Long Short-Term Memory as a Dynamically Computed Element-wise Weighted Sum

Omer Levy* **Kenton Lee*** **Nicholas FitzGerald** **Luke Zettlemoyer**
Paul G. Allen School, University of Washington, Seattle, WA
{omerlevy,kentonl,nfitz,lsz}@cs.washington.edu

Abstract

LSTMs were introduced to combat vanishing gradients in simple RNNs by augmenting them with gated additive recurrent connections. We present an alternative view to explain the success of LSTMs: the gates themselves are versatile recurrent models that provide more representational power than previously appreciated. We do this by decoupling the LSTM's gates from the embedded simple RNN, producing a new class of RNNs where the recurrence computes an element-wise weighted sum of context-independent functions of the input. Ablations on a range of problems demonstrate that the gating mechanism alone performs as well as an LSTM in most settings, strongly suggesting that the gates are doing much more in practice than just alleviating vanishing gradients.

1 Introduction

Long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997) has become the de-facto recurrent neural network (RNN) for learning representations of sequences in NLP. Like simple recurrent neural networks (S-RNNs) (Elman, 1990), LSTMs are able to learn non-linear functions of arbitrary-length input sequences. However, they also introduce an additional memory cell to mitigate the vanishing gradient problem (Hochreiter, 1991; Bengio et al., 1994). This memory is controlled by a mechanism of gates, whose additive connections allow long-distance dependencies to be learned more easily during backpropagation. While this view is mathematically accurate, in this paper we argue that it does not provide a complete picture of why LSTMs work in practice.

We present an alternate view to explain the success of LSTMs: the gates themselves are powerful recurrent models that provide more representational power than previously realized. To demonstrate this, we first show that LSTMs can be seen as a combination of two recurrent models: (1) an S-RNN, and (2) an element-wise weighted sum of the S-RNN's outputs over time, which is implicitly computed by the gates. We hypothesize that, for many practical NLP problems, the weighted sum serves as the main modeling component. The S-RNN, while theoretically expressive, is in practice only a minor contributor that clouds the mathematical clarity of the model. By replacing the S-RNN with a context-*independent* function of the input, we arrive at a much more restricted class of RNNs, where the main recurrence is via the element-wise weighted sums that the gates are computing.

We test our hypothesis on NLP problems, where LSTMs are wildly popular at least in part due to their ability to model crucial phenomena such as word order (Adi et al., 2017), syntactic structure (Linzen et al., 2016), and even long-range semantic dependencies (He et al., 2017). We consider four challenging tasks: language modeling, question answering, dependency parsing, and machine translation. Experiments show that while removing the gates from an LSTM can severely hurt performance, replacing the S-RNN with a simple linear transformation of the input results in minimal or no loss in model performance. We also show that, in many cases, LSTMs can be further simplified by removing the output gate, arriving at an even more transparent architecture, where the output is a context-*independent* function of the weighted sum. Together, these results suggest that the gates' ability to compute an element-wise weighted sum, rather than the non-linear transition dynamics of S-RNNs, are the driving force behind LSTM's success.

*The first two authors contributed equally to this paper.

Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Short Papers), pages 732–739
Melbourne, Australia, July 15 - 20, 2018.

2 What Do Memory Cells Compute?

LSTMs are typically motivated as an augmentation of simple RNNs (S-RNNs), defined as:

$$\boldsymbol{h}_t = \tanh(\boldsymbol{W}_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{hx}\boldsymbol{x}_t + \boldsymbol{b}_h) \qquad (1)$$

S-RNNs suffer from the vanishing gradient problem (Hochreiter, 1991; Bengio et al., 1994) due to compounding multiplicative updates of the hidden state. By introducing a memory cell and an output layer controlled by gates, LSTMs enable shortcuts through which gradients can flow when learning with backpropagation. This mechanism enables learning of long-distance dependencies while preserving the expressive power of recurrent non-linear transformations provided by S-RNNs.

