Counting in Language with RNNs

Heng xin Fun
University of Lugano
funh@usi.ch

Sergiy V Bokhnyak
University of Lugano
bokhns@usi.ch

Francesco Saverio Zuppichini
University of Lugano
francesco.saverio.zuppichini@usi.ch

Abstract

In this paper we examine a possible reason for the LSTM outperforming the GRU on language modeling and more specifically machine translation. We hypothesize that this has to do with counting. This is a consistent theme across the literature of long term dependence, counting, and language modeling for RNNs. Using the simplified forms of language – Context-Free and Context-Sensitive Languages – we show how exactly the LSTM performs its counting based on their cell states during inference and why the GRU cannot perform as well.

1 Introduction and Related Work

The LSTM [Hochreiter and Schmidhuber, 1997], [Gers et al., 2000], and the GRU [Cho et al., 2014][Chung et al., 2014] are two of the most popular RNN architectures used for language modeling and machine translations with deep learning. They lend well to the task because of their ability to propagate errors back through time for much longer time steps than the original RNN architectures due to their gating mechanisms [S. Hochreiter, 1998], [Hoc]. Thus for tasks that have sequential data with long term dependencies on the inputs between timesteps, the GRU and LSTM become natural choices.

Language modeling and machine translation is an obvious example of this type of problem. Subjects and predicate phrases can vary in length drastically. Furthermore adjective and prepositional phrases can take noun and verb phrases as objects much later in the text. Long term dependencies in the inputs require the RNN to have a persistent state that keeps track of information it has seen and be able to access this memory at much later time steps. [Linzen et al., 2016] showed that the LSTM cell was indeed keeping track of long range syntactically sensitive dependencies, while [Gulordava et al., 2018] demonstrated that the LSTM cell was able to store information about long-distance syntactic agreements much better than the Simple RNN. In regards to NMT systems, [Belinkov et al., 2018] showed that the RNNs in the higher layers of the NMT models were specializing in more abstract tasks of semantic tagging while the lower layers were storing and tracking part of speech tags.

The most prominent application of these RNNs is in the field of machine translation. There has been many studies recently investigating what kinds of architecture and hyperparameter combinations perform best for these tasks. The findings of Britz et al.[2017], Jozefowicz et al.[2015] generally indicate LSTM outperforming the GRU in machine translation and language modeling tasks. This naturally begs the question, why exactly does the LSTM generally outperform the GRU on these tasks. Further research showed findings on the unimportance of the forget gate. [Jozefowicz et al., 2015] say, “We discovered that the input gate is important, that the output gate is unimportant, and that the forget gate is extremely significant on all problems except language modelling. This is consistent with Mikolov et al. (2014), who showed that a standard RNN with a hard-coded integrator unit (similar
to an LSTM without a forget gate) can match the LSTM on language modelling” [Tomas Mikolov 2015].

\[ c^t = f^{t-1} \odot c^{t-1} + i^t \odot z^t \]  

(1)

The forget gate erases or down scales the values of the cell state of the LSTM. All of the papers above agree on the tremendous importance of the input gate and on the unimportance of the forget gate. Without the forget gate the LSTM has a persistent cell state that is only written to by the rightmost term of equation 1 whereas the absence of the forget gate ensures that the left term is simply the identity multiplied by the previous cell state.

Therefore there is a scenario of something that continuously adds (or removes) from a persistent state – a stack. We believe that this stack-like behavior is one of the main reasons for the LSTM outperforming the GRU on language modeling. This work was done concurrently with Weiss et al. [2017], which also showed LSTM outperforming GRU. One of the most basic forms of the stack, or push-down automata, is simply a structure that helps to count. Thus our hypothesis is that counting is one of the main tasks a model needs to be able to do, and to do well, in order to succeed in language modeling and machine translation tasks; or even if counting isn’t explicitly necessary for language modeling, the ability to count is a good indicator that a model will be able to solve language related tasks well. In previous work, Shi et al. [2016] showed that a LSTM in an encoder-decoder machine translation model is indeed using the cell state to track length in a full scale NMT WMT 2014 English to French task. Furthermore, Liu et al. [2018] showed that LSTM trained on natural language data such as Penn Treebank were able to count much longer sequences than on a modified version of Penn Treebank where the structure of the natural language was disturbed. They also saw that LSTM exploiting counting and memorization behavior to solve their task.

