Modeling Long-Distance Dependencies with Second-Order LSTMs

Stanford CS224N Custom Project

Richard Lin
Department of Computer Science
Stanford University
yifengl@stanford.edu

Noa Bendit-Shtull
Department of Statistics
Stanford University
nbshtull@stanford.edu

Sam Spinner
Symbolic Systems Program
Stanford University
spinners@stanford.edu

Abstract

A second-order LSTM is a modified LSTM architecture that conditionally selects one of multiple weight matrices to apply to each input, based on that input. Existing research on second-order RNNs focuses on character-level language modeling. Our task is to develop a second-order LSTM for word-level language models. Specifically, we test the hypothesis that second-order LSTM architectures have the improved ability (relative to a simple LSTM) to learn long distance dependencies within an input. We find that, although second-order models don't exhibit quantitative improvement, they can be trained in fewer epochs and are able to recover from unexpected inputs more quickly.

1 Introduction

One of the core challenges in NLP research is modeling the long-range dependencies in human language. Humans naturally have the ability to draw connections across sentences, paragraphs, and pages, but these links are more challenging for computers to model. In the basic RNN, the vanishing gradient problem limits a language models' memory; a range of models, such as LSTM and GRU, have been developed to reduce this friction.

In order to preserve long-term memory, these models must determine which information to store from previous hidden states. This introduces a related problem–if the model chooses the wrong information to carry in its long-term memory, it may be difficult to recover. As Kraus et al. explain, "If the RNN's hidden representation remembers the wrong information and reaches a bad numerical state for predicting future sequence elements, for instance as a result of an unexpected input, it may take many time-steps to recover" [1, p. 1].

This paper contributes to a small body of existing work focused on developing new architectures to improve RNN performance on modeling long-range dependencies; that is, to help RNNs remember the right information. Our goal is to assess whether a second-order LSTM – an LSTM that routes different inputs to different LSTM cells – can make progress in this research area.

2 Related Work

One of the earlier second-order solutions to RNN memory loss is the tensor RNN, which has a separate hidden-to-hidden weight matrix W_{hh} for each input dimension. If the input x_t has dimension M, we store M weight matrices $\{W_{hh}^{(1)}, \ldots, W_{hh}^{(M)}\}$. If the inputs x_t are one-hot representations of words or characters, then only one hidden-to-hidden weight matrix is used for each input.

The major drawback of this approach is the number of parameters required. The multiplicative RNN (mRNN) (Sutskever et al., 2011) solves this problem by allowing parameter sharing. Like the tensor RNN, the mRNN modifies vanilla RNN by making the hidden-to-hidden transition matrices input-dependent. However, mRNN limits the number of parameters by factorizing the hidden-to-hidden transition matrix with an intermediate diagonal matrix that is input-dependent. The hidden-to-hidden

matrix $W_{hh}^{(x_t)}$ is decomposed as $W_{hh}^{(x_t)} = W_{hn} \cdot \text{diag}(W_{nx} x_t) \cdot W_{nh}$ where the dimension n is a parameter of the model. Sutskever et al. find that, given the same number of parameters, the mRNN achieved lower bits per char than a standard RNN. [2]

However, Kraus et al. note that the mRNN does not outperform the LSTM, and hypothesize that the mRNN doesn't provide an avenue for retaining information over the long term. To handle this drawback, they propose a combination of the mRNN and LSTM architectures, the multiplicative LSTM (mLSTM). They suggest that the two architectures are complementary because the LSTM controls the flow of information through the network, while the mRNN allows input-dependent transitions. The authors compare their mLSTM to existing regular LSTM, stacked LSTM, and RNN models in a series of character-level language modeling tasks. The results showed that the mLSTM outperformed existing baselines on several benchmark datasets, demonstrating that large recursion depth is not necessary for successful results; the mLSTM uses only two recurrent transition matrices, while competing models use as many as ten layers. The reduced depth is valuable because it allows the mLSTM to be parallelized more easily. [1]

However, the mLSTM experiments only encompass character-level language modeling. It's not clear that the mLSTM model is easily extensible to word-level models, given that the number of parameters to be trained scales with the dimensionality of the inputs. We aim to extend this body of research by using an attention mechanism to learn the appropriate LSTM cell for each input.

