LSTMs Exploit Linguistic Attributes of Data

Nelson F. Liu[♠◇] **Omer Levy**[♠] **Roy Schwartz**[♠♣] **Chenhao Tan**[♡] **Noah A. Smith**[♠]
[♠]Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, WA, USA
[◇]Department of Linguistics, University of Washington, Seattle, WA, USA
[♣]Allen Institute for Artificial Intelligence, Seattle, WA, USA
[♡]Department of Computer Science, University of Colorado, Boulder, CO, USA
{nfliu,omerlevy,roysch,nasmith}@cs.washington.edu,
chenhao.tan@colorado.edu

Abstract

While recurrent neural networks have found success in a variety of natural language processing applications, they are general models of sequential data. We investigate how the properties of natural language data affect an LSTM's ability to learn a nonlinguistic task: recalling elements from its input. We find that models trained on natural language data are able to recall tokens from much longer sequences than models trained on non-language sequential data. Furthermore, we show that the LSTM learns to solve the memorization task by explicitly using a subset of its neurons to count timesteps in the input. We hypothesize that the patterns and structure in natural language data enable LSTMs to learn by providing approximate ways of reducing loss, but understanding the effect of different training data on the learnability of LSTMs remains an open question.

1 Introduction

Recurrent neural networks (RNNs; Elman, 1990), especially variants with gating mechanisms such as long short-term memory units (LSTM; Hochreiter and Schmidhuber, 1997) and gated recurrent units (GRU; Cho et al., 2014), have significantly advanced the state of the art in many NLP tasks (Mikolov et al., 2010; Vinyals et al., 2015; Bahdanau et al., 2015, among others). However, RNNs are general models of sequential data; they are not explicitly designed to capture the unique properties of language that distinguish it from generic time series data.

In this work, we probe how linguistic properties such as the hierarchical structure of language (Everaert et al., 2015), the dependencies between tokens, and the Zipfian distribution of token frequencies (Zipf, 1935) affect the ability of LSTMs to learn. To do this, we define a simple memorization task where the objective is to recall the identity of the token that occurred a fixed number of timesteps in the past, within a fixed-length input. Although the task itself is not linguistic, we use it because (1) it is a generic operation that might form part of a more complex function on arbitrary sequential data, and (2) its simplicity allows us to unfold the mechanism in the trained RNNs.

To study how linguistic properties of the training data affect an LSTM's ability to solve the memorization task, we consider several training regimens. In the first, we train on data sampled from a uniform distribution over a fixed vocabulary. In the second, the token frequencies have a Zipfian distribution, but are otherwise independent of each other. In another, the token frequencies have a Zipfian distribution but we add Markovian dependencies to the data. Lastly, we train the model on natural language sequences. To ensure that the models truly memorize, we evaluate on uniform samples containing *only rare words*.[1]

We observe that LSTMs trained to perform the memorization task on natural language data or data with a Zipfian distribution are able to memorize from sequences of greater length than LSTMs trained on uniformly-sampled data. Interestingly, increasing the length of Markovian dependencies in the data does not significantly help LSTMs to learn the task. We conclude that linguistic properties can help or even enable LSTMs to learn the memorization task. Why this is the case remains an open question, but we propose that the additional structure and patterns within natural language data provide additional noisy, approximate

[1]This distribution is adversarial with respect to the Zipfian and natural language training sets.

paths for the model to minimize its loss, thus offering more training signal than the uniform case, in which the only way to reduce the loss is to learn the memorization function.

We further inspect *how* the LSTM solves the memorization task, and find that some hidden units count the number of inputs. Shi et al. (2016a) analyzed LSTM encoder-decoder translation models and found that similar counting neurons regulate the length of generated translations. Since LSTMs better memorize (and thus better count) on language data than on non-language data, and counting plays a role in encoder-decoder models, our work could also lead to improved training for sequence-to-sequence models in non-language applications (e.g., Schwaller et al., 2017).

2 The Memorization Task

To assess the ability of LSTMs to retain and use information, we propose a simple memorization task. The model is presented with a sequence of tokens and is trained to recall the identity of the middle token.[2] We predict the middle token since predicting items near the beginning or the end might enable the model to avoid processing long sequences (e.g., to perfectly memorize the last token, simply set the forget gate to 0 and the input gate to 1).[3] All input sequences at train and test time are of equal length. To explore the effect of sequence length on LSTM task performance, we experiment with different input sequence lengths (10, 20, 40, 60, ..., 300).

