GAUSSIAN-GATED LSTM: IMPROVED CONVERGENCE BY REDUCING STATE UPDATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Recurrent neural networks can be difficult to train on long sequence data due to the well-known vanishing gradient problem. Some architectures incorporate methods to reduce RNN state updates, therefore allowing the network to preserve memory over long temporal intervals. To address these problems of convergence, this paper proposes a timing-gated LSTM RNN model, called the Gaussian-gated LSTM (g-LSTM). The time gate controls when a neuron can be updated during training, enabling longer memory persistence and better error-gradient flow. This model captures long-temporal dependencies better than an LSTM and the time gate parameters can be learned even from non-optimal initialization values. Because the time gate limits the updates of the neuron state, the number of computes needed for the network update is also reduced. By adding a computational budget term to the training loss, we can obtain a network which further reduces the number of computes by at least $10\times$. Finally, by employing a temporal curriculum learning schedule for the g-LSTM, we can reduce the convergence time of the equivalent LSTM network on long sequences.

1 INTRODUCTION

Numerous methods and architectures have been proposed to mitigate the vanishing gradient problem in RNNs, with LSTMs (Hochreiter & Schmidhuber, 1997) as one of the first prominent solutions doing so by including gating structures in the computation. Although the LSTM has excelled at handling many tasks (Schmidhuber, 2015; Lipton, 2015), it still has trouble learning complex and long time dependencies (Neil et al., 2016; Chang et al., 2017; Trinh et al., 2018).

In the last few years, various methods which reduce the state updates of an RNN (LSTM) have been explored to better learn long time dependencies from data. Clockwork RNNs (Koutnik et al., 2014) group the hidden units of the RNN into "modules," where each module is executed at pre-specified time steps thereby skipping time steps which helps learn longer time dependencies. Recently, various other methods have been proposed which can be characterized by the use of additional "time gates," k_n, that control the information flow from one time step to the next (Krueger et al., 2017; Campos et al., 2017). Phased LSTM (PLSTM) (Neil et al., 2016) learns a parameterized function, k_n, from the the time input of the current state, and was proven to be succesful at learning over long sequences.

The PLSTM time gate, parameterized by period, phase, and ratio parameters for each hidden unit, is defined through a modulo function with an ill defined gradient. Furthermore, with periodic functions being hard to learn using gradient based methods (Shamir, 2016), and with k_n being periodic, the PLSTM was unable to learn the time gate parameters and hence relied on careful initialization. In order to offset these difficulties, this work proposes a new LSTM variant called the gaussian-gated LSTM (g-LSTM). Similar to the PLSTM, it is an LSTM model with a parameterized k_n but with only two parameters per hidden unit. Unlike the PLSTM which uses a periodic formulation for k_n, the g-LSTM uses a Gaussian function.

The g-LSTM provides a number of advantages over the LSTM model:

- The g-LSTM network can process very long sequences by reducing the time over which the neurons can be updated. It converges faster than the LSTM, especially on sequences that are over 1000 steps.

- The "openness" of the neuron for an update can be adapted according to the task during training.
- By introducing a computational budget term into the loss function during training, the "openness" of the neuron can be optimized for a reduced computational budget. This reduction can be achieved with little or no degradation to the network performance.
- A "temporal curriculum" training schedule can be set up for the g-LSTM so that it helps to speed up the convergence of a normal LSTM RNN.

The paper is structured as follows: In section 2, we discuss briefly the related work. Then, in section 3, we present the formulation of the g-LSTM, the datasets used in this work and details about the experimental hyperparameters. In section 4, we present experiments substantiating the four specific claims made above regarding the advantages of the network. Finally in section 5, we conclude with a brief discussion of the results.

2 Related Work

There have been a multitude of proposed methods to improve the training of RNNs, especially for long sequences. Apart from incorporating additional gating structures, for e.g., the LSTM and the GRU (Cho et al., 2014), more recently various techniques were proposed to further increase the capabilities of recurrent networks to learn on sequences of length over 1000. Proposed initialization techniques such as the orthogonal initialization of kernel matrices (Cooijmans et al., 2016), and diagonal recurrent kernel matrices (*e.g.* Li et al. (2018)) have demonstrated success. Trinh et al. (2018) proposes using truncated backpropagation with an additional auxiliary loss to reconstruct previous events.

Alternately, methods based on skipping state updates were proposed, first explored by Koutnik et al. (2014) in the Clockwork RNN (CW-RNN). This network skips state updates by allowing different neurons to be "activated" on different, modulated clock cycles, Phased LSTM (PLSTM) (Neil et al., 2016) uses a modulo-periodic timing gate to limit state updates. The effectiveness of skipping state updates in a random manner was shown with the Zoneout network (Krueger et al., 2017). Skip RNN (Campos et al., 2017) learns a state skipping scheme from the data to shorten the effective sequence length in the computation. LSTM-Jump (Yu et al., 2017) uses a reinforcement learning algorithm to learn when to skip state updates, showing a method to more quickly process (long) sequential data with an RNN while maintaining an accuracy comparable to a baseline LSTM.