Some correlation between counting and language modeling has already been shown by Le, Jaitly, and Hinton [2015] who introduced the IRNN (RNNs with identity matrix initializations and ReLu activation) who were able to add sequences almost as well as the LSTM, and much better than regular RNNs. This task is clearly of the same form as we discussed above and thus is a form of counting. Further Le et al. [2015] show that the IRNN and the LSTM, who were able to count, easily outperform the RNNs which were shown unable to count.

Another study that showed similar correlation was again done by Joulin and Mikolov [2015]. They developed a Stack RNN – an RNN that is able to control a stack. Stack RNN and LSTM performed comparably on counting tasks such as CFL and CSL, better than RNN. And then applied to the Penn Treebank language modeling task, the Stack RNN was shown to perform equal as good as LSTM and superior to regular RNN.

2 Experimental Setup

The experiments we setup involve simple emulation of the pushdown automata, solving Context-Free Language. Examining the differences in how the LSTM and the GRU count will be helpful in understanding their differences in their ability to model language. The experiments were modeled after Gers and Schmidhuber [2001], and we also evaluated their performances on Context-Sensitive Language – a slightly harder version of counting that requires more than just a stack.

The models are trained on CFL $a^n b^n$ and CSL $a^n b^n c^n$ for $n \in \{1, ..., N\}$. The upper bound $N$ for CFL is $N = 10$ and for CSL $N = 40$. Training set is shuffled after epoch. We trained CFL for 15000 epochs across 20 seeds per model and for CSL 5000 epochs and across 10 seeds per model. For the optimizer we use gradient descent with momentum of 0.99 [Rumelhart et al., 1988]. We tried various learning rates $10^{-3}, 10^{-4}, 10^{-5}$. Greff et al. [2015] show that learning rates were important to the convergence of these models. We found that $10^{-3}$ worked best across all models and languages.

Once the model learns the training set with 100% accuracy, we begin to evaluate on unseen data $N_{new} = N_{old} + 1$, if the model is able to generalize to this higher $N$, we save this as a new max $N$ and continue incrementing $N$ until it fails. When it fails we continue training the model and test the model from $n$ up to previous max $N$ every 10 epochs.
2.1 Dataset

An example of CFL and CSL for \( n=2 \):

**CFL:** All strings \( a^n b^n \) with \( n \in \{1, \ldots, 10 \} \) For example, with \( n = 2 \), the targets are:

| Start: | S | a | a | b | b |
|--------|---|---|---|---|---|
| Target:| a | a/b| a/b| b | S |

**CSL:** All strings \( a^n b^n c^n \) with \( n \in \{1, \ldots, 40 \} \) For example, with \( n = 2 \), the targets are:

| Start: | S | a | a | b | b | c | c |
|--------|---|---|---|---|---|---|---|
| Target:| a | a/b| a/b| b | c | c | S |

2.2 Models and Parameters

The following table shows the models tested.

| Table 1: CFL | Model | Hidden Size | Layers | Trainable Parameters |
|-------------|-------|-------------|--------|----------------------|
| GRU2        | 2     | 1           |        | 36                   |
| GRU3        | 3     | 1           |        | 63                   |
| LSTM2       | 2     | 1           |        | 48                   |
| LSTMP2      | 2     | 1           |        | 54                   |

| Table 2: CSL | Model | Hidden Size | Layers | Trainable Parameters |
|-------------|-------|-------------|--------|----------------------|
| GRU4        | 4     | 2           |        | 216                  |
| GRU5        | 5     | 2           |        | 300                  |
| LSTM4       | 4     | 2           |        | 288                  |
| LSTMP4      | 4     | 2           |        | 312                  |

We select the hidden layers and cells number in order to be equivalent among models and specifically to not handicap the GRU since, a single cell, by construction, has fewer parameters than an equivalent LSTM cell. This is why we include the GRU configurations with slightly more hidden neurons than the LSTM, which have comparable trainable parameter sizes. Our comparisons mainly rely on comparing the best of the two GRU’s with the two LSTM’s for each problem.