Rather than viewing the gates as simply an auxiliary mechanism to address a *learning* problem, we present an alternate view that emphasizes their *modeling* strengths. We argue that the LSTM should be interpreted as a hybrid of two distinct recurrent architectures: (1) the S-RNN which provides multiplicative connections across timesteps, and (2) the memory cell which provides additive connections across timesteps. On top of these recurrences, an output layer is included that simply squashes and filters the memory cell at each step.

Throughout this paper, let $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ be the sequence of input vectors, $\{\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n\}$ be the sequence of output vectors, and $\{\boldsymbol{c}_1, \ldots, \boldsymbol{c}_n\}$ be the memory cell's states. Then, given the basic LSTM definition below, we can formally identify three sub-components.

$$\begin{aligned}
\widetilde{\boldsymbol{c}}_t &= \tanh(\boldsymbol{W}_{ch}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{cx}\boldsymbol{x}_t + \boldsymbol{b}_c) && (2)\\
\boldsymbol{i}_t &= \sigma(\boldsymbol{W}_{ih}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ix}\boldsymbol{x}_t + \boldsymbol{b}_i) && (3)\\
\boldsymbol{f}_t &= \sigma(\boldsymbol{W}_{fh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{fx}\boldsymbol{x}_t + \boldsymbol{b}_f) && (4)\\
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} && (5)\\
\boldsymbol{o}_t &= \sigma(\boldsymbol{W}_{oh}\boldsymbol{h}_{t-1} + \boldsymbol{W}_{ox}\boldsymbol{x}_t + \boldsymbol{b}_o) && (6)\\
\boldsymbol{h}_t &= \boldsymbol{o}_t \circ \tanh(\boldsymbol{c}_t) && (7)
\end{aligned}$$

Content Layer (Equation 2) We refer to $\widetilde{\boldsymbol{c}}_t$ as the content layer, which is the output of an S-RNN. Evaluating the need for multiplicative recurrent connections in the content layer is the focus of this work. The content layer is passed to the memory cell, which decides which parts of it to store.

Memory Cell (Equations 3-5) The memory cell $\boldsymbol{c}_t$ is controlled by two gates. The input gate $\boldsymbol{i}_t$ controls what part of the content ($\widetilde{\boldsymbol{c}}_t$) is written to the memory, while the forget gate $\boldsymbol{f}_t$ controls what part of the memory is deleted by filtering the previous state of the memory ($\boldsymbol{c}_{t-1}$). Writing to the memory is done by adding the filtered content ($\boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t$) to the retained memory ($\boldsymbol{f}_t \circ \boldsymbol{c}_{t-1}$).

Output Layer (Equations 6-7) The output layer $\boldsymbol{h}_t$ passes the memory cell through a $\tanh$ activation function and uses an output gate $\boldsymbol{o}_t$ to read selectively from the squashed memory cell.

Our goal is to study how much each of these components contribute to the empirical performance of LSTMs. In particular, it is worth considering the memory cell in more detail to reveal why it could serve as a standalone powerful model of long-distance context. It is possible to show that it implicitly computes an *element-wise weighted sum* of all the previous content layers by expanding the recurrence relation in Equation 5:

$$\begin{aligned}
\boldsymbol{c}_t &= \boldsymbol{i}_t \circ \widetilde{\boldsymbol{c}}_t + \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1}\\
&= \sum_{j=0}^{t} \left(\boldsymbol{i}_j \circ \prod_{k=j+1}^{t} \boldsymbol{f}_k\right) \circ \widetilde{\boldsymbol{c}}_j\\
&= \sum_{j=0}^{t} \boldsymbol{w}_j^t \circ \widetilde{\boldsymbol{c}}_j
\end{aligned} \qquad (8)$$

Each weight $\boldsymbol{w}_j^t$ is a product of the input gate $\boldsymbol{i}_j$ (when its respective input $\widetilde{\boldsymbol{c}}_j$ was read) and every subsequent forget gate $\boldsymbol{f}_k$. An interesting property of these weights is that, like the gates, they are also soft element-wise binary filters.