By preserving memory information over large timescales it has been shown that these solutions enable more efficient learning at large time scales (Koutnik et al., 2014). It has been suggested but not yet demonstrated in the literature that the parameters of the CW-RNN clock cycle and PLSTM timing gate could be learned in training. Currently, the implementation of these networks requires a careful initialization of these parameters. With the Gaussian-gated LSTM (g-LSTM) in this work we present a timing gated RNN network that is demonstrated to efficiently solve long sequence tasks while having the capability of learning its timing gate parameters during training.

3 Methods

3.1 g-LSTM

The g-LSTM is an LSTM model with an additional time gate (Fig. 1). This time gate is used to regulate the information flow in time. Equations 1 - 3 describe the update equations for the hidden and cell states of the LSTM. Equations 4 and 5 describe the gating mechanism of the time gate, k_n.

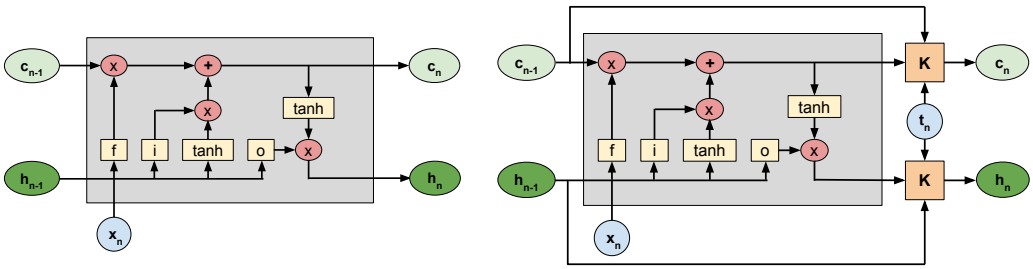
$$i_n = \sigma(x_n W_{xi} + h_{n-1} W_{hi} + b_i), f_n = \sigma(x_n W_{xf} + h_{n-1} W_{hf} + b_f) \tag{1}$$

$$\tilde{c}_n = f_n \odot c_{n-1} + i_n \odot \sigma(x_n W_{xg} + h_{n-1} W_{hg} + b_g) \tag{2}$$

$$o_n = \sigma(x_n W_{xo} + h_{n-1} W_{ho} + b_o), \tilde{h}_n = o_n \odot \tanh(\tilde{c}_n) \tag{3}$$

$$c_n = k_n \odot \tilde{c}_n + (1 - k_n) \odot c_{n-1} \tag{4}$$

$$h_n = k_n \odot \tilde{h}_n + (1 - k_n) \odot h_{n-1} \tag{5}$$



(a) LSTM (b) g-LSTM

Figure 1: Comparison of the LSTM and g-LSTM models. k is the computational block for the time gate in (b).

In a standard LSTM, the gating functions i_n, f_n, o_n, represent the *input*, *forget*, and *output* gates respectively at sequence index n. c_n is the cell activation vector, whereas x_n and h_n represent the input feature vector and the hidden output vector respectively. The cell state c_n itself is updated with a fraction of the previous cell state that is controlled by f_n, and a new input state created from the element-wise (Hadamard) product, denoted by $\odot$, of i_n and the candidate cell activation as in Eq. 2.

In the g-LSTM, we further control the cell state and the output hidden state through the k_n gate which is independent of the input data and hidden states, and is purely dependent on the time input corresponding to the sequence index n. The use of the Hadamard product ensures that each hidden unit is independently controlled by the corresponding time gate unit, thus enabling the different units in the layer to process the input at different time scales.

The time gate k_n is defined based on a Gaussian function as:

$$k_n = e^{-(t_n-\mu)^2/\sigma^2} \tag{6}$$

where the mean parameter, μ, defines the time when the hidden unit is "open" and the standard deviation, σ, controls the openness of the time gate for each unit around its corresponding μ. The time inputs $\mathbf{t} = \{t_1, t_2, ..., t_n, ..., t_N\}$ for the sequence $\mathbf{x} = \{x_1, x_2, ..., x_n, ..., x_N\}$ can correspond to the physical notion of time at the respective sequence input. In the absence of a standard notion of time, we use the sequence indices as the time input, i.e. $\mathbf{t} = \{1, 2, ..., n, ..., N\}$. In this work, we assume this notion of time by default.

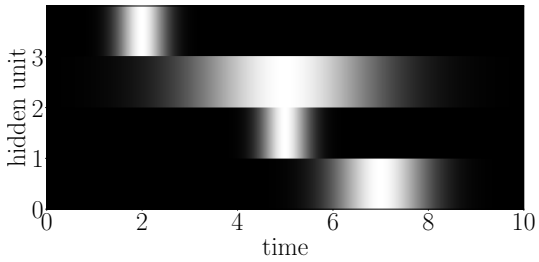


Figure 2: Time gate functions for four different hidden units on a sequence length of 10. From top to bottom, k_n is shown for (μ, σ) parameter values of $(2, 0.5)$, $(5, 2)$, $(5, 0.5)$, and $(7, 1)$.

