total, 27,088 of whom stay more than one week. It is a binary classification problem with almost balanced classes. We use the twelve features extracted for Interpolation-Prediction Networks (Shukla & Marlin, 2019). See Appendix B for more details.

Experiment Setup. We split the data into a training set of size 43,000, validation set of size 5,106, and test set of 5,105. We use cross entropy loss. To make a prediction, we apply a decoder to the RNN hidden state after processing the sequence of vital signs. We use a recurrent hidden size 512 and decoder hidden size 1024 for all models. We use one-layer GATO.

Results. We report average test accuracy across seeds in Table 1. GATO has higher accuracy than GRU and LSTM on this task despite having 1/40 of the model parameters. We study two learning rates. While performance varies across rates, GATO performs best on each rate. Even at the worse of the two rates, GATO performs about as well as the LSTM and GRU at either learning rate.

5.5 Experiment 4: Language Modeling on Penn TreeBank

Description and Results. A language model predicts the probability distribution of the next word in a sentence based on the previous words. We test GATO on the Penn Treebank dataset preprocessed by Mikolov et al. (2010). We choose hidden size 1300, embedding size 650 and a linear decoder for all the models for this task. We use one-layer GATO in this experiment. Table 2 shows model perplexity on Penn Treebank. GATO beats GRU and LSTM by perplexity score in our setting of the experiment. GATO enjoys a fewer number of parameters.

6 Conclusion and Future Work

GATO uses partitioned hidden states. One part of the hidden state is modified, but never used to update future states. This ensures that part of the hidden state does not have vanishing gradients. We also demonstrate the power of non-interacting hidden units. GATO performs as well as GRUs and LSTMs in four typical sequence modeling problems using an order of magnitude fewer parameters. Future directions include understanding when forward propagation is stable and how the principles used to develop GATO can be used to design feed-forward neural networks.

REFERENCES

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

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.

Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):6085, 2018.

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.

Emily Denton and Rob Fergus. Stochastic video generation with a learned prior. *arXiv preprint arXiv:1802.07687*, 2018.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 1990.

David Ha and Douglas Eck. A neural representation of sketch drawings. *arXiv preprint arXiv:1704.03477*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Ian Simon, Curtis Hawthorne, Noam Shazeer, Andrew M. Dai, Matthew D. Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJe4ShAcF7`.

Li Jing, Yichen Shen, Tena Dubcek, John Peurifoy, Scott Skirlo, Yann LeCun, Max Tegmark, and Marin Soljacic. Tunable efficient unitary neural networks (eunn) and their application to rnns. In *Proceedings of the 34th International Conference on Machine Learning*, 2017.

Alistair EW Johnson, Tom J Pollard, Lu Shen, H Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3:160035, 2016.

Kenji Kamiya, Kotaro Ozasa, Suminori Akiba, Ohstura Niwa, Kazunori Kodama, Noboru Takamura, Elena K Zaharieva, Yuko Kimura, and Richard Wakeford. Long-term effects of radiation exposure on health. *The lancet*, 386(9992):469–478, 2015.

Bhargav Kanuparthi, Devansh Arpit, Giancarlo Kerg, Nan Rosemary Ke, Ioannis Mitliagkas, and Yoshua Bengio. h-detach: Modifying the lstm gradient toward better optimization. In *International Conference on Learning Representations*, 2019.

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.

Alex M Lamb, Anirudh Goyal Alias Parth Goyal, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pp. 4601–4609, 2016.

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. In *Proceedings of 35th International Conference on Machine Learning*, 2018b.

Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=SyyGPP0TZ`.

Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *31st Conference on Neural Information Processing Systems, Workshop paper*, 2017.

Anton Maximilian Schäfer and Hans Georg Zimmermann. Recurrent neural networks are universal approximators. In *International Conference on Artificial Neural Networks*, pp. 632–640. Springer, 2006.

Satya Narayan Shukla and Benjamin M. Marlin. Interpolation-prediction networks for irregularly sampled time series. In *International Conference on Learning Representations*, pp. 4880—-4888, 2019.

Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Yiren Wang and Fei Tian. Recurrent residual learning for sequence classification. In *Conference on Empirical Methods in Natural Language Processing*, 2016.

Scott Wisdom, Thomas Powers, John Hershey, Jonathan Le Roux, and Les Atlas. Full-capacity unitary recurrent neural networks. In *Advances In Neural Information Processing Systems*, pp. 4880—-4888, 2016.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.

A GATO CANNOT LEARN ADDING WITHOUT LONG-TERM GRADIENT: ABLATION STUDY

Recall that GATO partitions its hidden state as $\mathbf{h}_t = [\mathbf{s}_t, \mathbf{r}_t]$ and that $\mathbf{s}_t$ and $\mathbf{r}_t$ are updated by:

$$\mathbf{r}_{t+1} = f_\phi(\mathbf{x}_{t+1}, \mathbf{r}_t)$$
$$\mathbf{s}_{t+1} = \mathbf{s}_t + g_\psi(\mathbf{x}_{t+1}, \mathbf{r}_t)$$

We experiment with two ablations of GATO for the adding task and show that the long-term gradients through $\partial \mathbf{s}_T / \partial \mathbf{s}_t$ are necessary for its performance.