3 Standalone Memory Cells are Powerful

The restricted space of element-wise weighted sums allows for easier mathematical analysis, visualization, and perhaps even learnability. However, constrained function spaces are also less expressive, and a natural question is whether these models will work well for NLP problems that involve understanding context. We hypothesize that the memory cell (which computes weighted sums) can function as a standalone contextualizer. To test this hypothesis, we present several simplifications of the LSTM's architecture (Section 3.1), and show on a variety of NLP benchmarks that there is a qualitative performance difference between models that contain a memory cell and those that do not (Section 3.2). We conclude that the content and output layers are relatively minor contributors, and that the space of element-wise weighted sums is sufficiently powerful to compete with fully parameterized LSTMs (Section 3.3).

3.1 Simplified Models

The modeling power of LSTMs is commonly assumed to derive from the S-RNN in the content layer, with the rest of the model acting as a learning aid to bypass the vanishing gradient problem. We first isolate the S-RNN by ablating the gates (denoted as *LSTM – GATES* for consistency).

To test whether the memory cell has enough modeling power of its own, we take an LSTM and replace the S-RNN in the content layer from Equation 2 with a simple linear transformation ($\widetilde{c}_t = \boldsymbol{W}_{cx}\boldsymbol{x}_t$) creating the *LSTM – S-RNN* model.

We further simplify the LSTM by removing the output gate from Equation 7 ($h_t = \tanh(\boldsymbol{c}_t)$), leaving only the activation function in the output layer (*LSTM – S-RNN – OUT*). After removing the S-RNN and the output gate from the LSTM, the entire ablated model can be written in a modular, compact form:

$$\boldsymbol{h}_t = \text{OUTPUT}\Big(\sum_{j=0}^{t} \boldsymbol{w}_j^t \circ \text{CONTENT}(\boldsymbol{x}_j)\Big) \quad (9)$$

where the content layer $\text{CONTENT}(\cdot)$ and the output layer $\text{OUTPUT}(\cdot)$ are both context-*independent* functions, making the entire model highly constrained and mathematically simpler. The complexity of modeling contextual information is needed only for computing the weights $\boldsymbol{w}_j^t$. As we will see in Section 3.2, both of these ablations perform on par with LSTMs on several tasks.

Finally, we ablate the hidden state from the gates as well, by computing each gate $\boldsymbol{g}_t$ via $\sigma(\boldsymbol{W}_{gx}\boldsymbol{x}_t + \boldsymbol{b}_g)$. In this model, the only recurrence is the additive connection in the memory cell; it has no multiplicative recurrent connections at all. It can be seen as a type of QRNN (Bradbury et al., 2016) or SRU (Lei et al., 2017b), but for consistency we label it as *LSTM – S-RNN – HIDDEN*.

3.2 Experiments

We compare model performance on four NLP tasks, with an experimental setup that is lenient towards LSTMs and harsh towards its simplifications. In each case, we use existing implementations and previously reported hyperparameter settings. Since these settings were tuned for LSTMs, any simplification that performs equally to (or better than) LSTMs under these LSTM-friendly settings provides strong evidence that the ablated component is not a contributing factor. For each task we also report the mean and standard deviation of 5 runs of the LSTM settings to demonstrate the typical variance observed due to training with different random initializations.

Language Modeling We evaluate the models on the Penn Treebank (PTB) (Marcus et al., 1993) language modeling benchmark. We use the implementation of Zaremba et al. (2014) from TensorFlow's tutorial while replacing any invocation of LSTMs with simpler models. We test two of their configurations: *medium* and *large* (Table 1).

Question Answering For question answering, we use two different QA systems on the Stanford question answering dataset (SQuAD) (Rajpurkar et al., 2016): the Bidirectional Attention Flow model (BiDAF) (Seo et al., 2016) and DrQA (Chen et al., 2017). BiDAF contains 3 LSTMs, which are referred to as the phrase layer, the modeling layer, and the span end encoder. Our experiments replace each of these LSTMs with their simplified counterparts. We directly use the implementation of BiDAF from AllenNLP (Gardner et al., 2017), and all experiments reuse the existing hyperparameters that were tuned for LSTMs. Likewise, we use an open-source implementation of DrQA[1] and replace only the LSTMs, while leaving everything else intact. Table 2 shows the results.