The "openness" of k_n as defined by the different parameterizations of the Gaussian function can be seen in Fig. 2. Here, k_n is plotted as a function of time (x-axis) with black values corresponding to a fully closed gate (value 0) and white values corresponding to a fully open gate (value 1). Note that lower values of σ ensure that the unit is processed only if the time input is μ, while higher σ values lead to the unit processed like in the standard LSTM, at all times.

3.2 BACK PROPAGATION FOR G-LSTM

An important characteristic of the g-LSTM is reduced gradient flow in back propagation training methods. By having the gating structure as in Eqs. 4 and 5 there are fewer gradient product terms, which reduces the likelihood of vanishing or exploding gradients.

In a gradient descent learning scheme for a given loss function, L, when training the recurrent parameters, W_h (from Eqs. 1 - 3), the gradient as in Eq. 7 is used.

$$\frac{\partial L}{\partial W_h} = \frac{\partial L}{\partial h_N} \frac{\partial h_N}{\partial W_h} \tag{7}$$

By the chain rule $\frac{\partial h_N}{\partial W_h}$ expands for all time steps of the sequence, $n \in \{1, ..., N\}$. Because each output state is gated by the time gate, k_n, the gradient terms in the expansion of $\frac{\partial h_N}{\partial W_h}$ are scaled by k_n. When the time gate is open less often, *i.e.* with a small σ value, then there are fewer influential gradient terms. More details are given in Appendix A.

3.3 DATASETS

To demonstrate the advantages of the g-LSTM, the experiments described in the paper are carried out on the adding task and two standard long sequence datasets: the sequential MNIST and the sequential CIFAR-10 datasets.

Adding task: In order to test the long sequence learning capability of the g-LSTM, we use the adding task (Hochreiter & Schmidhuber, 1997). In this task, the network is presented with two sequences of length N, $\mathbf{x} = \{x_1, ..., x_N\}$, $x_n \sim U(0, 1))$ and $\mathbf{m} = \{m_1, ..., m_N\}, m_n \in \{0, 1\}, \sum_{t=1}^{N} m_n = 2$. The sequence $\mathbf{m}$ has exactly two values of 1 and the remaining values of the sequence are 0. The indices of the "1" values are chosen at random. For each pair in the sequence $(\mathbf{x}, \mathbf{m})$, the associated label value, y, is the sum of the two values in $\mathbf{x}$ corresponding to the "1" values of $\mathbf{m}$. The objective of this task is to minimize the mean squared error between the predicted sum from the network, $\hat{y}$, and the labeled sum, y. A new training set of 5000 sequence samples is presented in every epoch during training in order to avoid overfitting. The test set consists of further 5000 samples. For $N > 1000$, it is known that LSTMs have difficulty learning the task and hence we focus on values of $N > 1000$ in this work.

sMNIST: The sequential MNIST dataset is widely used to analyze the performance of a recurrent model. This dataset consists of 60,000 training samples and 10,000 test samples, each a single vector sequence of length 784 corresponding to the 28×28 pixel images in the MNIST dataset (LeCun et al., 1998). We also use permuted MNIST (pMNIST), a permuted variant of the sMNIST dataset where the sequences are processed with a fixed random permutation, making the task harder.

sCIFAR-10: The sequential CIFAR-10 dataset is another long sequence dataset based on CIFAR-10 (Krizhevsky et al., 2009) with 10 classes. The 32×32 RGB pixel images are reshaped into sequences of length 1024 with 3 dimensional features corresponding the RGB channels at every time step. Like in the sMNIST dataset, the dataset consists of 60,000 training samples and 10,000 test samples.

3.4 EXPERIMENTAL HYPERPARAMETERS

For the adding task, a mean squared error (MSE) loss was used with the Adam optimizer (Kingma & Ba, 2014) with a learning rate of 10^{-3}. The g-LSTM time gate parameters were trained using a learning rate of 10^0. For both sMNIST and sCIFAR-10 datasets, the cross-entropy loss function was used along with the RMSProp optimizer (Tieleman & Hinton, 2012) with a learning rate of 10^{-3}. Decay parameters of 0.5 and 0.9 were used for sMNIST and sCIFAR-10, respectively.

4 RESULTS

This section presents the experimental results that demonstrate the advantages of g-LSTM. Section 4.1 presents results that demonstrate the faster convergence properties of the g-LSTM over the

LSTM. Section 4.2 shows the trainability of the time gate parameters of the g-LSTM even when the parameters are initialized in an arbitrary way. Section 4.3 presents a modified loss function used during training to reduce the number of computes for the network update and Section 4.4 presents a new "temporal curriculum" learning schedule that allows g-LSTMs to help LSTMs converge faster.