1. Set the update for $\mathbf{s}$ to $\mathbf{0}$. This makes GATO close to a standard RNN for half of its state.
2. Set the update for $\mathbf{s}$ to g_ψ. This is a standard RNN with half of its state but where the decoder can use g_ψ to compute an additional function of the state $\mathbf{r}$.

Both of these ablations lose GATO's identity block matrix in the Jacobian $\partial \mathbf{h}_T / \partial \mathbf{h}_t$. In these experiments, GATO does not learn the adding task for sequence length 750 above chance for either ablation, across three seeds.

B EXPERIMENTAL DETAILS

B.1 MIMIC-III

Features. To predict hospital length of stay on the MIMIC-III dataset, we use the same features extracted for Interpolation-Prediction Networks (Shukla & Marlin, 2019). The features include blood oxygen level, heart rate, respiratory rate, systolic blood pressure, diastolic blood pressure, temperature, oxygen saturation, and others. Not all features are recorded at each observation time. We use 25 features including 12 vitals, 12 additional missingness indicators for those features, and a measurement time.

Continuous Embeddings. All models apply a linear transformation followed by an element-wise `tanh` function to the continuous input before it is passed to the RNN. The linear transformation is learned alongside other model parameters.

B.2 INITIALIZATION

Recall that experiment 1 is copying, experiment 2 is adding, experiment 3 is hospital stay prediction, and experiment 4 is language modeling on Penn Treebank. We use different initializations for GATO in experiments 1 and 2 than we do for experiments 3 and 4. In experiments 1 and 2, we use initialization $\mathcal{U}[-0.1, 0.1]$ for all RNN parameters in GATO. In experiments 3 and 4, we use initialization $\mathcal{U}[-0.1, 0.1]$ for all RNN parameters besides the weight matrices in the linear layers used to project $\mathbf{x}$. For those, we use the default PyTorch initializations. For embeddings and decoders for all models, as well as all GRU and LSTM parameters, we use default PyTorch initializations.

B.3 LEARNING RATE AND OPTIMIZER

We use Adam optimizer for all experiments. We use learning rate 4e-3 for experiments 1 and 2, and 1e-4 for experiment 4. We check both learning rates 4e-3 and 1e-4 for experiment 3. In experiment 2, we start with learning rate 0.004 and check the training loss after seeing every 10,000 examples. If the training loss is larger than for the previous 10,000 examples, we divide the learning rate by 2.

B.4 VALIDATION AND TEST SCORES

On MIMIC-III and Penn Treebank, we report test set accuracy corresponding to each model's predictions when they achieved their best validation accuracy. Perplexity on the language modeling task is defined as in Merity et al. (2018).

B.5 BATCH SIZE

We use batch size 32, 64, 256, and 20 in experiment 1, 2, 3, and 4 respectively.

B.6 WEIGHT DECAY

We use 1.2e-6 weight decay (the default number in Merity et al. (2018)) in experiments 3 and 4. Since there is no generalization problem in experiments 1 and 2 (because data samples are drawn from the true generating distribution), we do not use weight decay in those experiments.

B.7 LANGUAGE MODEL TRAINING

We follow the training strategy in (Merity et al., 2018). With probability 0.95, we take a batch with sentence length sampled from $\mathcal{N}(70, 5^2)$ (then rounded to an integer) and with probability 0.05 we take a batch with sentence length sampled from $\mathcal{N}(35, 5^2)$ (then rounded to an integer).

B.8 OTHER TRICKS

We do not use dropout, gradient clipping and other tricks in all the experiments.

C VANISHING AND EXPLODING GRADIENTS

Consider the RNN as in Equation (1) and recall that σ is an element-wise non-linearity:

$$\mathbf{h}_{t+1} = \sigma(\mathbf{U}\mathbf{h}_t + \mathbf{W}\mathbf{x}_{t+1}).$$

Let $\mathcal{L}_T$ be the loss at time T. Let $\mathbf{D}_t = \texttt{diag}(\sigma'(\mathbf{U}\mathbf{h}_{t-1} + \mathbf{W}\mathbf{x}_t))$ be the Jacobian for σ.

Then the gradient of the loss $\mathcal{L}_T$ with respect to $\mathbf{U}$ is:

$$\frac{\partial \mathcal{L}_T}{\partial \mathbf{U}} = \sum_{t=1}^{T} \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} \frac{\partial \mathbf{h}_T}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{U}} = \sum_{t=1}^{T} \left(\left(\prod_{k=t}^{T} \mathbf{U}^\top \mathbf{D}_k \right) \frac{\partial \mathcal{L}_T}{\partial \mathbf{h}_T} \mathbf{h}_{t-1}^\top \right).$$

Suppose that λ is a lower-bound on the absolute-value of the Jacobian $\mathbf{D}_k$'s entries. Then if the smallest singular value of $\mathbf{U}$ is larger than $1/\lambda$, $\partial \mathbf{h}_T / \partial \mathbf{h}_t$ blows up exponentially with time gap $T - t$. This is called the exploding gradient problem. Suppose instead that λ is an upper-bound on the absolute-value of $\mathbf{D}_k$'s entries. Then if the largest singular value of $\mathbf{U}$ is smaller than $1/\lambda$, $\partial \mathbf{h}_T / \partial \mathbf{h}_t$ decays exponentially with time gap $T - t$. This is called the vanishing gradient problem. The term $\partial \mathbf{h}_T / \partial \mathbf{h}_t$ in the gradient calculation for $\partial \mathcal{L}_T / \partial \mathbf{U}$ is therefore problematic for models like the RNN. Such issues can stop losses $\mathcal{L}_T$ at later time steps T from backpropagating to $\mathbf{U}$ to adjust the computation of earlier states $\mathbf{h}_t$ for $t \ll T$.

