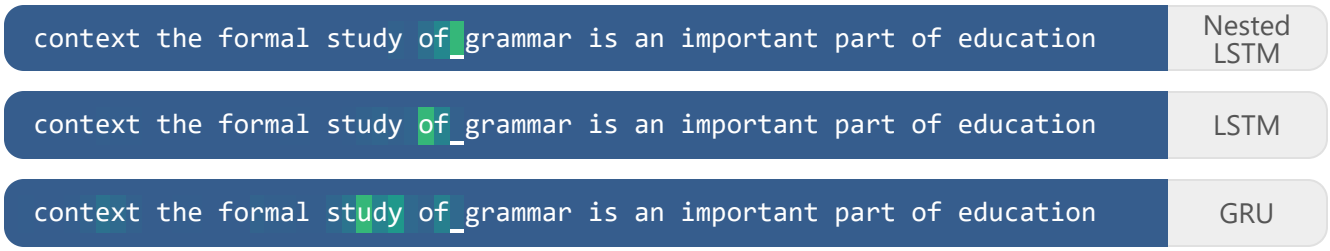
Visualizing memorization in RNNs

Inspecting gradient magnitudes in context can be a powerful tool to see when recurrent units use short-term or long-term contextual understanding.



This **connectivity visualization** shows how strongly previous input characters influence the current target character in an autocomplete problem. For example, in the prediction of "grammar" the GRU RNN initially uses long-term memorization but as more characters become available the RNN switches to short-term memorization. (reset)

AUTHORS
Andreas Madsen

AFFILIATIONS
NearForm Ltd

PUBLISHED
March 25, 2019

DOI
10.23915/distill.00016

Memorization in Recurrent Neural Networks (RNNs) continues to pose a challenge in many applications. We'd like RNNs to be able to store information over many timesteps and retrieve it when it becomes relevant — but vanilla RNNs often struggle to do this.

Several network architectures have been proposed to tackle aspects of this problem, such as Long-Short-Term Memory (LSTM) [1] units and Gated Recurrent Units (GRU) [2]. However, the practical problem of memorization still poses a challenge. As such, developing new recurrent units that are better at memorization continues to be an active field of research.

To compare a recurrent unit against its alternatives, both past and recent papers, such as the Nested LSTM paper by Monzi et al. [3], heavily rely on quantitative comparisons. These comparisons often measure accuracy or cross entropy loss on standard problems such as Penn Treebank [4], Chinese Poetry Generation, or text8 [5], where the task is to predict the next character given existing input.

While quantitative comparisons are useful, they only provide partial insight into the how a recurrent unit memorizes. A model can, for example, achieve high accuracy and cross entropy loss by just providing highly accurate predictions in cases that only require short-term memorization, while being inaccurate at predictions that require long-term memorization. For example, when autocompleting words in a sentence, a model with only short-term understanding could still exhibit high accuracy completing the ends of words once most of the characters are present. However, without longer term contextual understanding it won't be able to predict words when only a few characters are known.

This article presents a qualitative visualization method for comparing recurrent units with regards to memorization and contextual understanding. The method is applied to the three recurrent units mentioned above: Nested LSTMs, LSTMs, and GRUs.

Recurrent Units

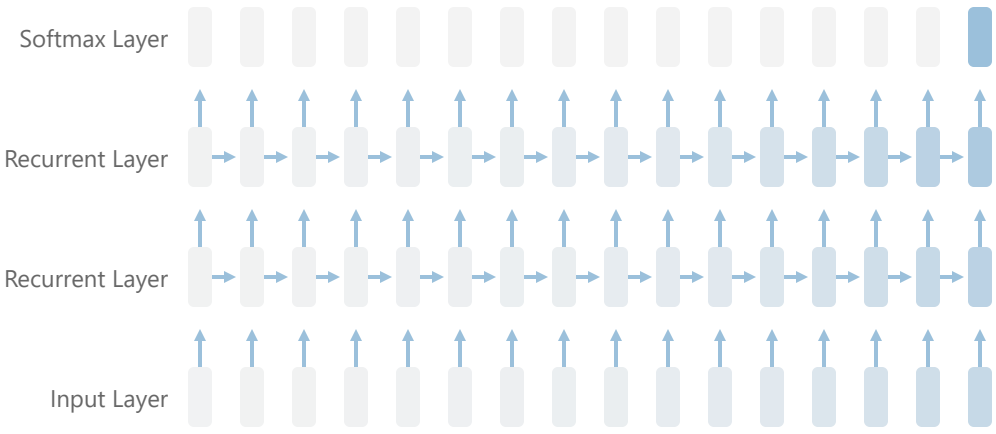
The networks that will be analyzed all use a simple RNN structure:

$$h_\ell^t = \text{Unit}(h_{\ell-1}^t, h_\ell^{t-1}), \text{ where: } h_0^t = x_t$$