Dependency Parsing For dependency parsing, we use the Deep Biaffine Dependency Parser (Dozat and Manning, 2016), which relies on stacked bidirectional LSTMs to learn context-sensitive word embeddings for determining arcs between a pair of words. We directly use their released implementation, which is evaluated on the Universal Dependencies English Web Treebank v1.3 (Silveira et al., 2014). In our experiments, we use the existing hyperparameters and only replace the LSTMs with the simplified architectures. Table 3 shows the results.

Machine Translation For machine translation, we used OpenNMT (Klein et al., 2017) to train English to German translation models on the multimodal benchmarks from WMT 2016 (used in OpenNMT's readme file). We use OpenNMT's default model and hyperparameters, replacing the stacked bidirectional LSTM encoder with the sim-

[1] https://github.com/hitvoice/DrQA

Configuration	Model	Perplexity
PTB (Medium)	LSTM	83.9 ± 0.3
	– S-RNN	80.5
	– S-RNN – OUT	81.6
	– S-RNN – HIDDEN	83.3
	– GATES	140.9
PTB (Large)	LSTM	78.8 ± 0.2
	– S-RNN	76.0
	– S-RNN – OUT	78.5
	– S-RNN – HIDDEN	82.9
	– GATES	126.1

Table 1: Performance on language modeling benchmarks, measured by perplexity.

System	Model	EM	F1
BiDAF	LSTM	67.9 ± 0.3	77.5 ± 0.2
	– S-RNN	68.4	78.2
	– S-RNN – OUT	67.4	77.2
	– S-RNN – HIDDEN	66.5	76.6
	– GATES	62.9	73.3
DrQA	LSTM	68.8 ± 0.2	78.2 ± 0.2
	– S-RNN	68.0	77.2
	– S-RNN – OUT	68.7	77.9
	– S-RNN – HIDDEN	67.9	77.2
	– GATES	56.4	66.5

Table 2: Performance on SQuAD, measured by exact match (EM) and span overlap (F1).

Model	UAS	LAS
LSTM	90.60 ± 0.21	88.05 ± 0.33
– S-RNN	90.77	88.49
– S-RNN – OUT	90.70	88.31
– S-RNN – HIDDEN	90.53	87.96
– GATES	87.75	84.61

Table 3: Performance on the universal dependencies parsing benchmark, measured by unlabeled (UAS) and labeled attachment score (LAS).

Model	BLEU
LSTM	38.19 ± 0.1
– S-RNN	37.84
– S-RNN – OUT	38.36
– S-RNN – HIDDEN	36.98
– GATES	26.52

Table 4: Performance on the WMT 2016 multimodal English to German benchmark.

plified architectures.[2] Table 4 shows the results.

3.3 Discussion

We showed four major ablations of the LSTM. In the S-RNN experiments (*LSTM – GATES*), we ablate the memory cell and the output layer. In the *LSTM – S-RNN* and *LSTM – S-RNN – OUT* experiments, we ablate the S-RNN. In the *LSTM – S-RNN – HIDDEN*, we remove not only the S-RNN in the content layer, but also the S-RNNs in the gates, resulting in a model whose sole recurrence is in the memory cell's additive connection.

As consistent with previous literature, removing the memory cell degrades performance drastically. In contrast, removing the S-RNN makes little to no difference in the final performance, suggesting that the memory cell alone is largely responsible for the success of LSTMs in NLP.

Even after removing every multiplicative recurrence from the memory cell itself, the model's performance remains well above the vanilla S-RNN's, and falls within the standard deviation of an LSTM's on some tasks (see Table 3). This latter result indicates that the additive recurrent connection in the memory cell – and not the multiplicative recurrent connections in the content layer or in the gates – is the most important computational element in an LSTM. As a corollary, this result also suggests that a weighted sum of context words, while mathematically simple, is a powerful model of contextual information.