2.3 Implementations

The implementations we used for this experiment were the TensorFlow implementations of **GRUCell** and **LSTMCell**, using the peephole implementation provided by TensorFlow [Sak et al., 2014] as part of the **LSTMCell**. The TensorFlow implementation of the LSTM with peephole is slightly different from the one used by [Gers and Schmidhuber, 2000], whose gates are of the form:

\[
g^i = \sigma(W_x^i + U_c^{i-1} + b)
\]

Whereas the TensorFlow implementation looks at the previous hidden state and has a mask on the previous cell state. Below are the LSTM equations with the peephole, but the regular LSTM equations

3
are identical excluding the $p \odot c^{t-1}$ term.

- **block input**: $z^t = g(W_zx^t + R_zh_{t-1} + b_z)$  
  \(3\)
- **input gate**: $i^t = \sigma(W_ix^t + R_ih_{t-1} + p_i \odot c^{t-1} + b_i)$  
  \(4\)
- **forget gate**: $f^t = \sigma(W_fx^t + R_fh_{t-1} + p_f \odot c^{t-1} + b_f)$  
  \(5\)
- **output gate**: $o^t = \sigma(W_ox^t + R_oh_{t-1} + p_o \odot c^{t-1} + p_{nc} \odot + b_o)$  
  \(6\)
- **cell state**: $c^t = z^t \odot i^t + c^{t-1} \odot f^{t-1}$  
  \(7\)
- **hidden state**: $h^t = g(c^t) \odot o^t$  
  \(8\)

And the **GRUCell** equations:

- **update gate**: $z^t = \sigma_g(W_zx^t + R_zh_{t-1} + b_z)$  
  \(10\)
- **reset gate**: $r^t = \sigma_g(W_rx^t + R_rh_{t-1} + b_r)$  
  \(11\)
- **hidden state**: $h^t = (1 - z_t) \odot h_{t-1} + z_t \odot \sigma_h(W_hx^t + R_h(r_t \odot h_{t-1}) + b_h)$  
  \(12\)

### 2.4 CFL Results

The training accuracy curves and generalization plots can be seen in the appendix in Figure section[5]. The following table shows the max $n$ for each model.

| Model                  | Min | Mean | Max  |
|------------------------|-----|------|------|
| LSTM peephole 1L 2H    | 0.0 | 18.0 | 433.0|
| GRU 1L 2H              | 0.0 | 5.2  | 40.0 |
| LSTM 1L 2H             | 0.0 | 19.25| 32.0 |
| GRU 1L 3H              | 0.0 | 10.55| 30.0 |

### 2.5 CSL Results

The training accuracy curves and generalization plots can be seen in the appendix in Figure section[5]. Below are the tables showing the maximum generalizations for CSL:

| Model                  | Min | Mean | Max  |
|------------------------|-----|------|------|
| LSTM peephole 2L 8H    | 0.0 | 54.4 | 936.0|
| LSTM peephole 2L 4H    | 0.0 | 14.14| 431.0|
| LSTM 2L 8H             | 48.0| 55.90| 90.0 |
| LSTM 2L 4H             | 42.0| 53.00| 65.0 |
| GRU 2L 8H              | 40.0| 44.11| 52.0 |
| GRU 2L 10H             | 0.0 | 40.50| 49.0 |
| GRU 2L 4H              | 0.0 | 12.57| 48.0 |
| GRU 2L 5H              | 0.0 | 30.71| 47.0 |

### 3 LSTMP solves CFL

2 out of 20 seeds of the LSTMP with 2 hidden units were able to generalize $n > 100000$ from only the training set of $n = 1..10$. We stopped testing the model after $n = 100100$. Below shows the cell state of this model. These models were excluded from the table and plots because of their outlier effects on the plots and averages.
3.1 Cell States during Counting

Here we will examine and compare the inner cell states of the most successful models of the LSTM and GRU. For the LSTM it is one of the peephole seeds, which generalized to strings with \( n > 10000 \) from only being trained one input of maximum length \( n = 10 \). We can safely conclude that this particular model has solved CFL (we actually have not seen it fail, it simply became too resource intensive to try longer and longer strings). The most successful GRU was one with 2 hidden units surprisingly (though on average the GRU3s did better) which was great because we did not need to do dimensionality reduction to plot the two dimensional cell state space. Each point is the cell state (or hidden state for GRU) at each time step, and the label next to each point represents the character which this cell state follows, thus A-1 represents the first A seen and the point represents the cell state after processing this input and writing to the cell state (cite equation).