Output for layer ℓ at time t. Recurrent unit of choice.

$$y^t = \text{Softmax}(h_L^t)$$

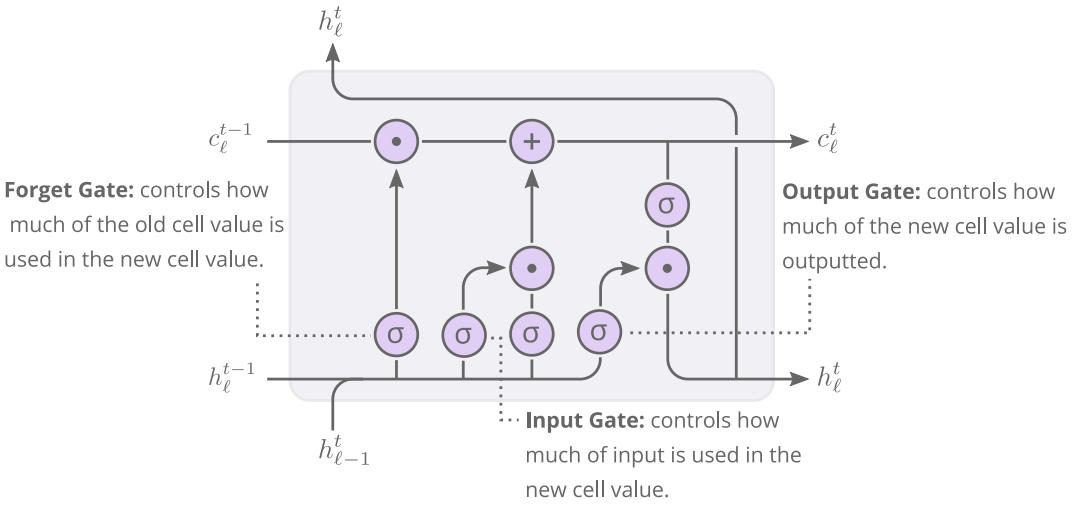
In theory, the time dependency allows it in each iteration to know about every part of the sequence that came before. However, this time dependency typically causes a vanishing gradient problem that results in long-term dependencies being ignored during training [6].



Vanishing Gradient: where the contribution from the earlier steps becomes insignificant in the gradient for the vanilla RNN unit.

Several solutions to the vanishing gradient problem have been proposed over the years. The most popular are the aforementioned LSTM and GRU units, but this is still an area of active research. Both LSTM and GRU are well known and thoroughly explained in literature. Recently, Nested LSTMs have also been proposed [3] — an explanation of Nested LSTMs can be found in the appendix.

Nested LSTM | LSTM | GRU



Recurrent Unit, LSTM: allows for long-term memorization by gateing its update, thereby solving the vanishing gradient problem.

It is not entirely clear why one recurrent unit performs better than another in some applications, while in other applications it is another type of recurrent unit that performs better. Theoretically they all solve the vanishing gradient problem, but in practice their performance is highly application dependent.

Understanding why these differences occur is likely an opaque and challenging problem. The purpose of this article is to demonstrate a visualization technique that can better highlight what these differences are. Hopefully, such an understanding can lead to a deeper understanding.

Comparing Recurrent Units

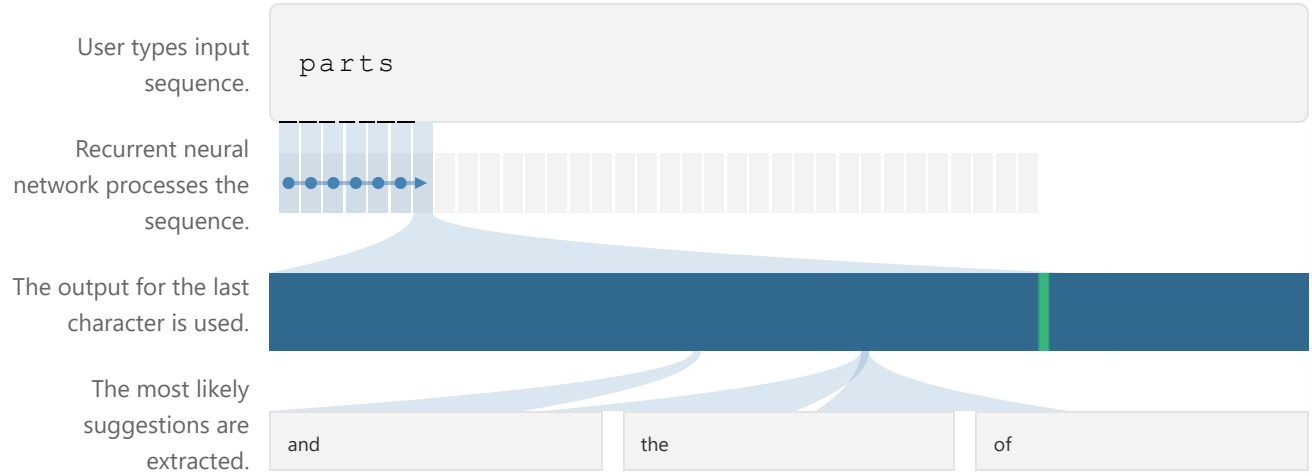
Comparing different Recurrent Units is often more involved than simply comparing the accuracy or cross entropy loss. Differences in these high-level quantitative measures can have many explanations and may only be because of some small improvement in predictions that only requires short-term contextual understanding, while it is often the long-term contextual understanding that is of interest.