4 LSTM as Self-Attention

Attention mechanisms are widely used in the NLP literature to aggregate over a sequence (Cho et al., 2014; Bahdanau et al., 2015) or contextualize tokens within a sequence (Cheng et al., 2016; Parikh et al., 2016) by *explicitly* computing weighted sums. In the previous sections, we demonstrated that LSTMs implicitly compute weighted sums as well, and that this computation is central to their success. How, then, are these two computations related, and in what ways do they differ?

After simplifying the content layer and removing the output gate (*LSTM – S-RNN – OUT*), the model's computation can be expressed as a weighted sum of context-independent functions of the inputs (Equation 9 in Section 3.1). This formula abstracts over both the simplified LSTM and the family of attention mechanisms, and through this lens, the memory cell's computation can be seen as a "cousin" of self-attention. In fact, we can also leverage this abstraction to visualize the

[2] For the S-RNN baseline (*LSTM – GATES*), we had to tune the learning rate to 0.1 because the default value (1.0) resulted in exploding gradients. This is the only case where hyperparameters were modified in all of our experiments.

simplified LSTM's weights as is commonly done with attention (see Appendix A for visualization).

However, there are three major differences in how the *weights* w_j^t are computed.

First, the LSTM's weights are *vectors*, while attention typically computes scalar weights; i.e. a separate weighted sum is computed for every dimension of the LSTM's memory cell. Multi-headed self-attention (Vaswani et al., 2017) can be seen as a middle ground between the two approaches, allocating a scalar weight for different subsets of the dimensions.

Second, the weighted sum is accumulated with a dynamic program. This enables a linear rather than quadratic complexity in comparison to self-attention, but reduces the amount of parallel computation. This accumulation also creates an inductive bias of attending to nearby words, since the weights can only decrease over time.

Finally, attention has a probabilistic interpretation due to the softmax normalization, while the sum of weights in LSTMs can grow up to the sequence length. In variants of the LSTM that tie the input and forget gate, such as coupled-gate LSTMs (Greff et al., 2016) and GRUs (Cho et al., 2014), the memory cell instead computes a weighted *average* with a probabilistic interpretation. These variants compute locally normalized distributions via a product of sigmoids rather than globally normalized distributions via a single softmax.

5 Related Work

Many variants of LSTMs (Hochreiter and Schmidhuber, 1997) have been previously explored. These typically consist of a different parameterization of the gates, such as LSTMs with peephole connections (Gers and Schmidhuber, 2000), or a rewiring of the connections, such as GRUs (Cho et al., 2014). However, these modifications invariably maintain the recurrent content layer. Even more systematic explorations (Józefowicz et al., 2015; Greff et al., 2016; Zoph and Le, 2017) do not question the importance of the embedded S-RNN. This is the first study to provide apples-to-apples comparisons between LSTMs with and without the recurrent content layer.

Several other recent works have also reported promising results with recurrent models that are vastly simpler than LSTMs, such as quasi-recurrent neural networks (Bradbury et al., 2016), strongly-typed recurrent neural networks (Balduzzi and Ghifary, 2016), recurrent additive networks (Lee et al., 2017), kernel neural networks (Lei et al., 2017a), and simple recurrent units (Lei et al., 2017b), making it increasingly apparent that LSTMs are over-parameterized. While these works indicate an obvious trend, they do not focus on explaining what LSTMs are learning. In our carefully controlled ablation studies, we propose and evaluate the minimal changes required to test our hypothesis that LSTMs are powerful because they dynamically compute element-wise weighted sums of content layers.

6 Conclusion

We presented an alternate view of LSTMs: they are a hybrid of S-RNNs and a gated model that dynamically computes weighted sums of the S-RNN outputs. Our experiments investigated whether the S-RNN is a necessary component of LSTMs. In other words, are the gates alone as powerful of a model as an LSTM? Results across four major NLP tasks (language modeling, question answering, dependency parsing, and machine translation) indicate that LSTMs suffer little to no performance loss when removing the S-RNN. This provides evidence that the gating mechanism is doing the heavy lifting in modeling context. We further ablate the recurrence in each gate and find that this incurs only a modest drop in performance, indicating that the real modeling power of LSTMs stems from their ability to compute element-wise weighted sums of context-independent functions of their inputs.