Another set of research analyzes long term dependencies in formal languages, which have more predictable patterns than natural language. John Hewitt's paper (currently under review), shows that a one-layer LSTM with $3m$ hidden units can successfully model a formal syntax composed of parentheses. Specifically, an LSTM can capture an m-bounded Dyck-k language by implementing a stack in its hidden layer. Empirically, he finds that an LSTM is successful, but requires much more data to make predictions for larger values of m. [3]

3 Approach

Our project is divided into two stages, a proof-of-concept and an application to natural language. The proof-of-concept uses synthetic language, specifically the family of m-bounded Dyck-k (in this case, Dyck-2) languages. The second stage of the project extends the analysis to natural language, where successful modeling of long-distance dependency is more difficult to quantify.

3.1 Baseline Model

We implement the simple LSTM in John Hewitt's paper [3] and use it as our baseline. This baseline model consists of a single layer LSTM and the hidden states at each time step are passed through a fully connected layer to generate the predicted probability distribution across all vocabulary words. The baseline model is based on the LSTM model from public repository [4] but we modified it extensively to integrate with our framework.

3.2 Second-order Models

We constructed two original second-order models, an assignment-based model and an attention-based model. An S-dimensional second-order model requires S transition weight matrices $W_{hh}^{(s)}$ for $s \in \{1, \ldots, S\}$, where each $W_{hh}^{(s)}$ remembers information about specific inputs. We represent these transition matrices using S LSTM cells, with each LSTM cell containing one $W_{hh}^{(s)}$.

3.2.1 Assignment Model

For the assignment-based second-order LSTM model, each word in the vocabulary is deterministically mapped to an LSTM cell. For the m-bounded Dyck-k datasets, we have two LSTM cells to store information about two types of parenthesis in the data. We assign all type a parentheses to one LSTM cell and all type b parentheses to the other. For the natural language datasets, we tried to assign words to LSTM cells based on word clustering. We experimented with t_SNE, PCA, and KMeans to cluster the GloVe embedding space, with K equal to the number of LSTM cells.

However, we abandoned the assignment-based approach for two reasons. First, clustering GloVe vectors was computationally slow and expensive, and had to be rerun to cluster the vocabulary of each

new dataset. More importantly, using pre-trained word vectors and predetermined input assignments introduced *supervision* to our model, making it incomparable to the baseline. Therefore, we decided to continue with an unsupervised attention model.

3.2.2 Attention Model

In the attention-based second-order LSTM model, the attention mechanism sends each input to all LSTM cells, and uses learned attention scores to compute a linear combination of the hidden and cell state outputs. Specifically, an S-dimensional second-order LSTM has attention matrix V with dimension $S \times embedding\_dim$. For each of the LSTM cells, we compute the intermediate hidden state and memory cell state $h_t^{(s)}$ and $c_t^{(s)}$ (represented in Figure 1 as joint $\hat{h}_t$), and then use attention to compute a linear combination of the intermediate states to get the new states h_t and c_t, as follows:

$$e_t = Vx_t \tag{a}$$

$$\alpha_t = \text{softmax}\left(\frac{e_t}{\tau}\right) \tag{b}$$

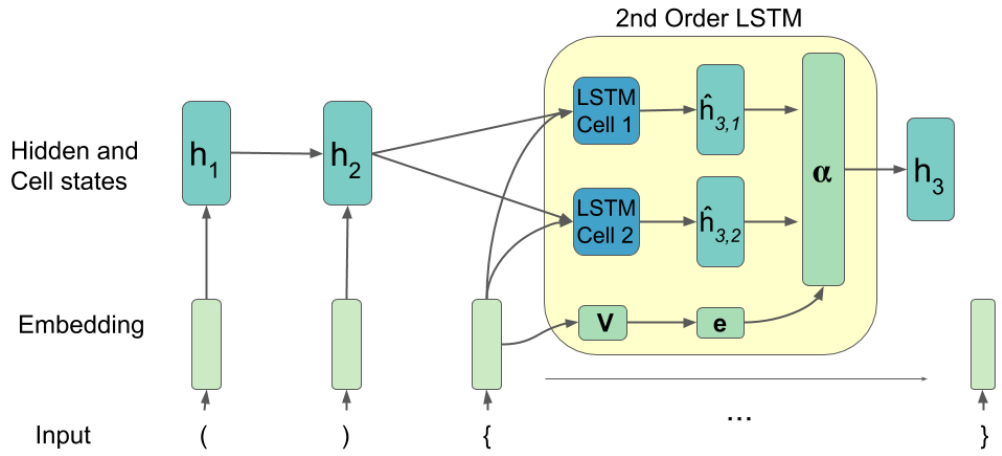
$$h_t^{(s)}, c_t^{(s)} = \text{LSTMCell}^{(s)}(x_t, (h_{t-1}, c_{t-1})) \tag{c}$$