A problem for qualitative analysis

Therefore a good problem for qualitatively analyzing contextual understanding should have a human-

interpretable output and depend both on long-term and short-term contextual understanding. The typical problems that are often used, such as Penn Treebank [4], Chinese Poetry Generation, or text8 [5] generation do not have outputs that are easy to reason about, as they require an extensive understanding of either grammar, Chinese poetry, or only output a single letter.
To this end, this article studies the autocomplete problem. Each character is mapped to a target that represents the entire word. The space leading up to the word should also map to that target. This prediction based on the space character is in particular useful for showing contextual understanding.

The autocomplete problem is quite similar to the text8 generation problem: the only difference is that instead of predicting the next letter, the model predicts an entire word. This makes the output much more interpretable. Finally, because of its close relation to text8 generation, existing literature on text8 generation is relevant and comparable, in the sense that models that work well on text8 generation should work well on the autocomplete problem.



Autocomplete: An application that has a humanly interpretable output, while depending on both short and long-term contextual understanding. In this case, the network uses past information and understands the next word should be a country.

The output in this figure was produced by the GRU model; all model setups are described in the appendix. *Try removing the last letters to see that the network continues to give meaningful suggestions. You can also type in your own text.* (reset).

The autocomplete dataset is constructed from the full text8 dataset. The recurrent neural networks used to solve the problem have two layers, each with 600 units. There are three models, using GRU, LSTM, and Nested LSTM. See the appendix for more details.

Connectivity in the Autocomplete Problem

In the recently published Nested LSTM paper [3], they qualitatively compared their Nested LSTM unit to other recurrent units, to show how it memorizes in comparison, by visualizing individual cell activations.

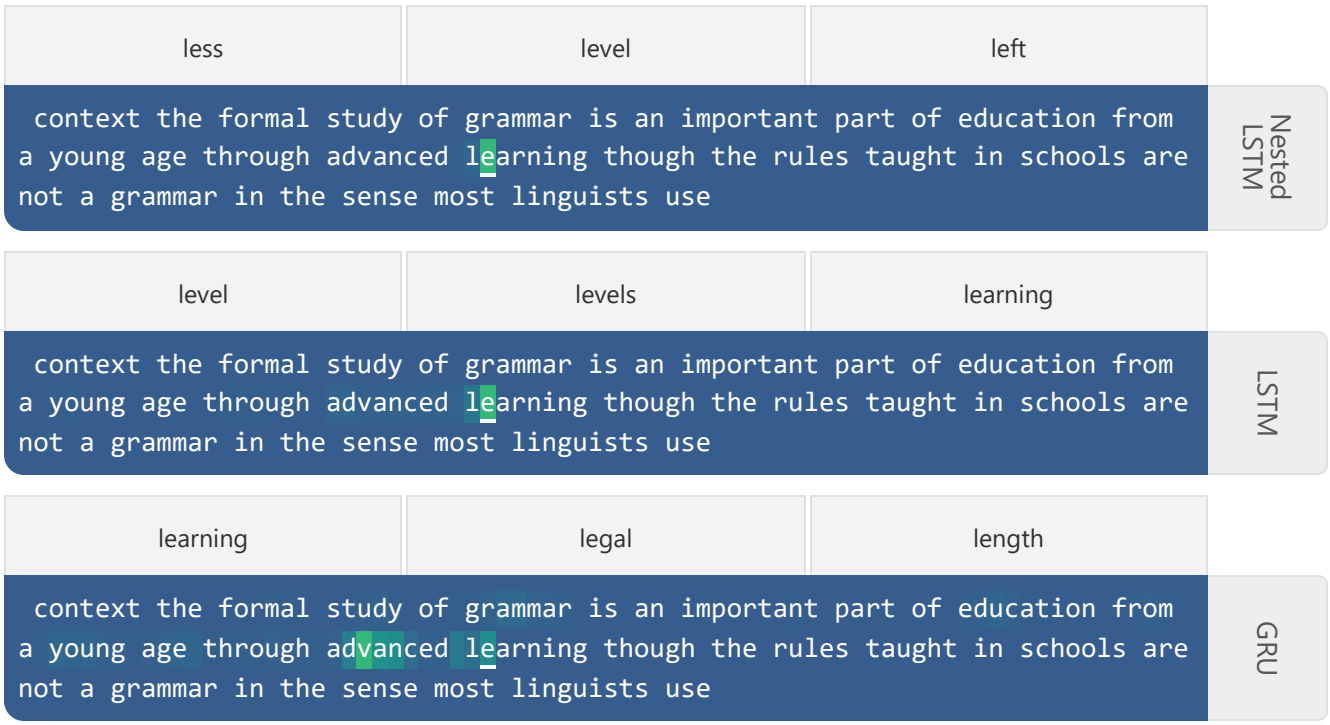
This visualization was inspired by Karpathy et al. [7] where they identify cells that capture a specific feature. To identify a specific feature, this visualization approach works well. However, it is not a useful argument for memorization in general as the output is entirely dependent on what feature the specific cell captures.