This realization allows us to mathematically relate LSTMs and other gated RNNs to attention-based models. Casting an LSTM as a dynamically-computed attention mechanism enables the visualization of how context is used at every timestep, shedding light on the inner workings of the relatively opaque LSTM.

Acknowledgements

The research was supported in part by DARPA under the DEFT program (FA8750-13-2-0019), the ARO (W911NF-16-1-0121), the NSF (IIS-1252835, IIS-1562364), gifts from Google, Tencent, and Nvidia, and an Allen Distinguished Investigator Award. We also thank Yoav Goldberg, Benjamin Heinzerling, Tao Lei, and the UW NLP group for helpful conversations and comments on the work.

References

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. 2017. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. In *ICLR*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *ICLR*.

David Balduzzi and Muhammad Ghifary. 2016. Strongly-typed recurrent neural networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. pages 1292–1300. http://jmlr.org/proceedings/papers/v48/balduzzi16.html.

Yoshua Bengio, Patrice Y. Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks* 5(2):157–166.

James Bradbury, Stephen Merity, Caiming Xiong, and Richard Socher. 2016. Quasi-recurrent neural networks. *CoRR* abs/1611.01576.

Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, Vancouver, Canada, pages 1870–1879. http://aclweb.org/anthology/P17-1171.

Jianpeng Cheng, Li Dong, and Mirella Lapata. 2016. Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 551–561. https://aclweb.org/anthology/D16-1053.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. 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)*. Association for Computational Linguistics, Doha, Qatar, pages 1724–1734. http://www.aclweb.org/anthology/D14-1179.

Timothy Dozat and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing. *CoRR* abs/1611.01734.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science* 14:179–211.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson Liu, Matthew Peters, Michael Schmitz, and Luke Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform. http://allennlp.org/papers/AllenNLP_white_paper.pdf.

Felix A. Gers and Jürgen Schmidhuber. 2000. Recurrent nets that time and count. In *IJCNN*.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2016. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems* .

Luheng He, Kenton Lee, Mike Lewis, and Luke Zettlemoyer. 2017. Deep semantic role labeling: What works and whats next. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Sepp Hochreiter. 1991. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München* 91.

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

Rafal Józefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *ICML*.

Guillaume Klein, Yoon Kim, Yuntian Deng, Jean Senellart, and Alexander M. Rush. 2017. Opennmt: Open-source toolkit for neural machine translation. In *Proc. ACL*. https://doi.org/10.18653/v1/P17-4012.

Kenton Lee, Omer Levy, and Luke Zettlemoyer. 2017. Recurrent additive networks. *arXiv preprint arXiv:1705.07393* .

Tao Lei, Wengong Jin, Regina Barzilay, and Tommi Jaakkola. 2017a. Deriving neural architectures from sequence and graph kernels. In *ICML*.

Tao Lei, Yu Zhang, and Yoav Artzi. 2017b. Training rnns as fast as cnns. *arXiv preprint arXiv:1709.02755* .

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of lstms to learn syntax-sensitive dependencies. *TACL* 4:521–535.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics* 19:313–330.

Ankur Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Austin, Texas, pages 2249–2255. https://aclweb.org/anthology/D16-1244.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*.

Min Joon Seo, Aniruddha Kembhavi, Ali Farhadi, and Hannaneh Hajishirzi. 2016. Bidirectional attention flow for machine comprehension. *CoRR* abs/1611.01603.

Natalia Silveira, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Christopher D. Manning. 2014. A gold standard dependency corpus for English. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *arXiv preprint arXiv:1706.03762* .

Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* .

Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. In *ICLR*.