4.1 FAST CONVERGENCE PROPERTIES OF G-LSTM

First, we look at the convergence properties of the g-LSTM, on the long-sequence adding task, the sMNIST task and the sCIFAR-10 task. Table 1 details the architectures used for networks in the experiments in this section. Similar to the architecture from Trinh et al. (2018), the recurrent layer of the sCIFAR-10 network is followed by two 256 unit fully-connected (FC) layers, where DropConnect (Wan et al. (2013)) ($p = 0.5$) is applied to the second FC layer. Also the kernel matrices in the LSTM networks were initialized in an orthogonal manner as described in (Cooijmans et al., 2016), while the g-LSTM network kernels required no such specific initialization.

		Initialization		Performance	
Dataset	# units	μ	σ	g-LSTM	LSTM
Adding (N=1000)	110	$\sim U(300, 700)$	40	$3.8 \cdot 10^{-5}$	$1.4 \cdot 10^{-3}$
Adding (N=2000)	110	$\sim U(500, 1500)$	40	$1.3 \cdot 10^{-3}$	$1.6 \cdot 10^{-1}$
sMNIST	110	$\sim U(1, 784)$	250	1.3%	1.8%
sCIFAR-10	128	$\sim U(1, 1024)$	650	41.1%	41.8%

Table 1: Network architectures and performance for the convergence experiments in subsection 4.1. The performance metric is the final mean squared error (MSE) loss for the adding task, and the label error rate for both sMNIST and sCIFAR-10.

The test performances of these networks during the course of the training on different datasets are shown in Figure 3, while the corresponding final performance metrics at the end of training are shown in Table 1. From Figure 3, it is evident that the test loss of the g-LSTM decreases faster in training than the LSTM across all datasets.

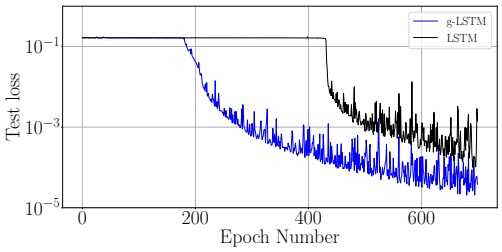
Table 2 compares the performance of various networks including the g-LSTM and the baseline LSTM on sMNIST and sCIFAR-10 (from Table 1). The results show that the g-LSTM consistently performs better than the LSTM and has a similar performance to other state-of-the-art networks.

Network	sMNIST	pMNIST	sCIFAR-10
LSTM (this work)	1.8%	8.4%	41.8%
g-LSTM	1.3%	7.5%	41.1%
r-LSTM (Trinh et al., 2018)	1.6%	4.8%	27.8%
Zoneout (Krueger et al., 2017)	1.3%	6.9%	-
IndRNN (6 layers) (Li et al., 2018)	1.0%	4.0%	-
BN-LSTM (Cooijmans et al., 2016)	1.0%	4.6%	-
Skip LSTM (Campos et al., 2017)	2.7%	-	-

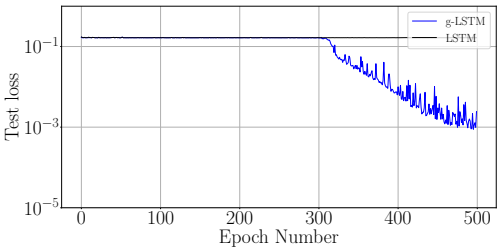
Table 2: Comparison of label error rates across different networks.

4.2 TRAINABILITY OF THE TIME GATE PARAMETERS OF G-LSTM

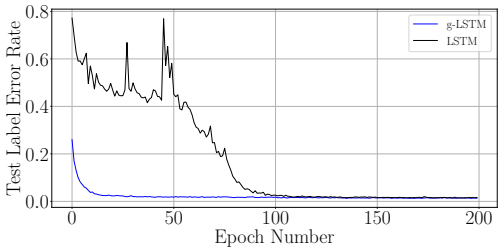
As discussed in section 2, one of the main advantages of the g-LSTM over the PLSTM is that the time gate parameters can be trainable. In order to demonstrate this, we look at the performance of the g-LSTM network with different time gate parameter initializations. We concern ourselves with sequences of length 1000 for the adding task in this section. The time gate parameters are initialized in a way to temporally constrain the network so that it can only process for a short period of time. For example, a network with time gate parameters initialized with $\mu = 500$ and $\sigma = 40$ as in Figure 4(a) can only process a short period of time around the middle of sequence. It follows



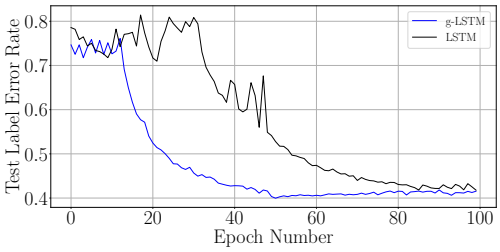
(a) adding task ($N = 1000$)



(b) adding task ($N = 2000$)