Instead, to get a better idea of how well each model memorizes and uses memory for contextual understanding, the connectivity between the desired output and the input is analyzed. This is calculated as:

$$\text{connectivity}(\quad t \quad , \quad \tilde{t} \quad) = \left|\left| \frac{\partial (h_L^{\tilde{t}})_k}{\partial x^t} \right|\right|_2$$

Input time index. Output time index. Magnitude of the gradient, between the logits for the desired output $(h_L^{\tilde{t}})_k$ and the input x^t.

Exploring the connectivity gives a surprising amount of insight into the different models' ability for long-term contextual understanding. Try and interact with the figure below yourself to see what information the different models use for their predictions.



Connectivity: the connection strength between the target for the selected character and the input characters is highlighted in green (reset). *Hover over or tap the text to change the selected character.*

Let's highlight three specific situations:

1. Observe how the models predict the word "learning" with only the first two characters as input. The Nested LSTM model barely uses past information and thus only suggests common words starting with the letter "l".

In contrast, the LSTM and GRU models both suggests the word “learning”. The GRU model shows stronger connectivity with the word “advanced”, and we see in the suggestions that it predicts a higher probability for “learning” than the LSTM model.

1 2 3

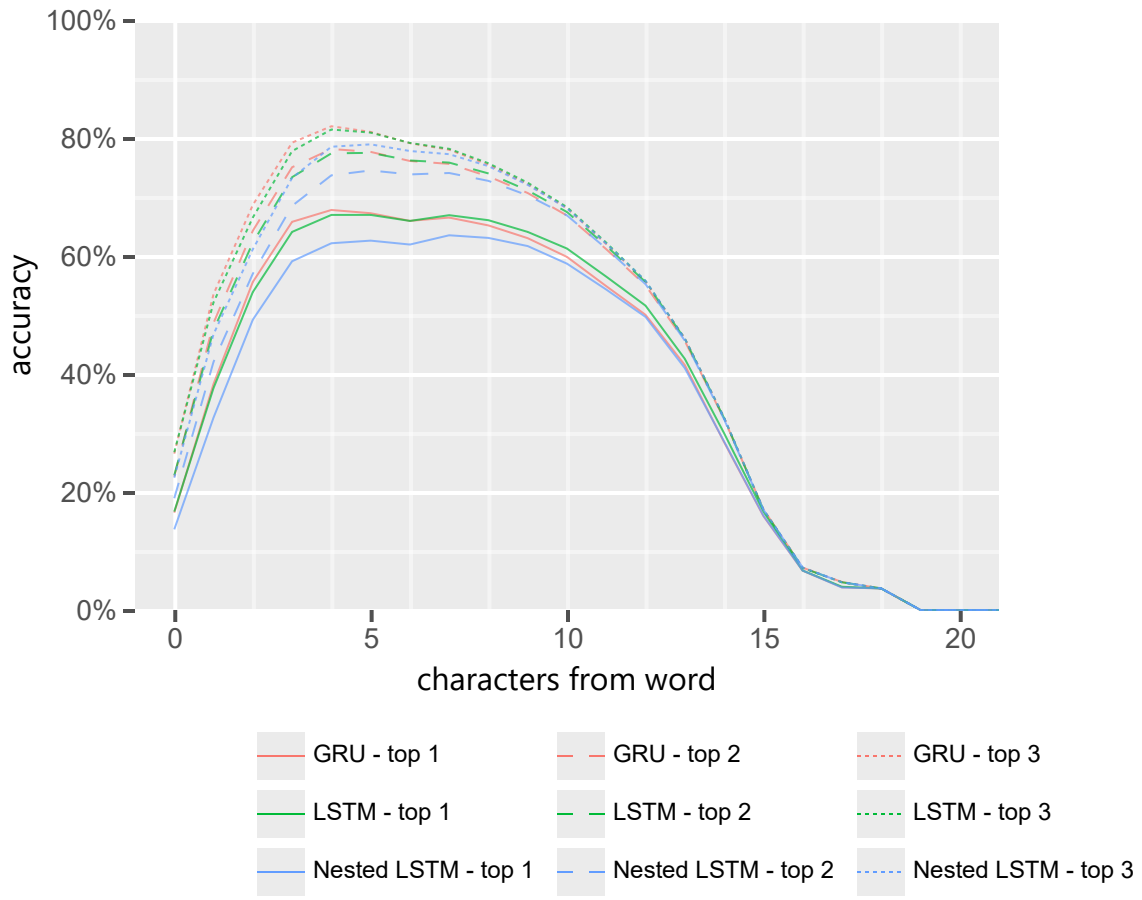
The colored number links above change the connectivity figure's displayed timestep and explanation.