$$h_t = \sum_{s=1}^{S} \alpha_{t,s} \cdot h_t^{(s)} \tag{d}$$

$$c_t = \sum_{s=1}^{S} \alpha_{t,s} \cdot c_t^{(s)} \tag{e}$$

τ is a temperature parameter that affects the outcome of the softmax function. Temperature is initialized "hot" with $\tau = 1$ (making (b) effectively a normal softmax), but then decreases with each epoch by a constant multiplier. The smaller the value of τ, the more probability mass is put on one LSTM cell. When $\tau \approx 0.1$, (b) is effectively one-hot. We decay the temperature throughout training so that eventually each input word gets assigned to one particular LSTM cell and the hidden state output by that cell is effectively the next hidden state.

Figure 1: Second-order LSTM architecture



4 Experiments

4.1 Data

Synthetic Language (Parentheses)

The m-bounded Dyck-k data are comprised of k unique types of parentheses, with at most m unclosed parentheses appearing at any time. This formal language is useful as a proof of concept because the

relationship between an open parenthesis and its corresponding close parenthesis is well-defined. This allows us to test our model's ability to predict long-term dependencies. The data for $m = 4, 6, 8$ was provided by John Hewitt [3]. The training datasets and test datasets each contain 10,000 samples, and the validation datasets contain 4,000 samples.

Natural Language (WikiText-2)

The WikiText-2 dataset [5], which contains 2 million training tokens and a vocabulary of 33k words, is publicly available. The training set contains 37k words, and the validation and test set contain approximately 4k words each. We use this data to test our model's performance on natural language modeling tasks.

For the natural language model, we load the raw data as a continuous stream; the sequence length is determined by the BPTT parameter, and the batches are constructed so that the hidden state is transferable from one batch to the next. Specifically, each index in a batch is a continuation of the same index in the previous batch.

4.2 Evaluation method

We use two quantitative evaluation metrics, perplexity and parenthesis prediction accuracy. The formula for parenthesis prediction accuracy is from John's paper submission [3]. The prediction accuracy is measured only for close parentheses, since there is no reason to favor one open parenthesis over another. A prediction is considered correct if at least 80% of the probability mass on *any* close parenthesis is placed on the true close parenthesis. The distance between an open parenthesis and its corresponding close parenthesis is called closing distance. The long distance prediction accuracy (LDPA) is calculated separately for each possible closing distance. We also report the worst-case prediction accuracy (WCPA), which is the minimum prediction accuracy over all distances.

4.3 Experimental details

For all datasets, the initial LSTM hidden state is initialized to zero. The models are all trained using an Adam optimizer, with a learning rate of 1e-4. The model is trained with early stopping after 6 epochs of increasing validation perplexity. We used learning rate decay of 0.5, with patience parameter 3. For training the attention models, the temperature parameter τ is initialized to 1, and decays with a multiplier of 0.9. For testing, temperature is set arbitrarily low.

For the m-bounded Dyck-2 data, we use word embeddings of size 30, an LSTM hidden layer of size $3m$, and and batch size 10. No dropout is used. The attention models are trained with two LSTM cells.

For the WikiText-2 data, we use word embeddings of size 300, and an LSTM hidden layer of size 600. Batch size is 64, and the BPTT value is 70. We use dropout of 0.5 on both the input and output. The attention models are trained with both two and five LSTM cells.