A Weight Visualization

Given the empirical evidence that LSTMs are effectively learning weighted sums of the content layers, it is natural to investigate what weights the model learns in practice. Using the more mathematically transparent simplification of LSTMs, we can visualize the weights $\boldsymbol{w}_j^t$ that are placed on every input j at every timestep t (see Equation 9).

Unlike attention mechanisms, these weights are vectors rather than scalar values. Therefore, we can only provide a coarse-grained visualization of the weights by rendering their L^2-norm, as shown in Table 5. In the visualization, each column indicates the word represented by the weighted sum, and each row indicates the word over which the weighted sum is computed. Dark horizontal streaks indicate the duration for which a word was remembered. Unsurprisingly, the weights on the diagonal are always the largest since it indicates the weight of the current word. More interesting task-specific patterns emerge when inspecting the off-diagonals that represent the weight on the context words.

The first visualization uses the language model. Due to the language modeling setup, there are only non-zero weights on the current or previous words. We find that the common function words are quickly forgotten, while infrequent words that signal the topic are remembered over very long distances.

The second visualization uses the dependency parser. In this setting, since the recurrent architectures are bidirectional, there are non-zero weights on all words in the sentence. The top-right triangle indicates weights from the forward direction, and the bottom-left triangle indicates from the backward direction. For syntax, we see a significantly different pattern. Function words that are useful for determining syntax are more likely to be remembered. Weights on head words are also likely to persist until the end of a constituent.

This illustration provides only a glimpse into what the model is capturing, and perhaps future, more detailed visualizations that take the individual dimensions into account can provide further insight into what LSTMs are learning in practice.