These observations show that the connectivity visualization is a powerful tool for comparing models in terms of which previous inputs they use for contextual understanding. However, it is only possible to compare models on the same dataset, and on a specific example. As such, while these observations may show that Nested LSTM is not very capable of long-term contextual understanding in this example; these observations may not generalize to other datasets or hyperparameters.

Future work; quantitative metric

From the above observations it appears that short-term contextual understanding often involves the word that is being predicted itself. That is, the models switch to using previously seen letters from the word itself, as more letters become available. In contrast, at the beginning of predicting a word, models — especially the GRU network — use previously seen words as context for the prediction.

This observation suggests a quantitative metric: measure the accuracy given how many letters from the word being predicted are already known. It is not clear that this is best quantitative metric: it is highly problem dependent, and it also doesn’t summarize the model to a single number, which one may wish for a more direct comparison.



Accuracy Graph: shows the accuracy given a fixed number of characters in a word that the RNN has seen. 0 characters mean that the RNN has only seen the space leading up to the word, including all the previous text which should provide context. The different line styles, indicates if the correct word should appear among the top 1, 2, or 3 suggestions.

These results suggest that the GRU model is better at long-term contextual understanding, while the LSTM model is better at short-term contextual understanding. These observations are valuable, as it justifies why the overall accuracy of the GRU and LSTM models are almost identical, while the connectivity visualization shows that the GRU model is far better at long-term contextual understanding.

While more detailed quantitative metrics like this provides new insight, qualitative analysis like the connectivity figure presented in this article still has great value. As the connectivity visualization gives an intuitive understanding of how the model works, which a quantitative metric cannot. It also shows that a wrong prediction can still be considered a useful prediction, such as a synonym or a contextually reasonable prediction.

Conclusion

Looking at overall accuracy and cross entropy loss in itself is not that interesting. Different models may prioritize either long-term or short-term contextual understanding, while both models can have similar accuracy and cross entropy.

A qualitative analysis, where one looks at how previous input is used in the prediction is therefore also important when judging models. In this case, the connectivity visualization together with the autocomplete predictions, reveals that the GRU model is much more capable of long-term contextual understanding, compared to LSTM and Nested LSTM. In the case of LSTM, the difference is much higher than one would guess from just looking at the overall accuracy and cross entropy loss alone. This observation is not that interesting in itself as it is likely very dependent on the hyperparameters, and the specific application.

Much more valuable is that this visualization method makes it possible to intuitively understand how the models are different, to a much higher degree than just looking at accuracy and cross entropy. For this application, it is clear that the GRU model uses repeating words and semantic meaning of past words to make its prediction, to a much higher degree than the LSTM and Nested LSTM models. This is both a valuable insight when choosing the final model, but also essential knowledge when developing better models in the future.

Acknowledgments

Many thanks to the authors of the original Nested LSTM paper [3], Joel Ruben, Antony Moniz, and David Krueger. Even though the findings here were not the same, the paper have inspired much of this article, as it shows that something as familiar as the recurrent unit is still an open research area.