4.4 Results

Table 1 shows the test results for the baseline model and the second-order LSTM on each of the synthetic languages. Figure 2 shows the relationship between closing distances and long-distance prediction accuracy for the baseline model on 4, 6, and 8-bounded Dyck-2 datasets.

Table 1 shows that the attention model achieves marginally lower perplexity than the baseline model across $m4$, $m6$, and $m8$. The attention model slightly under-performs relative to the baseline model on WCPA for the $m4$, but performs slightly better on the test set for $m6$ and $m8$. In Figure 2, the LDPA for the $m8$ baseline is zero near distance 300, which is why the WCPA for $m8$ baseline is zero, while the WCPA for the attention model is 0.67.

Overall, the attention model performs similarly to the baseline model on the synthetic data. This is as expected, because John's research showed that a single LSTM can fully capture the behavior of a stack. Although it did not show quantitative improvement over the baseline, our attention model is much more stable during training than the baseline model. The validation WCPA for the baseline model oscillates drastically throughout training but WCPA for the attention model does not. The attention model also trains in fewer epochs than the baseline model, as shown in Figure 3.

Table 1: Results for baseline and second-order models on m-bounded Dyck-2 datasets. For perplexity, lower is better, and for WCPA higher is better.

			Perplexity		WCPA	
Data	Model	Params	Val.	Test	Val.	Test
4-bounded Dyck-2	Baseline LSTM	2.1M	2.391	2.393	1	1
4-bounded Dyck-2	Attention LSTM	4.1M	2.386	2.387	0.994	0.999
6-bounded Dyck-2	Baseline LSTM	3.5M	2.534	2.533	0.98	0.857
6-bounded Dyck-2	Attention LSTM	7M	2.530	2.530	0.929	0.875
8-bounded Dyck-2	Baseline LSTM	5.3M	2.618	2.617	0	0
8-bounded Dyck-2	Attention LSTM	8.6M	2.612	2.611	0	0.667

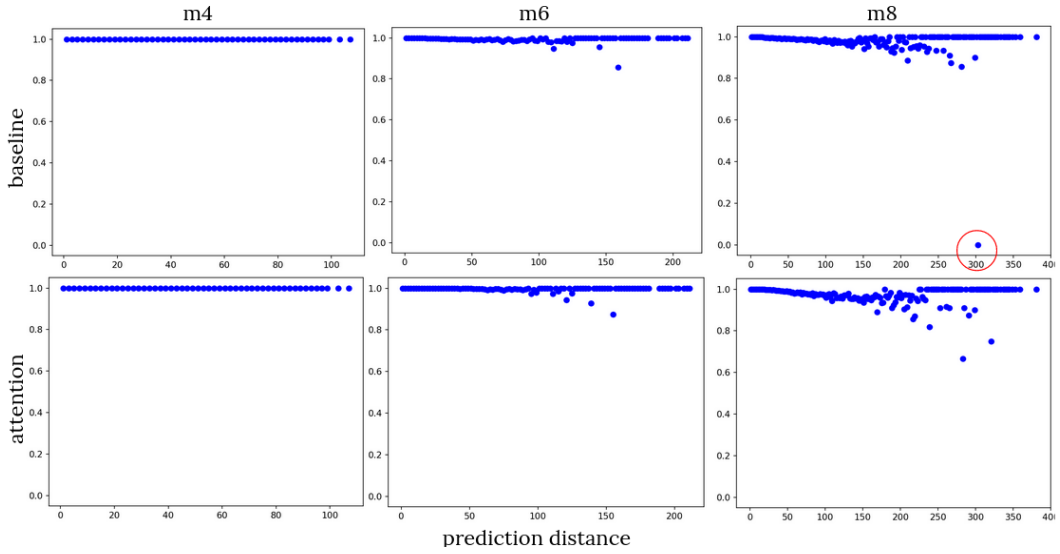


Figure 2: LDPA for Dyck-2 language models trained with the baseline LSTM and second-order LSTM. The horizontal axis is the closing distance between open and close parentheses, and the vertical axis is prediction accuracy.