![Figure 1](image1.png)

(a) Cell state of LSTMP2

(b) Hidden state of GRU2

Figure 1: With input string size \( n = 14 \), which both models solve

In [a] one can see that the step size of the cell state is constant, meaning when the input is an A, the cell state takes a step of approximately size 0.4 in the negative y-direction. Upon seeing the first B, the LSTM shifts the cell state to a different x-coordinate and takes one step in the positive y-direction. Then with every subsequent B observed it takes a step up until it gets to its start y-coordinate (close to zero) at which point it predicts the stop character S.

This is obviously not the case for the GRU hidden state seen in [b]. The GRU tries to implement a similar method of counting (this time upwards rather than down) with the same shift in x-coordinate once it sees the first B. However what is immediately apparent is the diminishing step size as it sees more and more As. And we can imagine that if our \( n \) grows large these steps would become negligible, thus diminishing how well the GRU is able to remember the count. This inability to make consistent steps must be the reason for the GRU’s failure to truly solve CFL, unlike the LSTM.

4 Decoupled GRU

We suspected – from the observations seen above – that the coupled gating of the GRU was the problem. The LSTM was able to make fairly consistent steps to the cell state each time it saw an A and reverse those steps once it began observing B’s. This requires that the LSTM did not forget its
previous cell state, and added to it. In theory, the cell state of the LSTM has no bound on the values it can remember, because it can continuously add to its cell state \(^7\). The GRU coupled update gate prevents this from being as easy. If we look at the way GRU writes to its next state, this should be obvious:

\[
h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t
\]  

(13)

Where \(z_t\) is the update gate and \(\tilde{h}_t\) is the candidate hidden state. From this equation it should be clear that in order for the GRU to add to its hidden state it must erase from it, thereby limiting its ability to count since if it learns a consistent step size \((z_t \odot \tilde{h}_t)\) its state will be scaled down by some multiplicative factor. Thereby for it to generalize to larger sizes of \(n\) it would have to learn not a constant step sizes but scaled step sizes, which makes the learning much more complicated. Thereby it seems that the regular GRU memory is bounded because of its update rule in a way that the LSTM is not. This is clearly seen in [1b] where we can see the smaller and smaller step sizes that the GRU must take in order to prevent erasing the count that it has saved in its cell state.

Thus we decided to run experiment with a modified GRU that has an input gate and forget gate instead of just update gate. The results can be seen in the appendix in Figure section [5] and compared to the regular GRU CFL results in Figure section [5].

This model of GRU is much more similar to the LSTM. And nevertheless it underperforms – it does about the same as the GRU2 on both generalization and fitting the training data. Though it still has differences in that the GRU cell state and hidden state are identical whereas in the LSTM the output gate separates them, as well as the reset gate on the previous hidden state in the GRU, which the regular LSTM does not have. In a way the output gate and the reset gate do the same function for the LSTM and GRU, respectively, across time steps – they are the gate applied to the previous state before calculating the current state. However, they may have a greater impact in the grand scheme of things, as shown by the results above. From the perspective of calculating the current state, beside the update rule of course, the biggest differences are:

1. The reset gate looks at the current input \(x_t\) whereas the output gate looks at the previous input \(x_{t-1}\). Therefore the state used in calculating the candidate state in the LSTM is independent of the current input. The state used in calculating the candidate state of the GRU is in some ways dependent on the current input (cite the GRU equation).

2. The fully connected layer that reads the outputs of the LSTM/GRU cells and produces the actual outputs, has direct access to the GRU hidden state \(h_t\) whereas the LSTM’s outputs are gated by the output gate. This relates to the different order of the output and reset gates discussed above, which may have an impact not just on how the cell’s state persists but also on what is actually being used in the calculation of the overall output of the network.