(c) sMNIST

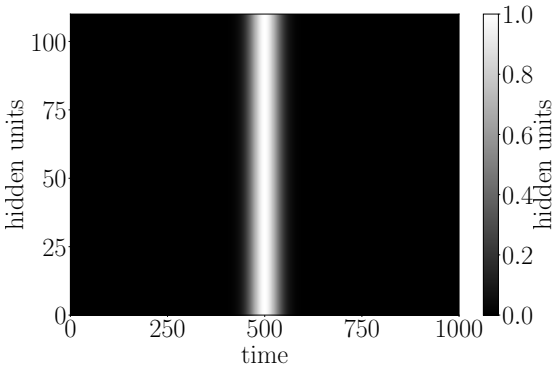


(d) sCIFAR-10

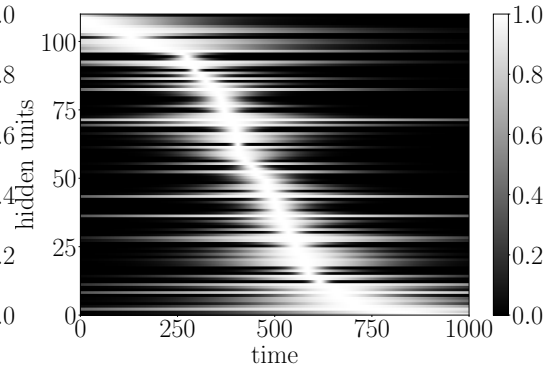
Figure 3: Test loss and test label error rates across training epochs for LSTM (black) and g-LSTM (blue) networks on different tasks.

that the network would be unable to learn with these parameters because in the adding task the input data is distributed equally across the sequence length ($T = 1000$). Therefore, in order to learn the task from this initialization, the time gate parameters must learn a distribution such that the gates over all hidden units are open across the entirety of the sequence.

We observe that the time gate parameters do learn, as shown in Fig. 4 (b), and thereby enable the network to solve the task. Independent of various time gate initializations, the network reaches a MSE of around 3.9×10^{-5} at the end of 700 epochs; details of which could be found in Appendix B. The ability of the network to learn the time gate parameters necessary to cover the entire sequence is especially significant because it shows that even with this narrow time window initialization that requires learning of the time gates, the g-LSTM learns the task, whereas the PLSTM does not learn the task as well. An example of this is shown in Appendix C.



(a) Pre-training



(b) Post-training

Figure 4: Time gate behavior pre and post training, demonstrating the ability of the network to learn from extreme initialization parameters.

4.3 REDUCTION IN COMPUTATION

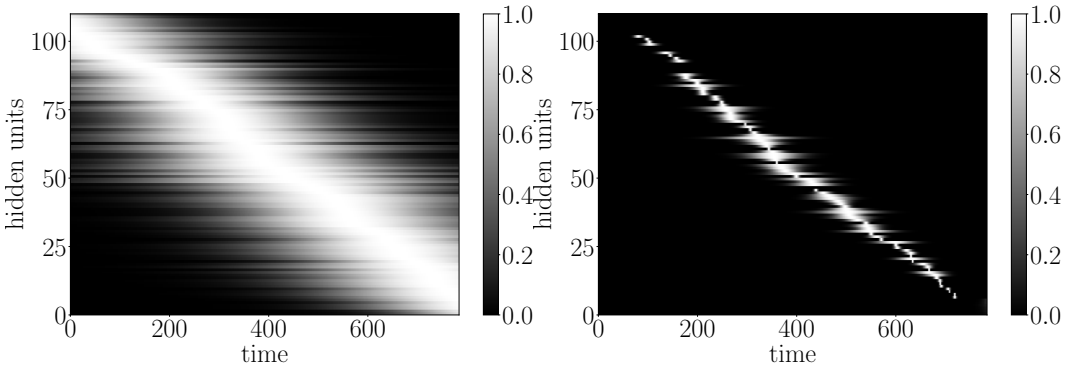
Although the formulation of the g-LSTM appears to require more updates, it offers substantial speedup as a large proportion of the neurons can be skipped in a timestep at runtime. We can set a threshold on the time gate so that we skip all corresponding computations for time steps where k_n is below this threshold. To further reduce the number of operations, it is preferred that the σ of the k_n for different neurons should be small but the network performance should not be significantly degraded. To achieve this goal, we included a "computational budget" loss term during the optimization of the gate parameters, μ and σ. The loss equation for updating the k_n parameters is given by:

$$L = L_{data} + \lambda L_{budget}.$$

Similar to the Skip RNN network (Campos et al., 2017), a budget loss term which minimizes the average openness of the time gate over time is applied:

$$L_{budget} = \lambda \mathbb{E}[k_n] \approx \lambda \sum_{n=1}^{N} \sum_{j=1}^{J} k_n^{(j)}$$

for every neuron j of the g-LSTM. The study was carried out on sMNIST using a network with 110 units, σ initialized to 50, μ initialized uniformly at random between 1 to 784, and a λ value of 1. The network's performance of 97.8% accuracy was comparable to the network's performance of 98.7% when no additional budget constraint was imposed. The final σ range for the budgeted g-LSTM is much smaller compared to that of the g-LSTM as shown in Fig. 5. There is only a small drop in accuracy for the budgeted g-LSTM versus the g-LSTM (see Table 2), even though there is a significant decrease in the average time gate openness across all hidden units.