Table 2 shows the validation and test results on the WikiText-2 dataset. The attention model with 2 cells and 5 cells both outperform the baseline single-cell model by a significant margin. We suspect the attention model is able to perform better on the WikiText-2 dataset than the parentheses dataset because there are more intricate dependencies between words. The long-distance relationship between parentheses can be captured by just a stack, whereas the relationship between words are harder to capture. It is also possible that the the significant increase in the number of parameters between the baseline and the attention model is responsible for the improved performance.

Table 2: Results for baseline LSTM and second-order LSTM on the WikiText-2 dataset

				Perplexity	
Dataset	Model	Cells	Params	Val.	Test
WikiText-2	Baseline LSTM	1	2.2M	157.0	147.2
WikiText-2	Attention LSTM	2	4.4M	152.0	143.6
WikiText-2	Attention LSTM	5	11M	148.3	139.5

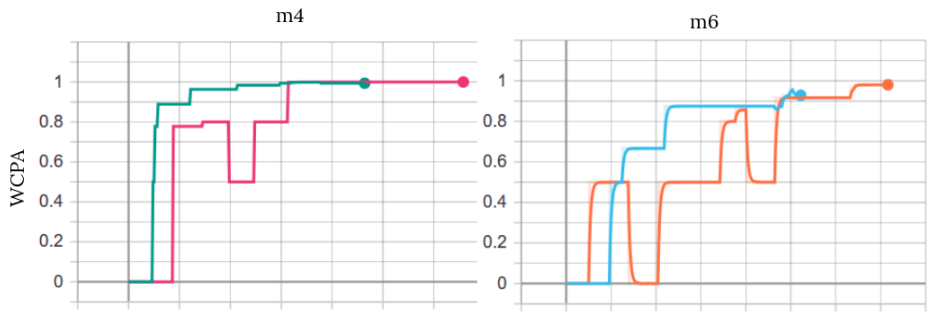


Figure 3: Convergence of WCPA for the baseline LSTM (pink and orange) versus the attention LSTM (green and blue) on the Dyck-2 data. Although the models achieve similar values of perplexity and WCPA, the attention model learns WCPA more quickly and monotonically.

5 Analysis

Differentiation of Second-Order Units

To confirm that the multiple LSTM cells in the attention model are different from each other, we compare the hidden-to-hidden transition matrices. Figure 4 and 5 show the principal angles between sub-spaces (PABS) for pairs of hidden-to-hidden weight matrices. All of the angles between subspaces are greater than zero, meaning that the intersection between subspaces has dimension zero. This indicates that the LSTM cells are differentiating from each other.

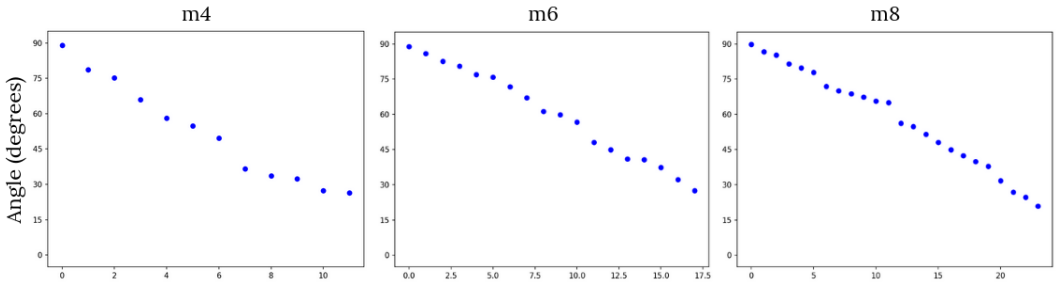


Figure 4: PABS for attention LSTM model hidden-to-hidden transition matrices. The horizontal axis enumerates the principal vectors, and the vertical axis gives the angles in degrees between 0 and 90.