D LSTM

The LSTM (Hochreiter & Schmidhuber, 1997) maintains an auxilary cell state that helps copy the hidden state forward in time as a sequence is processed. $\mathbf{c}_t$ is called the cell state, and $\mathbf{h}_t$ is called the hidden state. Letting $\mathbf{f}_t, \mathbf{i}_t, \mathbf{g}_t$ and $\mathbf{o}_t$ be non-linear functions of $\boldsymbol{h}_{t-1}$ and $\mathbf{x}_t$ and letting $\odot$ denote element-wise product, the LSTM hidden state computation for $\mathbf{c}$ and $\mathbf{h}$ is defined by

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t; \qquad \mathbf{h}_t = \mathbf{o}_t \odot \texttt{tanh}(\mathbf{c}_t). \tag{7}$$

In the update of the cell state, the *forget* gate $\mathbf{f}_t$ determines the influence of previous cell states on the next. The *input* gate $\mathbf{i}_t$ determines the influence of new information on the cell. $\mathbf{f}_t$, $\mathbf{i}_t$, $\mathbf{o}_t$ and $\mathbf{g}_t$ are each defined via usual RNN recurrences:

$$\begin{aligned} \mathbf{g}_t &= \texttt{tanh}(\mathbf{W}_{gh}\mathbf{h}_{t-1} + \mathbf{W}_{gx}\mathbf{x}_t + \mathbf{b}_g), \quad & \mathbf{f}_t &= \sigma(\mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{W}_{fx}\mathbf{x}_t + \mathbf{b}_f), \\ \mathbf{i}_t &= \sigma(\mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{W}_{ix}\mathbf{x}_t + \mathbf{b}_i), \quad & \mathbf{o}_t &= \sigma(\mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{W}_{ox}\mathbf{x}_t + \mathbf{b}_o). \end{aligned}$$

E GATED RECURRENT UNITS

The GRU (Cho et al., 2014) is similar to the LSTM but with fewer gates and parameters. First, two RNN-like functions $\mathbf{r}$ and $\mathbf{z}$ of inputs $\mathbf{x}$ and $\mathbf{h}_{t-1}$ are computed. They are called the reset and update gates, respectively. Then, a third RNN-like function with inputs $\mathbf{x}$, $\mathbf{h}_{t-1}$, and $\mathbf{r}$ computes a proposed state $\tilde{\mathbf{h}}$. Finally, the new state is a convex combination of the previous and proposed states. Let σ be the sigmoid and ϕ be any non-linearity like the `tanh`. Then the GRU computation is:

$$
\begin{aligned}
\mathbf{r} &= \sigma\Big(\mathbf{W}_r\mathbf{x} + \mathbf{U}\mathbf{h}_{t-1}\Big) \\
\mathbf{z} &= \sigma\Big(\mathbf{W}_z\mathbf{x} + \mathbf{U}\mathbf{h}_{t-1}\Big) \\
\tilde{\mathbf{h}} &= \phi\Big(\Big(\mathbf{W}\mathbf{x} + \mathbf{U}\Big(\mathbf{r} \odot \mathbf{h}_{t-1}\Big)\Big)\Big) \\
\mathbf{h} &= \mathbf{z} \odot \mathbf{h}_{t-1} + (1 - \mathbf{z}) \odot \tilde{\mathbf{h}}
\end{aligned}
$$

The expression for $\mathbf{h}$ is a convex combination of $\mathbf{h}_{t-1}$ and proposed update $\tilde{\mathbf{h}}$. Therefore, if ϕ is chosen to be bounded by 1 and if $\mathbf{h}$ starts with all coordinates in $(0, 1)$, the state will stay bounded.

F LSTMS ALSO HAVE GRADIENT PROBLEMS

By the form of the update $\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$ in Equation (7), $\partial \mathbf{c}_T / \partial \mathbf{c}_t$ has two gradient paths: one through each term in the sum. The first path is only linear in the cell state. This has led to claims that the LSTM avoids vanishing gradients by allowing information to flow along cell states over long time intervals. However, further analysis reveals that the second path depends on the entries of the weight matrices $\mathbf{W}_{gh}, \mathbf{W}_{fh}, \mathbf{W}_{ih}, \mathbf{W}_{oh}$, exponentiating their magnitude to power $T - t$ (Kanuparthi et al., 2019). The magnitude of the second gradient path can suppress the first so that weight updates are made in favor of short-term dependencies.