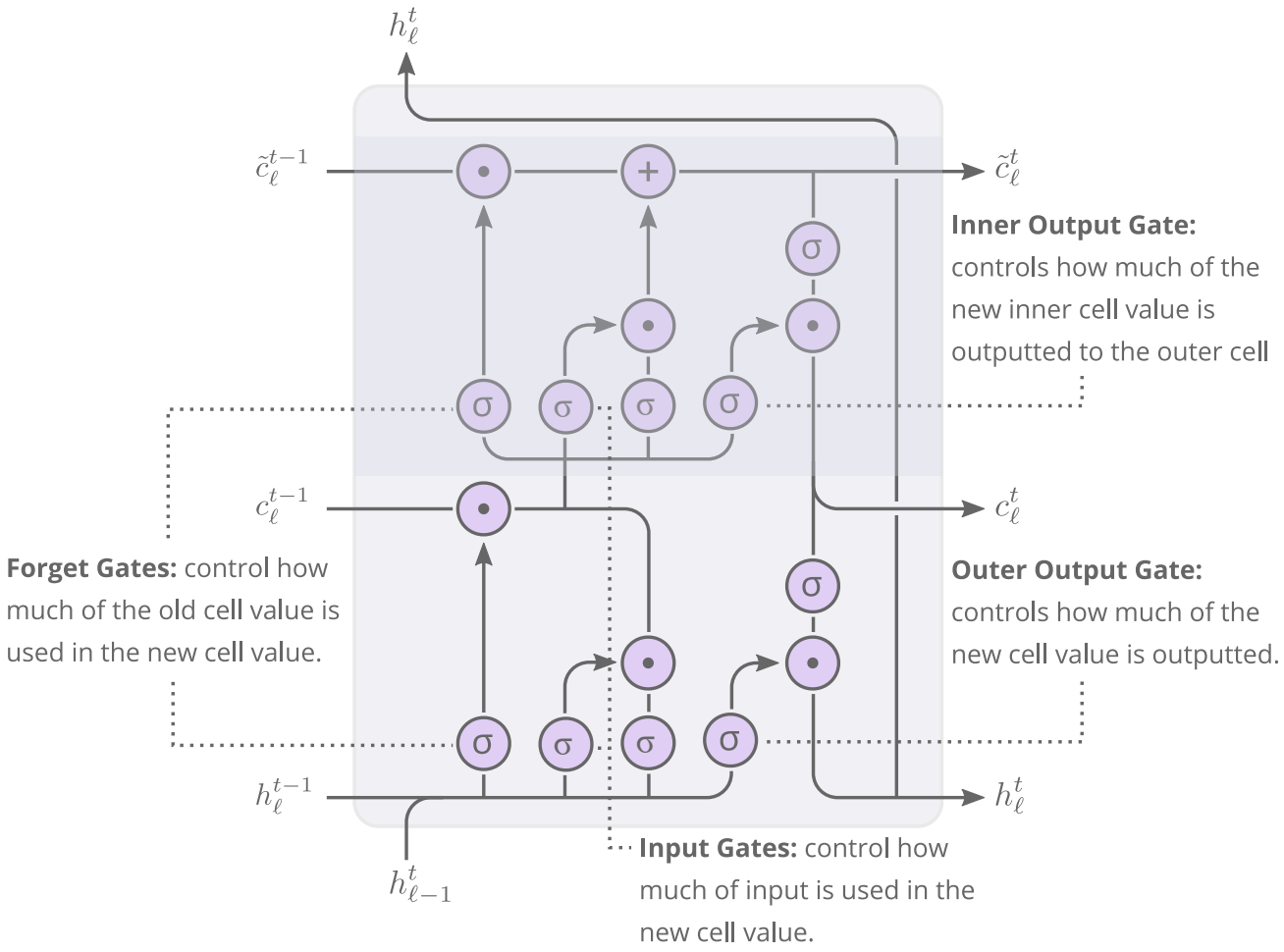
I am also grateful for the excellent feedback and patience from the Distill team, especially Christopher Olah and Ludwig Schubert, as well as the feedback from the peer-reviewers. Their feedback has dramatically improved the quality of this article.

Discussion and Review

Review 1 - Abhinav Sharma
Review 2 - Dylan Cashman
Review 3 - Ruth Fong

Nested LSTM

The Nested LSTM unit attempt to solve the long-term memorization from a more practical point of view. Where the standard LSTM unit solves the vanishing gradient problem by adding internal memory, and the GRU attempt to be a faster solution than LSTM by using no internal memory, the Nested LSTM goes in the opposite direction of GRU by adding additional memory to the unit [3]. The idea here is that adding additional memory to the unit allows for more long-term memorization.



Nested LSTM: makes the cell update depend on another LSTM unit, supposedly this allows more long-term memory compared to stacking LSTM layers.

The additional memory is integrated into the LSTM unit by changing how the cell value c_ℓ^t is updated. Instead of defining the cell value update as $c_\ell^t = i_\ell^t \odot z_\ell^t + f_\ell^t \odot c_\ell^{t-1}$, as done in vanilla LSTM, it uses another LSTM unit:

$$c_\ell^t = \text{LSTM}(i_\ell^t \odot z_\ell^t, f_\ell^t \odot c_\ell^{t-1})$$

See the defintion of $\text{LSTM}(\cdot, \cdot)$ futher down in the appendix.