(a) g-LSTM (b) budgeted g-LSTM

Figure 5: Time gate behavior of (a) g-LSTM and (b) budgeted g-LSTM for 110 units post training.

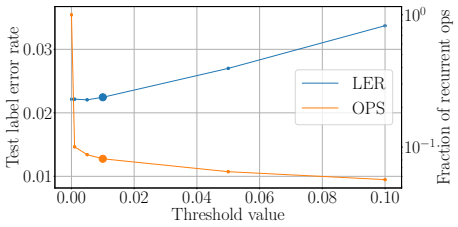


Figure 6: Reduction in computes as a function of threshold for budgeted g-LSTM.

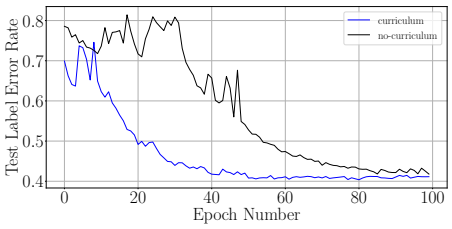


Figure 7: Speed up in convergence of LSTM using the temporal curriculum learning schedule.

In order to reduce the number of computes, we set a threshold, v_T for k_n so that the update steps are carried out only if $k_n > v_T$. By increasing v_T, the number of computes decreases as shown in Fig. 6. In the case of $v_T = 0.01$, only 8.2% of the time gates are open on average across all

hidden units and the 784 time steps. Furthermore, the accuracy dropped only slightly to 97.7% from 97.8%. For the remaining time and units, the LSTM update equations can be "skipped" over and the previous neuron state can be copied over as the current state.

We give a quantitative estimate for the number of operations (Ops) corresponding to the number of update equations for a g-LSTM. In the estimate, we count a multiply and an add operation as 1 Op and non-linear functions as 5 Ops. For an LSTM, the number of operations is given by

$$N_{LSTM} = T\,H(8D + 8H + 29)$$

where T is the number of time steps, H is the number of hidden units, and D is the dimension of the input data. For a g-LSTM, the number of operations is given by

$$N_{g-LSTM} = N_{LSTM} + N_{gate} + N_{gauss}$$

where $N_{gate} = 4\,T\,H$ is the number of operations for computing the time gate following Eqs. 4 and 5, and $N_{gauss} = 9\,T\,H$ is the number of operations for computing the Gaussian function of the time gate. When $v_T = 0$, the total number of operations for the g-LSTM network on the sMNIST dataset is around 80 MOps for $N = 110$ and $T = 784$, whereas this number is reduced to 7.6 MOps for the budgeted g-LSTM with $v_T = 0.01$.

4.4 Temporal curriculum training schedule for LSTMs

We demonstrate that it is possible to train a LSTM network by using a "temporal curriculum" training schedule for the equivalent g-LSTM network. According to this schedule, the initial σ values of the g-LSTM network are increased continuously throughout the training period ending up with high values by the end of training. With such high values, the time gates are essentially open, resulting in an LSTM network. At every training epoch, the lowest $\rho\%$ of the σ values, $\hat{\boldsymbol{\sigma}}$ in the layer are updated as,

$$\hat{\boldsymbol{\sigma}} \longrightarrow (1 + \alpha) \cdot \hat{\boldsymbol{\sigma}} \tag{8}$$

We analyze the impact of this training schedule for training an LSTM network on sCIFAR-10. The equivalent g-LSTM network with 110 units was considered with μ initialized uniformly at random between 1 and 1024 and σ initialized to 50. An α value of 1/6 and ρ value of 15% was chosen. To ensure that the time gate is fully open by the end of training, σ is set to 5000 across all units during the last 10 epochs of training. The learning rate of the time gate parameters is set to 0, *i.e.* μ and σ are no longer updated. Figure 7 shows that the temporal curriculum training schedule allows for faster convergence of an LSTM network. The final weights of the trained g-LSTM network can then be copied over to a LSTM network for inference.

5 Conclusion

This work proposes a novel RNN variant with a time gate which is parameterized by the input in time. The model uses a time gate similar to the PLSTM, but unlike the periodic function for the PLSTM, the g-LSTM uses a Gaussian function which allows better training properties on long input sequences. The g-LSTM network shows faster convergence and produce higher accuracies than LSTM networks, as demonstrated for the adding task sequences which are longer than 1000 timesteps, and on the sMNIST and sCIFAR-10 datasets.