We conclude that it is one of these differences, or some combination of all of the differences that grant the advantage to the LSTM in solving CFL and other counting related tasks. These subtle differences could be investigated further to fully understand which parts of the LSTM cell are the most important to its success at counting and therefore language tasks.

5 Conclusion

As argued in the introduction, we believe there is a lot of evidence supporting the claim that success at language modeling requires an ability to count. Since there is empirical support for the fact that the LSTM outperforms the GRU in language related tasks, we believe that our results showing how fundamental this inability to count is for the GRU, we believe we make a contribution to the study of both RNNs and their success on language related tasks. Our experiments along with the other recent paper by Weiss et al. [2017], show almost beyond reasonable doubt that the GRU is not able to count as well as the LSTM, furthering our hypothesis that there is a correlation between success at performance on language related tasks and the ability to count.

We believe that this line of research could be useful in both understanding how language works ourselves, as well as improving our language models ability to understand and use human language. And there is much more work to be done in finding the exact correlation of these two tasks.
Figure 2: MAXN CFL small models
Figure 3: MAXN CSL medium size models
Figure 5: Decouple GRU CFL

(a) Large size models accuracy on CSL

(b) LSTM8 CSL

(c) LSTMP8 CSL

(d) GRU8 CSL

(e) GRU10 CSL
References

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

Felix A. Gers, Jürgen A. Schmidhuber, and Fred A. Cummins. Learning to forget: Continual prediction with lstm. *Neural Comput.*, 12(10):2451–2471, October 2000. ISSN 0899-7667. doi: 10.1162/089976600300015015. URL http://dx.doi.org/10.1162/089976600300015015

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

J. Schmidhuber S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 6(2):107–116, April 1998. ISSN 0218-4885. doi: 10.1142/S0218488598000094. URL http://dx.doi.org/10.1142/S0218488598000094

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of lstms to learn syntax-sensitive dependencies. *CoRR*, abs/1611.01368, 2016. URL http://arxiv.org/abs/1611.01368

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. Colorless green recurrent networks dream hierarchically. *CoRR*, abs/1803.11138, 2018. URL http://arxiv.org/abs/1803.11138

Yonatan Belinkov, Lluís Marquez, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James R. Glass. Evaluating layers of representation in neural machine translation on part-of-speech and semantic tagging tasks. *CoRR*, abs/1801.07772, 2018. URL http://arxiv.org/abs/1801.07772

Denny Britz, Anna Goldie, Minh-Thang Luong, and Quoc V. Le. Massive exploration of neural machine translation architectures. *CoRR*, abs/1703.03906, 2017.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 2342–2350, 2015.

Sumit Chopra Michael Mathieu Marc’Aurelio Ranzato Tomas Mikolov, Armand Joulin. Learning longer memory in recurrent neural networks. *ICLR*, 2015.

Gail Weiss, Yoav Goldberg, and Eran Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. *CoRR*, abs/1711.09576, 2017. URL http://arxiv.org/abs/1711.09576

Xing Shi, Kevin Knight, and Deniz Yuret. Why neural translations are the right length. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2278–2282. Association for Computational Linguistics, 2016. doi: 10.18653/v1/D16-1248. URL http://www.aclweb.org/anthology/D16-1248

N. F. Liu, O. Levy, R. Schwartz, C. Tan, and N. A. Smith. LSTMs Exploit Linguistic Attributes of Data. *ArXiv e-prints*, May 2018.

Quoc V. Le, Navdeep Jaitly, and Geoffrey E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *CoRR*, abs/1504.00941, 2015. URL http://arxiv.org/abs/1504.00941

Armand Joulin and Tomas Mikolov. Inferring algorithmic patterns with stack-augmented recurrent nets. *CoRR*, abs/1503.01007, 2015. URL http://arxiv.org/abs/1503.01007
Felix A Gers and E Schmidhuber. Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340, 2001.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. ISBN 0-262-01097-6. URL [http://dl.acm.org/citation.cfm?id=65669.104451](http://dl.acm.org/citation.cfm?id=65669.104451).

Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. LSTM: A search space odyssey. *CoRR*, abs/1503.04069, 2015.

Hasim Sak, Andrew W. Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*, 2014.

F. A. Gers and J. Schmidhuber. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194 vol.3, 2000. doi: 10.1109/IJCNN.2000.861302.