3 Experimental Setup

We modify the linguistic properties of the training data and observe the effects on model performance. Further details are found in Appendix A, and we release code for reproducing our results.[4]

Model. We train an LSTM-based sequence prediction model to perform the memorization task. The model embeds input tokens with a randomly initialized embedding matrix. The embedded inputs are encoded by a single-layer LSTM and the final hidden state is passed through a linear projection to produce a probability distribution over the vocabulary.

Our goal is to evaluate the memorization ability of the LSTM, so we freeze the weights of the embedding matrix and the linear output projection during training. This forces the model to rely on the LSTM parameters (the only trainable weights), since it cannot gain an advantage in the task by shifting words favorably in either the (random) input or output embedding vector spaces. We also tie the weights of the embeddings and output projection so the LSTM can focus on memorizing the timestep of interest rather than also transforming input vectors to the output embedding space.[5] Finally, to examine the effect of model capacity on memorization ability, we experiment with different hidden state size values.

Datasets. We experiment with several distributions of training data for the memorization task. In all cases, a 10K vocabulary is used.

- In the *uniform* setup, each token in the training dataset is randomly sampled from a uniform distribution over the vocabulary.
- In the *unigram* setup, we modify the *uniform* data by integrating the Zipfian token frequencies found in natural language data. The input sequences are taken from a modified version of the Penn Treebank (Marcus et al., 1993) with randomly permuted tokens.
- In the *5gram*, *10gram*, and *50gram* settings, we seek to augment the *unigram* setting with Markovian dependencies. We generate the dataset by grouping the tokens of the Penn Treebank into 5, 10, or 50-length chunks and randomly permuting these chunks.
- In the *language* setup, we assess the effect of using real language. The input sequences here are taken from the Penn Treebank, and thus this setup further extends the *5gram*, *10gram*, and *50gram* datasets by adding the remaining structural properties of natural language.

We evaluate each model on a test set of uniformly sampled tokens from the 100 rarest words in the vocabulary. This evaluation setup ensures that, regardless of the data distribution the models were trained on, they are forced to generalize in

[2] Or the $(\frac{n}{2}+1)$th token if the sequence length n is even.

[3] We experimented with predicting tokens at a range of positions, and our results are not sensitive to the choice of predicting *exactly* the middle token.

[4] http://nelsonliu.me/papers/lstms-exploit-linguistic-attributes/

[5] Tying these weights constrains the embedding size to always equal the LSTM hidden state size.

order to perform well on the test set. For instance, in a test on data with a Zipfian token distribution, the model may do well by simply exploiting the training distribution (e.g., by ignoring the long tail of rare words).

4 Results

We first observe that, in every case, the LSTM is able to perform the task perfectly (or nearly so), up to some input sequence length threshold. Once the input sequence length exceeds this threshold, performance drops rapidly.

How does the training data distribution affect performance on the memorization task? Figure 1 compares memorization performance of an LSTM with 50 hidden units on various input sequence lengths when training on each of the datasets. Recall that the test set of only rare words is fixed for each length, regardless of the training data. When trained on the *uniform* dataset, the model is perfect up to length 10, but does no better than the random baseline with lengths above 10. Training on the *unigram* setting enables the model to memorize from longer sequences (up to 20), but it begins to fail with input sequences of length 40; evaluation accuracy quickly falls to 0.[6] Adding Markovian dependencies to the *unigram* dataset leads to small improvements, enabling the LSTM to successfully learn on inputs of up to length 40 (in the case of *5gram* and *10gram*) and inputs of up to length 60 (in the case of *50gram*). Lastly, training on *language* significantly improves model performance, and it is able to perfectly memorize with input sequences of up to 160 tokens before any significant degradation. These results clearly indicate that training on data with linguistic properties helps the LSTM learn the non-linguistic task of memorization, even though the test set has an adversarial non-linguistic distribution.

How does adding hidden units affect memorization performance? Figure 2 compares memorization performance on each dataset for LSTMs with 50, 100, and 200 hidden units. When training on the *uniform* dataset, increasing the number of LSTM hidden units (and thus also the embedding size) to 100 or 200 does not help it memorize longer sequences. Indeed, even at 400 and 800 we saw no improvement (not shown in Figure 2). When training on any of the other datasets, adding more hidden units eventually leads to perfect memorization for all tested input sequence lengths. We take these results as a suggestion that successful learning for this task requires sufficiently high capacity (dimensionality in the LSTM). The capacity need is diminished when the training data is linguistic, but LSTMs trained on the *uniform* set cannot learn the memorization task even given high capacity.