We further demonstrate that the time gate parameters of the g-LSTM (unlike those of the PLSTM) are learnable even when the time gates are initialized in an extreme manner for the adding task. The time gate of the g-LSTM can reduce the number of computes that is needed for the updates of the LSTM equations and with an additional loss term to reduce the compute budget, the σ values of the time gate are reduced leading to a $10\times$ decrease in number of actual computes and little loss in network accuracy, for the SMNIST dataset. We also show that our proposed temporal curriculum training schedule for the g-LSTM can help a corresponding LSTM network to converge on training for long sequences.

REFERENCES

Victor Campos, Brendan Jou, Xavier Giro i Nieto, Jordi Torres, and Shih-Fu Chang. Skip RNN: learning to skip state updates in recurrent neural networks. *CoRR*, abs/1708.06834, 2017. URL http://arxiv.org/abs/1708.06834.

Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael J. Witbrock, Mark Hasegawa-Johnson, and Thomas S. Huang. Dilated recurrent neural networks. *CoRR*, abs/1710.02224, 2017. URL http://arxiv.org/abs/1710.02224.

Kyunghyun Cho, Bart van Merriënboer, Çalar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN Encoder–Decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. URL http://www.aclweb.org/anthology/D14-1179.

Tim Cooijmans, Nicolas Ballas, Cesar Laurent, and Aaron C. Courville. Recurrent batch normalization. *CoRR*, abs/1603.09025, 2016. URL http://arxiv.org/abs/1603.09025.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Jan Koutnik, Klaus Greff, Faustino J. Gomez, and Jürgen Schmidhuber. A clockwork RNN. *CoRR*, abs/1402.3511, 2014. URL http://arxiv.org/abs/1402.3511.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10 (Canadian Institute for Advanced Research). 2009. URL http://www.cs.toronto.edu/~kriz/cifar.html.

David Krueger, Tegan Maharaj, Janos Kramar, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Aaron Courville, and Christopher Pal. Zoneout: Regularizing RNNs by randomly preserving hidden activations. 2017. URL https://openreview.net/forum?id=rJqBEPcxe.

Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791.

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, 2018.

Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased LSTM: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pp. 3882–3890, 2016.

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015. doi: 10.1016/j.neunet.2014.09.003. Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

Ohad Shamir. Distribution-specific hardness of learning neural networks. *CoRR*, abs/1609.01037, 2016. URL http://arxiv.org/abs/1609.01037.

Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

Trieu H. Trinh, Andrew M. Dai, Thang Luong, and Quoc V. Le. Learning longer-term dependencies in RNNs with auxiliary losses. *CoRR*, abs/1803.00144, 2018. URL http://arxiv.org/abs/1803.00144.

Li Wan, Matthew Zeiler, Sixin Zhang, Yann LeCun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David McAllester (eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pp. 1058–1066, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL http://proceedings.mlr.press/v28/wan13.html.

Adams Wei Yu, Hongrae Lee, and Quoc V. Le. Learning to skim text. *CoRR*, abs/1704.06877, 2017. URL http://arxiv.org/abs/1704.06877.

A BACK PROPAGATION IN GAUSSIAN-GATED RNN

For ease of illustration we analyze the gradient of a plain RNN with a Gaussian time gate (Eqs. 9 and 10).

$$h_n = k_n \cdot \tilde{h}_n + (1 - k_n) \cdot h_{n-1} \tag{9}$$

$$\tilde{h}_n = f(W_x x_n + W_h h_{n-1}) \tag{10}$$

$$\frac{\partial h_N}{\partial W_h} = \frac{\partial \tilde{h}_0}{\partial W_h} \prod_{n=1}^{N} (k_n W_h f'_n + (1 - k_n)) + \sum_{n=1}^{N} (k_n f'_n h_{n-1}) \prod_{s=n+1}^{N} (k_s W_h f'_s + (1 - k_s)) \tag{11}$$

$$\text{where } \tfrac{\partial \tilde{h}_0}{\partial W_h} = 1, h_0 = \tilde{h}_0 = 0.$$

From Eq. 11 we can deduce some information about the advantages of the Gaussian time gate in gradient flow for two simple cases of the function k_n.

In Case 1 we choose a timing gate openness which corresponds to a very small σ for the Gaussian gate, *i.e.* the gate is only open for 1 time step.

$$k_5 = 1, k_n = 0 \quad \forall \quad n \in \{1, ..., N\} \backslash \{5\}$$

$$\frac{\partial h_N}{\partial W_h} = f'_5 + f'_5 f'_5 h_4 = f'_5 W_h \left(1 + f'_5 h_4\right)$$

In Case 2 we choose a timing gate openness which corresponds to a slightly larger σ for the Gaussian gate, *i.e.* it is open for 5 time steps.

$$k_2 = 1, k_3 = 1, k_4 = 1, k_5 = 1, k_6 = 1, k_n = 0 \quad \forall \quad n \in \{1, ..., N\} \backslash \{2, 3, 4, 5, 6\}$$