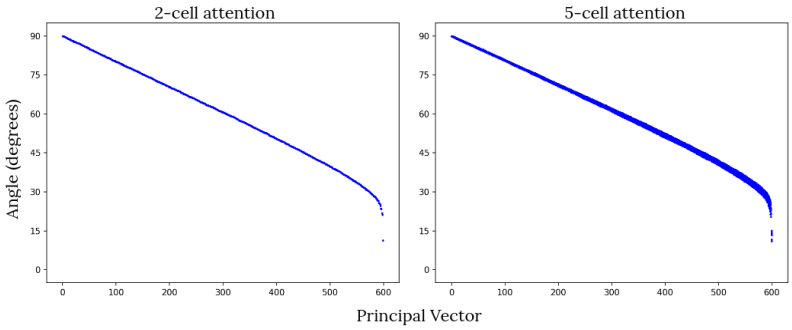


Figure 5: PABS for WikiText-2 attention LSTM models. For the five cell model, all ten pairwise comparisons of hidden-to-hidden matrices are shown. The horizontal axis enumerates the principal vectors, and the vertical axis gives the angles in degrees (between 0 and 90)

In order to understand the nature of the differentiation, we created heatmaps of sample sentences. The color of the word indicates which LSTM unit was assigned the largest attention score, $s^* = \arg\max_s \alpha_{t,s}$, and the intensity of the color reflects the magnitude of α_{t,s^*}. Figure 6 shows sample heatmaps for both the formal and natural languages. The heatmap for the formal language shows that all open parentheses are assigned to one LSTM unit, and all closed parentheses are assigned to the other. This pattern was contrary to our expectations; we expected the two LSTM units to each store the stack for one type of parentheses. However, if the inputs are grammatically correct, a closed parentheses will always indicate a pop from the stack and an open parentheses will always indicate a push to the stack. Therefore, it seems reasonable that one LSTM unit encodes "pop" and the other encodes "push."

The pattern for the natural language heatmap is less discernible. There is no pattern with regard to part of speech or word order within the sentence. The attention is consistent in that individual words are always (softly) assigned to the same cells. Some phrases, like "first down" and "fourth down," are also highlighted consistently. It is possible that a larger number of LSTM units, a larger training set, or fine-tuning of the model parameters would yield a more interpretable pattern. Future work could explore the parameters that yield the best differentiation.

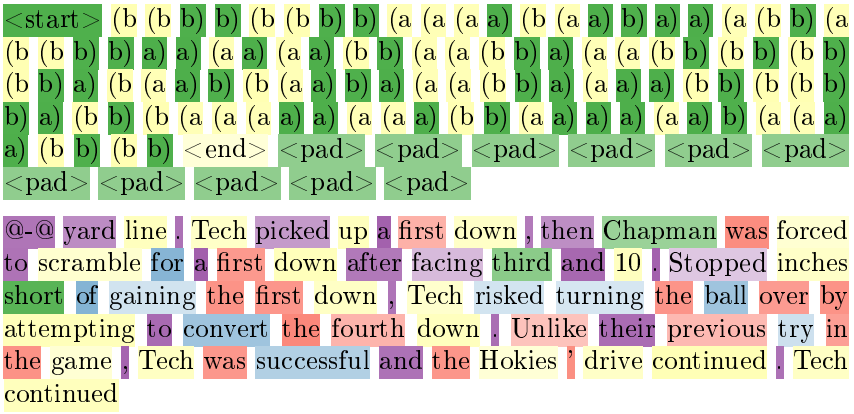


Figure 6: Attention model heatmaps; each color indicates which LSTM unit has the highest attention score, and the intensity of the color represents the magnitude of the attention score. Above, $m4$-bounded Dyck-2 2-cell attention LSTM; below, WikiText-2 5-cell attention LSTM.

Syntactic Evaluation

We adapted [6] to investigate the syntactic strengths and weaknesses of the attention-based model. This syntactic evaluation system provides pairs of nearly-identical sentences, one grammatically correct and one incorrect. The incorrect sentences fall into three broad categories: subject-verb agreement, reflexive anaphoras, and negative polarity items (NPIs). The evaluation metric is the percentage of sentence pairs (in each category) for which a model assigns a higher log likelihood to the sentence that is grammatically correct.

We use this evaluation metric to compare the syntactic awareness of our baseline and second-order attention models. In 11 out of the 19 sub-categories that were tested, the baseline and attention model were quite similar, varying by less than 1%. Four of the remaining categories showed a difference between 2-5%. However, in some sentences that contained an NPI, the baseline and attention models demonstrated specific advantages over one another depending on the nature of the NPI error. Sentences containing an NPI are made grammatical through the presence of a licensor, which gives the sentence negative polarity. For example, a common negative licensor is the word, "no."