[6] Manual inspection of the trained models reveals that they predict the most frequent words in the corpus. Since the evaluation set has only the 100 rarest types, performance (0% accuracy) is actually worse than in the *uniform* setting.

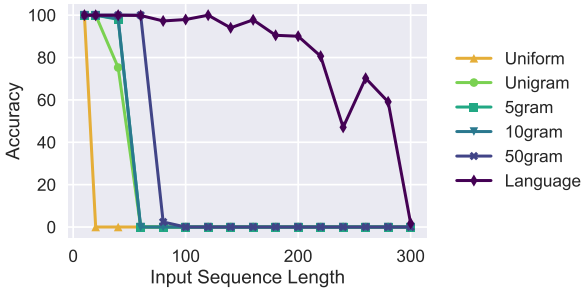

Figure 1: Test set accuracy of LSTMs with 50 hidden units trained on the *uniform*, *∗gram*, and *language* datasets with various input sequence lengths. *5gram* and *10gram* perform nearly identically, so the differences may not be apparent in the figure. *unigram* accuracy plateaus to 0, and *uniform* accuracy plateaus to ≈0.01% (random baseline). Best viewed in color.

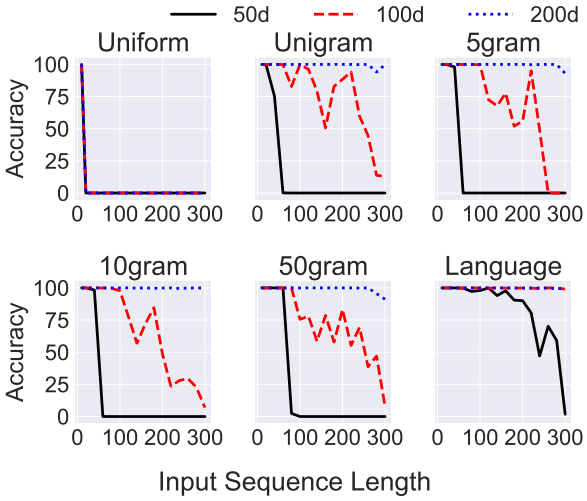

Figure 2: Test set accuracy of LSTMs with 50, 100 or 200 hidden units trained on each dataset with various input sequence lengths.

5 Analysis

Throughout this section, we analyze an LSTM with 100 hidden units trained with the *language* setting with an input sequence length of 300. This setting is a somewhat closer simulation of current NLP models, since it is trained on real language and recalls perfectly with input sequence lengths of 300 (the most difficult setting tested).

How do LSTMs solve the memorization task? A simple way to solve the memorization task is by counting. Since all of the input sequences are of equal length and the timestep to predict is constant throughout training and testing, a successful learner could maintain a counter from the start of the input to the position of the token to be predicted (the middle item). Then, it discards its previous cell state, consumes the label's vector, and maintains this new cell state until the end of the sequence (i.e., by setting its forget gate near 1 and its input gate near 0).

While LSTMs clearly have the expressive power needed to count and memorize, whether they can learn to do so from data is another matter. Past work has demonstrated that the LSTMs in an encoder-decoder machine translation model learn to increment and decrement a counter to generate translations of proper length (Shi et al., 2016a) and that representations produced by autoencoding LSTMs contain information about the input sequence length (Adi et al., 2017). Our experiments isolate the counting aspect from other linguistic properties of translation and autoencoding (which may indeed be correlated with counting), and also test this ability with an adversarial test distribution and much longer input sequences.

We adopt the method of Shi et al. (2016a) to investigate whether LSTMs solve the memorization task by learning to count. We identify the neurons that best predict timestep information by fitting a linear regression model to predict the number of inputs seen from the hidden unit activation. When evaluating on the test set, we observe that the LSTM cell state as a whole is very predictive of the timestep, with $R^2 = 0.998$.

While no single neuron perfectly records the timestep, several of them are strongly correlated. In our model instance, neuron 77 has the highest correlation ($R^2 = 0.919$), and neuron 61 is next ($R^2 = 0.901$). The activations of these neurons over time for a random correctly classified test input linearly increase up to the target token, after which the activations falls to nearly 0 (Figure 3).