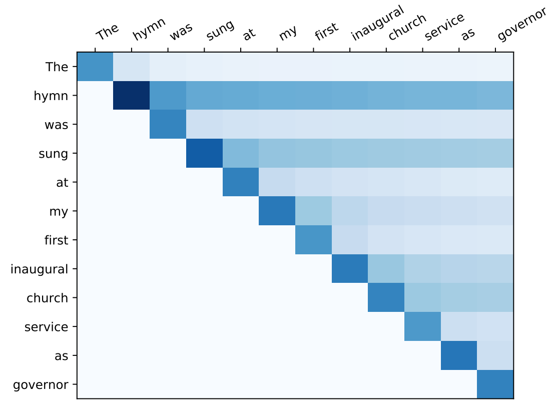
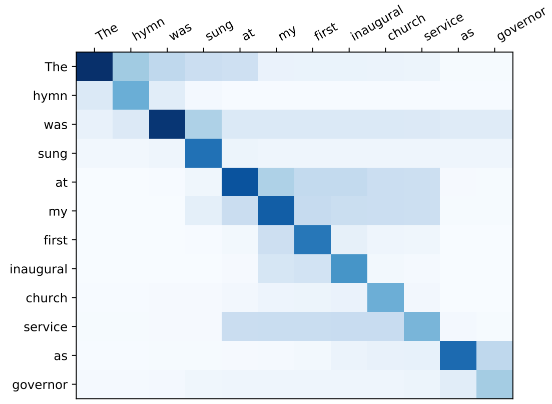
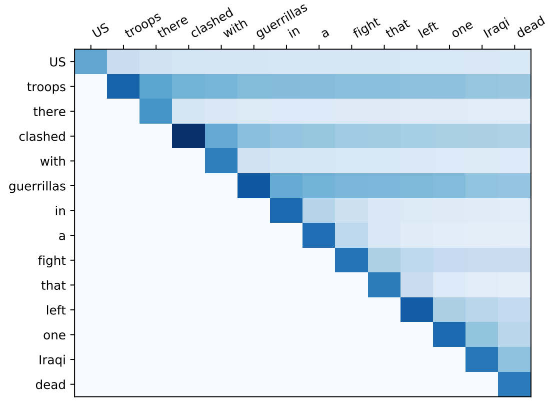
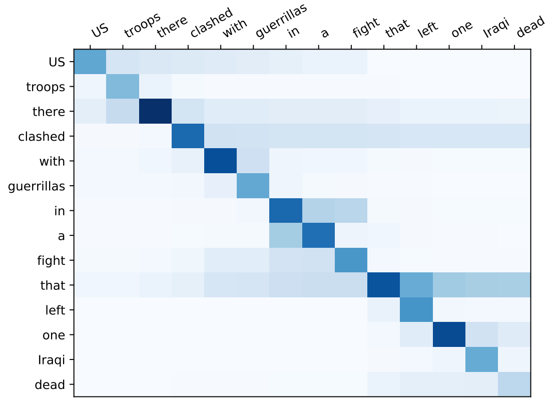
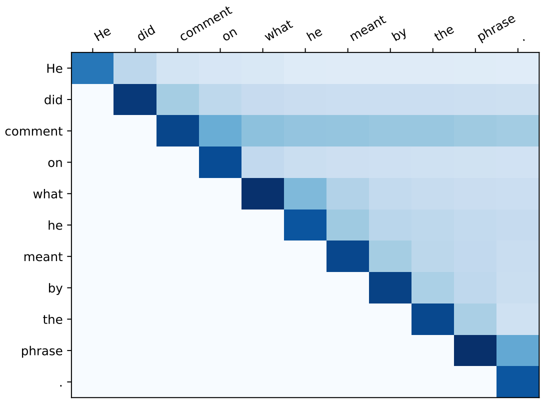
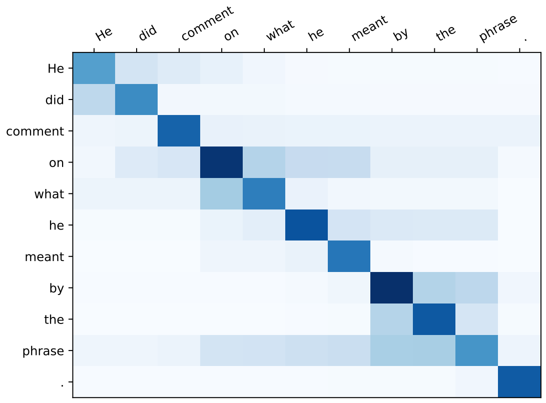
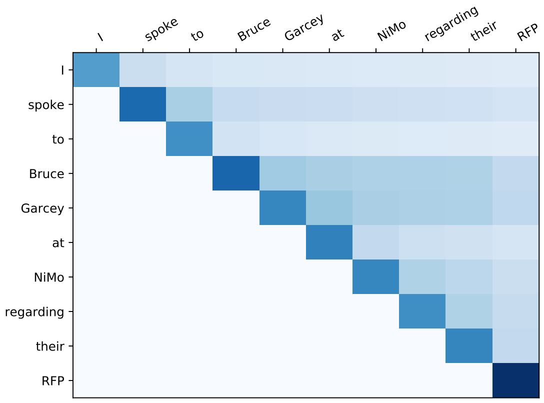
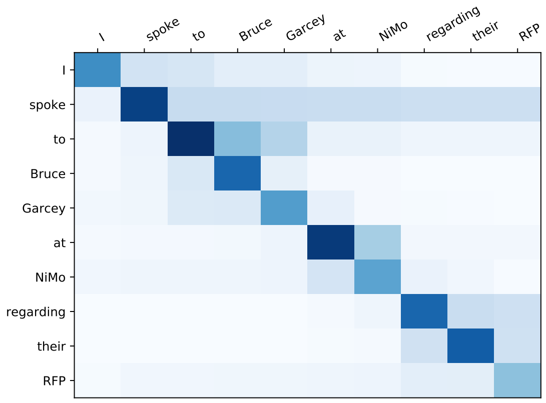
Language model weights	Dependency parser weights
	
	
	
	

Table 5: Visualization of the weights on context words learned by the memory cell. Each column represents the current word t, and each row represents a context word j. The gating mechanism implicitly computes element-wise weighted sums over each column. The darkness of each square indicates the L^2-norm of the vector weights $\boldsymbol{w}_j^t$ from Equation 9. Figures on the left show weights learned by a language model. Figures on the right show weights learned by a dependency parser.