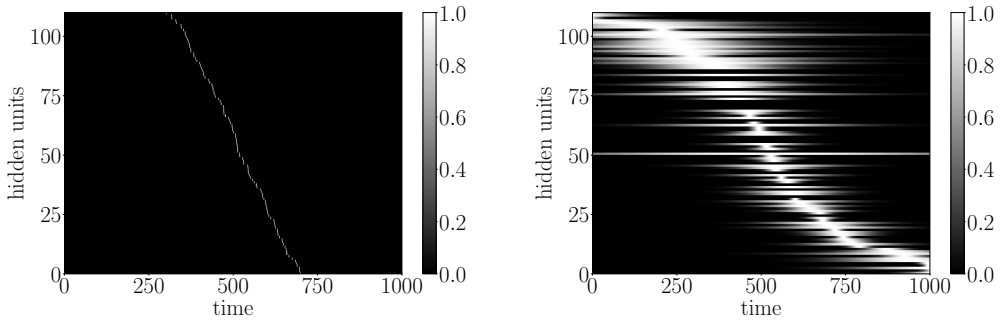
$$\frac{\partial h_N}{\partial W_h} = f'_2 W_h f'_3 W_h f'_4 W_h f'_5 W_h f'_6 W_h \left(1 + f'_2 h_1 + f'_3 h_2 + f'_4 h_3 + f'_5 h_4 + f'_6 h_5\right)$$

These cases show that there are fewer terms in the gradient for a timing gate that is opened for only a small percentage of the sequence.

B COMPARING VARIOUS G-LSTM INITIALIZATIONS

	Initialization		
Experiment ID	μ	σ	Final MSE Loss
A1	$\sim U(300, 700)$	1	$4.4 \cdot 10^{-4}$
A2	$\sim U(0, 400)$	40	$2.0 \cdot 10^{-5}$
A3	$\sim U(600, 1000)$	40	$4.0 \cdot 10^{-4}$

Table 3: Adding task (T=1000): 110 unit g-LSTM network initializations and performances.



(a) Experiment A1: before training

(b) Experiment A1: after training

(c) Experiment A2: before training

(d) Experiment A2: after training

(e) Experiment A3: before training

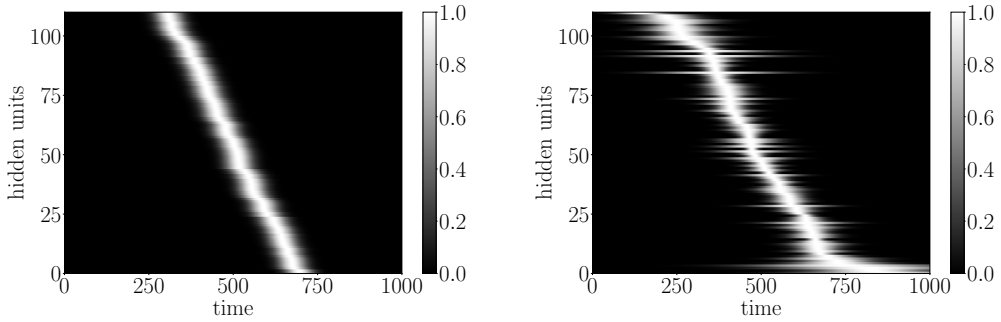
(f) Experiment A3: after training

Figure 8: Additional experiments: timing gate openness for non-optimal initializations

C COMPARING TIME GATE PARAMETER TRAINABILITY IN G-LSTM AND PLSTM

Network	Initialization	Final MSE Loss
g-LSTM	$\mu \sim U(300, 700)$, $\sigma = 40$	$7.7 \cdot 10^{-5}$
PLSTM	$\tau = 1000$, $s \sim U(250, 650)$, $r = 0.10$	$2.4 \cdot 10^{-4}$

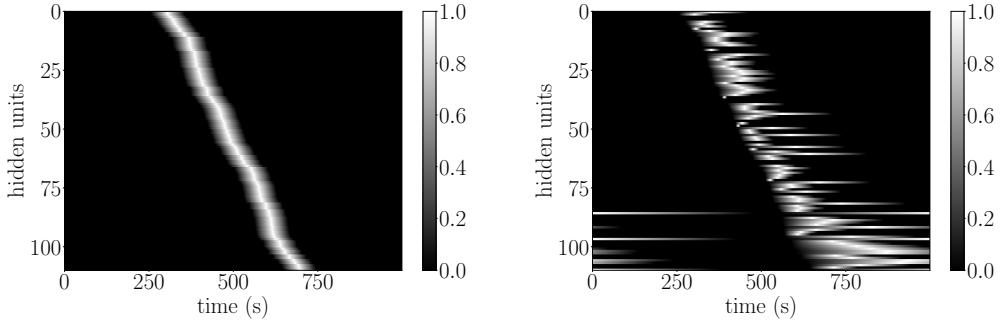
Table 4: Adding task (T=1000): Comparing similar initializations of 110 unit g-LSTM and PLSTM network, trained for 500 epochs.



(a) before training

(b) after training (500 epochs)

Figure 9: g-LSTM



(a) before training

(b) after training (500 epochs)

Figure 10: PLSTM