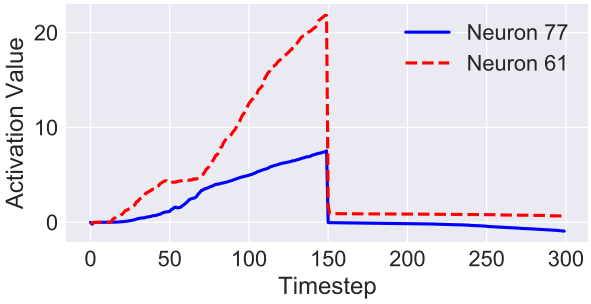

Figure 3: Activations of the neurons at indices 77 and 61 over time, showing counter-like behavior up to the target timestep to be remembered.

One hypothesis for why linguistic data helps. During training, the LSTM must: (1) determine what the objective is (here, "remember the middle token") and (2) adjust its weights to minimize loss. We observed that adding hidden units to LSTMs trained on *unigram* or *language* sets improves their ability to learn from long input sequences, but does not affect LSTMs trained on the *uniform* dataset. One explanation for this disparity is that LSTMs trained on *uniform* data are simply not learning what the task is—they do not realize that the label always matches the token in the middle of the input sequence, and thus they cannot properly optimize for the task, even with more hidden units. On the other hand, models trained on *unigram* or *language* can determine that the label is always the middle token, and can thus learn the task. Minimizing training loss ought to be easier with more parameters, so adding hidden units to LSTMs trained on data with linguistic attributes increases the length of input sequences that they can learn from.

But why might LSTMs trained on data with linguistic attributes be able to effectively learn the task for long input sequences, whereas LSTMs trained on the *uniform* dataset cannot? We conjecture that linguistic data offers more reasonable, if approximate, pathways to loss minimization, such as counting frequent words or phrases. In the *uniform* setting, the model has only one path to success: true memorization, and it cannot find an effective way to reduce the loss. In other words, linguistic structure and the patterns of language may provide additional signals that correlate with the label and facilitate learning the memorization task.

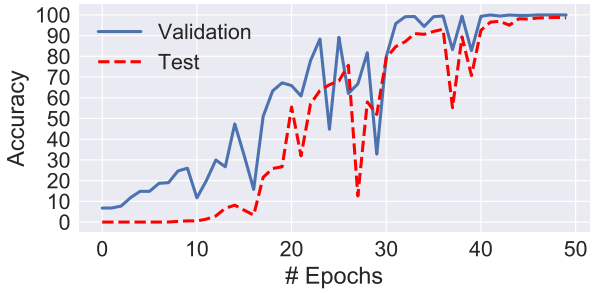

Figure 4: Model validation and test accuracy over time during training. Validation improves faster than test, indicating that the model exploits linguistic properties of the data during training.

Figure 4 shows that models trained on the *unigram* and *language* datasets converge to high validation accuracy faster than high test accuracy. This suggests that models trained on data with linguistic attributes first learn to do well on the training data by exploiting the properties of language and not truly memorizing. Perhaps the model generalizes to actually recalling the target token later, as it refines itself with examples from the long tail of infrequent tokens.

Figure 4 may show this shift from exploiting linguistic properties to true memorization. The validation and test accuracy curves are quite synchronized from epoch 37 onward, indicating that the model's updates affect both sets identically. The model clearly learns a strategy that works well on both datasets, which strongly suggests that it has learned to memorize. In addition, when the model begins to move toward true memorization, we'd expect validation accuracy to momentarily falter as it moves away from the crutches of linguistic features—this may be the dip at around epoch 35 from perfect validation accuracy to around 95% accuracy.

6 Related Work

To our knowledge, this work is the first to study how linguistic properties in training data affect the ability of LSTMs to learn a general, non-linguistic, sequence processing task.

Previous studies have sought to better understand the empirical capabilities of LSTMs trained on natural language data. Linzen et al. (2016) measured the ability of LSTMs to learn syntactic long range dependencies commonly found in language, and Gulordava et al. (2018) provide evidence that LSTMs can learn the hierarchical structure of language. Blevins et al. (2018) show that the internal representations of LSTMs encode syntactic information, even when trained without explicit syntactic supervision.

Also related is the work of Weiss et al. (2018), who demonstrate that LSTMs are able to count infinitely, since their cell states are unbounded, while GRUs cannot count infinitely since the activations are constrained to a finite range. One avenue of future work could compare the performance of LSTMs and GRUs on the memorization task.

Past studies have also investigated what information RNNs encode by directly examining hidden unit activations (Karpathy et al., 2016; Li et al., 2016; Shi et al., 2016a, among others) and by training an auxiliary classifier to predict various properties of interest from hidden state vectors (Shi et al., 2016b; Adi et al., 2017; Belinkov et al., 2017, among others).