The complete set of equations then becomes:

$$\begin{aligned}
i_\ell^t &= \sigma_i(IW_{\ell-1,\ell} h_{\ell-1}^t + IW_{\ell,\ell} h_\ell^{t-1}) \\
f_\ell^t &= \sigma_f(FW_{\ell-1,\ell} h_{\ell-1}^t + FW_{\ell,\ell} h_\ell^{t-1}) \\
o_\ell^t &= \sigma_o(OW_{\ell-1,\ell} h_{\ell-1}^t + OW_{\ell,\ell} h_\ell^{t-1}) \\
z_\ell^t &= \sigma_z(ZW_{\ell-1,\ell} h_{\ell-1}^t + ZW_{\ell,\ell} h_\ell^{t-1}) \\
c_\ell^t &= \text{LSTM}(i_\ell^t \odot z_\ell^t, f_\ell^t \odot c_\ell^{t-1}) \\
h_\ell^t &= \text{NLSTM}(h_{\ell-1}^t, h_\ell^{t-1}) := o_\ell^t \odot \sigma_h(c_\ell^t)
\end{aligned}$$

Like in vanilla LSTM, the gate activation functions $(\sigma_i(\cdot), \sigma_f(\cdot), \text{ and } \sigma_o(\cdot))$ are usually the simoid activation function. However, only the $\sigma_h(\cdot)$ is set to $\tanh(\cdot)$. While, $\sigma_z(\cdot)$ is just the identity function, otherwise two non-linear activation functions would be applied on the same scalar without any change, except for the multiplication by the input gate. The activation functions for $\text{LSTM}(\cdot, \cdot)$ remains the same.

The abstraction, of how to combine the input with the cell value, allows for a lot of flexibility. Using this abstraction, it is not only possible to add one extra internal memory state but the internal $\mathrm{LSTM}(\cdot,\cdot)$ unit can recursively be replaced by as many internal $\mathrm{NLSTM}(\cdot,\cdot)$ units as one would wish, thereby adding even more internal memory.

Long Short-Term Memory

The equations defining $\mathrm{LSTM}(\cdot,\cdot)$ as used in $\mathrm{NLSTM}(\cdot,\cdot)$ are:

$$
\begin{aligned}
\tilde{i}_\ell^t &= \sigma_i(IW_{\ell-1,\ell}\tilde{h}_{\ell-1}^t + IW_{\ell,\ell}\tilde{h}_\ell^{t-1}) \\
\tilde{f}_\ell^t &= \sigma_f(FW_{\ell-1,\ell}\tilde{h}_{\ell-1}^t + FW_{\ell,\ell}\tilde{h}_\ell^{t-1}) \\
\tilde{o}_\ell^t &= \sigma_o(OW_{\ell-1,\ell}\tilde{h}_{\ell-1}^t + OW_{\ell,\ell}\tilde{h}_\ell^{t-1}) \\
\tilde{z}_\ell^t &= \sigma_z(ZW_{\ell-1,\ell}\tilde{h}_{\ell-1}^t + ZW_{\ell,\ell}\tilde{h}_\ell^{t-1}) \\
\tilde{c}_\ell^t &= \tilde{i}_\ell^t \odot \tilde{z}_\ell^t + \tilde{f}_\ell^t \odot \tilde{c}_\ell^{t-1} \\
\tilde{h}_\ell^t &= \mathrm{LSTM}(\tilde{h}_{\ell-1}^t, \tilde{h}_\ell^{t-1}) \coloneqq \tilde{o}_\ell^t \odot \sigma_h(\tilde{c}_\ell^t)
\end{aligned}
$$

In terms of the Nested LSTM unit, $\tilde{h}_{\ell-1}^t = i_\ell^t \odot z_\ell^t$ and $\tilde{h}_\ell^{t-1} = f_\ell^t \odot c_\ell^{t-1}$.

The gate activation functions $(\sigma_i(\cdot), \sigma_f(\cdot),\ \text{and}\ \sigma_o(\cdot))$ are usually the simoid activation function. While $(\sigma_z(\cdot)\ \text{and}\ \sigma_h(\cdot))$ are usually $\tanh(\cdot)$.

Autocomplete Problem

The autocomplete dataset is constructed from the full text8 dataset, where each observation consists of maximum 200 characters and is ensured not to contain partial words. 90% of the observations are used for training, 5% for validation and 5% for testing.

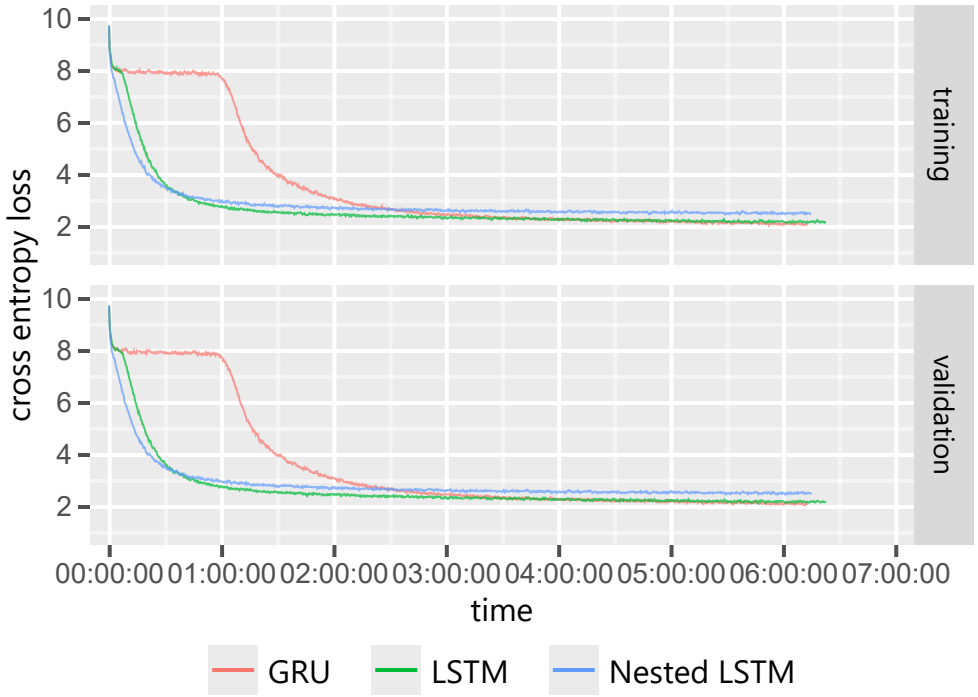
The input vocabulary is a-z, space, and a padding symbol. The output vocabulary consists of the $2^{14} = 16384$ most frequent words, and two additional symbols, one for padding and one for unknown words. The network is not penalized for predicting padding and unknown words wrong.

The GRU, LSTM each have 2 layers of 600 units. Similarly, the Nested LSTM model has 1 layer of 600 units but with 2 internal memory states. Additionally, each model has an input embedding layer and a final dense layer to match the vocabulary size.

Model	Units	Layers	Depth	Parameters		
				Embedding	Recurrent	Dense
GRU	600	2	N/A	16200	4323600	9847986
LSTM	600	2	N/A	16200	5764800	9847986
Nested LSTM	600	1	2	16200	5764800	9847986

Model Configurations: shows the number of layers, units, and parameters for each model.

There are 456896 sequences in the training dataset, and a mini-batch size of 64 observations is used. A single iteration over the entire dataset then corresponds to 7139 mini-batches. The training runs twice over the dataset, thus corresponding to trained for 14278 mini-batches. For training, Adam optimization is used with default parameters.



Model training: shows the training loss and validation loss for the GRU, LSTM, and Nested LSTM models when training on the autocomplete problem. The x-axis is time or mini-batches.

Evaluating the model on the test-dataset yields the following cross entropy losses and accuracies.

Model	Cross Entropy	Accuracy
GRU	2.1170	52.01%
LSTM	2.1713	51.40%
Nested LSTM	2.4950	47.10%

Model testing: shows the testing loss and accuracy for the GRU, LSTM, and Nested LSTM models on the autocomplete problem.

The implementation is available at https://github.com/distillpub/post — memorization-in-rnns .

References

1. **Long short-term memory**
   Hochreiter, S. and Schmidhuber, J., 1997. Neural computation, Vol 9(8), pp. 1735--1780. MIT Press.

2. **Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation** [PDF]
   Cho, K., Merrienboer, B.v., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y., 2014. arXivreprint arXiv:1406.1078.

3. **Nested LSTMs** [PDF]
   Moniz, J.R.A. and Krueger, D., 2018. arXivreprint arXiv:1801.10308.

4. **The Penn Treebank: Annotating Predicate Argument Structure** [link]
   Marcus, M., Kim, G., Marcinkiewicz, M.A., MacIntyre, R., Bies, A., Ferguson, M., Katz, K. and Schasberger, B., 1994. Proceedings of the

Workshop on Human Language Technology, pp. 114--119. Association for Computational Linguistics. DOI: 10.3115/1075812.1075835

5. **text8 Dataset** [link]
   Mahoney, M., 2006.

6. **On the difficulty of training recurrent neural networks** [PDF]
   Pascanu, R., Mikolov, T. and Bengio, Y., 2013. arXivreprint arXiv:1211.5063.

7. **Visualizing and Understanding Recurrent Networks** [PDF]
   Karpathy, A., Johnson, J. and Fei-Fei, L., 2015. arXivreprint arXiv:1506.02078.

Updates and Corrections

If you see mistakes or want to suggest changes, please create an issue on GitHub.

Reuse

Citation

For attribution in academic contexts, please cite this work as

```
Madsen, "Visualizing memorization in RNNs", Distill, 2019.
```

BibTeX citation

```
@article{madsen2019visualizing,
  author = {Madsen, Andreas},
  title = {Visualizing memorization in RNNs},
  journal = {Distill},
  year = {2019},
  note = {https://distill.pub/2019/memorization-in-rnns},
  doi = {10.23915/distill.00016}
}
```