7 Conclusion

In this work, we examine how linguistic attributes in training data can affect an LSTM's ability to learn a simple memorization task. We find that LSTMs trained on uniformly sampled data are only able to learn the task with the sequence length of 10, whereas LSTMs trained with language data are able to learn on sequences of up to 300 tokens.

We further investigate how the LSTM learns to solve the task, and find that it uses a subset of its hidden units to track timestep information. It is still an open question why LSTMs trained on linguistic data are able to learn the task whereas LSTMs trained on uniformly sampled data cannot; based on our observations, we hypothesize that the additional patterns and structure in language-based data may provide the model with approximate paths of loss minimization, and improve LSTM trainability as a result.

Acknowledgments

We thank the ARK as well as the anonymous reviewers for their valuable feedback. NL is supported by a Washington Research Foundation Fellowship and a Barry M. Goldwater Scholarship. This work was supported in part by a hardware gift from NVIDIA Corporation and a UW High Performance Computing Club Cloud Credit Award.

References

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

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

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James Glass. 2017. What do neural machine translation models learn about morphology? In *Proc. of ACL*.

Terra Blevins, Omer Levy, and Luke Zettlemoyer. 2018. Deep RNNs learn hierarchical syntax. In *Proc. of ACL*.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Proc. of EMNLP*.

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

Martin B.H. Everaert, Marinus A.C. Huybregts, Noam Chomsky, Robert C. Berwick, and Johan J. Bolhuis. 2015. Structures, not strings: linguistics as part of the cognitive sciences. *Trends in Cognitive Sciences*, 19(12):729–743.

Kristina Gulordava, Piotr Bojanowski, Edouard Grave, Tal Linzen, and Marco Baroni. 2018. Colorless green recurrent networks dream hierarchically. In *Proc. of NAACL*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*, 9 8:1735–80.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2016. Visualizing and understanding recurrent networks. In *Proc. of ICLR*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: a method for stochastic optimization. In *Proc. of ICLR*.

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2016. Visualizing and understanding neural models in NLP. In *Proc. of NAACL*.

Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. 2016. Assessing the ability of LSTMs to learn syntax-sensitive dependencies. *Transactions of the Association of Computational Linguistics*, 4(1):521–535.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. 1993. Building a large annotated corpus of English: the Penn Treebank. *Computational Linguistics*, 19(2):313–330.

Tomáš Mikolov, Martin Karafiát, Lukás Burget, Jan Cernocký, and Sanjeev Khudanpur. 2010. Recurrent neural network based language model. In *Proc. of INTERSPEECH*.

Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. 2011. Extensions of recurrent neural network language model. In *Proc of. ICASSP*.

Philippe Schwaller, Theophile Gaudin, David Lanyi, Costas Bekas, and Teodoro Laino. 2017. "found in translation": predicting outcomes of complex organic chemistry reactions using neural sequence-to-sequence models. In *Proc. of NIPS Machine Learning for Molecules and Materials Workshop*.

Xing Shi, Kevin Knight, and Deniz Yuret. 2016a. Why neural translations are the right length. In *Proc. of EMNLP*.

Xing Shi, Inkit Padhi, and Kevin Knight. 2016b. Does string-based neural MT learn source syntax? In *Proc. of EMNLP*.

Oriol Vinyals, Lukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2015. Grammar as a foreign language. In *Proc. of NIPS*.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2018. On the practical computational power of finite precision RNNs for language recognition. In *Proc. of ACL*.

George Kingsley Zipf. 1935. *The Psycho-biology of Language*. Houghton, Mifflin.

Appendices

A Experimental Setup Details

Penn Treebank Processing Our experiments use a preprocessed version of the Penn Treebank commonly used in the language modeling community and first introduced by Mikolov et al. (2011). This dataset has 10K types, hence why we use this vocabulary size for all experiments. We generate examples by concatenating the sentences together and taking subsequences of the desired input sequence length.

Training The model is trained end-to-end to directly predict the tokens at a particular timestep in the past; it is optimized with Adam (Kingma and Ba, 2015) with an initial learning rate of 0.001, which is halved whenever the validation dataset (a held-out portion of the training dataset) loss fails to improve for three consecutive epochs. The model is trained for a maximum of 240 epochs or until it converges to perfect validation performance. We do not use dropout; we included it in initial experiments, but it severely hampered model performance and does not make much sense for a task where the goal is to explicitly memorize. We ran each experiment three times with different random seeds and evaluate the model with the highest validation accuracy on the test set. We take the best since we are interested in whether the LSTMs *can* be trained for the task.