To distinguish between sentence types displayed below, grammatical sentences contain the correct negative polarity licensor, meaning the sentence is properly negative. Intrusive sentences do not contain a true polarity licensor, and are not typically positive nor negative. Ungrammatical sentences either contain a positive polarity licensor or do not contain a negative licensor, and thus should not contain an NPI. An oracle model, when provided a grammatical, an intrusive, and an ungrammatical

sentence would always assign the highest log likelihood in that order. The following are example sentences using the NPI "ever":

Simple Grammatical: *No authors will ever be popular.*

Simple Intrusive: *The authors will ever be popular.*

Simple Ungrammatical: *Many authors will ever be popular.*

Cross Clause Grammatical: *No authors that the security guards like will ever be popular.*

Cross Clause Intrusive: *The authors that no security guards like will ever be popular.*

Cross Clause Ungrammatical: *Many authors that the security guards like will ever be popular.*

Table 3: Category-specific comparison of syntactic evaluation results between models

	Baseline	Attention
Cross Clause NPI, grammatical vs. intrusive	0.59	0.42
Cross Clause NPI, intrusive vs. ungrammatical	0.46	0.63
Simple NPI, intrusive vs. ungrammatical	0.88	0.53
Simple NPI, grammatical vs. intrusive	0.06	0.14

Our model outperforms the baseline for Cross Clause intrusive vs. ungrammatical sentences, which can be viewed as supporting evidence for the claim that "architectures with hidden-to-hidden transition functions that are input-dependent are better suited to recover from surprising inputs" [1]. Intrusive Cross Clause NPI sentences are typically constructed by moving the negative licensor to an atypical (or surprising) location in the sentence (e.g. "no" appears after "the authors that" in the example above). Below is the loss score each model assigns to the Cross Clause intrusive NPI sentence. Scores that are lower in magnitude indicate better performance.

	The	*authors*	*that*	*no*	*security*	*guards*	*like*	*will*	*ever*	*be*	*famous*	.
Baseline:	-8.39	-9.12	-5.73	-6.08	-8.48	-7.66	-11.98	-7.05	-7.96	-1.14	-7.49	-0.71
Attention:	-7.95	-8.49	-5.27	-5.75	-8.56	-8.15	-5.79	-8.24	-6.49	-0.8	-9.22	-0.55

The intrusive sentence rearranges the first four words of the grammatical sentence, and as such, constitutes an unusual or surprising input. The baseline model seems confused by this unusual input, and does not recognize "security guards" as the subject of the adjective clause, as evidenced by the high magnitude loss score assigned to the verb "like." The attention model is not as surprised, recognizing that "security guards" can behave as the subject of the adjective clause, even though the true subject of the sentence, "the authors", is less obvious.

Conversely, it appears that when a complicated sentence contains perfect grammar, the baseline performs better than the attention-model, as evidenced by the results for Cross Clause grammatical vs. intrusive sentences. From this, we speculate that for perfect, grammatical English, the baseline model outperforms the attention model. However, for English lacking perfect grammar, the attention model outperforms the baseline. This behavior is likely dependent on the dataset on which each model is trained, WikiText2, which contains language that is likely to display conventionally correct grammar. As a generalization, the baseline outperforms the attention model when provided input that uses similar language to the dataset on which both were trained. When the language of the input differs from the language of the dataset, the attention model outperforms baseline. Performance between the two models on all types of sentences, however, is more similar than dissimilar.

6 Conclusion

Overall, the performance of the attention-based second-order LSTM was comparable to the baseline. The attention-based model training on the m-bounded Dyck-k dataset learned LDPA and WCPA more quickly and stably than the baseline and trained in fewer epochs. On the WikiText-2 dataset, the attention-based second-order LSTM achieved lower perplexity than baseline (139.5 for 5 hidden cells and 143.6 for 2 hidden cells, compared to 147.2 baseline). However, the parameters are not shared

between cells within the attention model, so the attention models contain more parameters than the baseline LSTM.

Furthermore, syntactic evaluation of the attention-based model provided some support to the hypothesis that second-order LSTMs are better able to recover from surprising sentence inputs, although this was only observed for specific types of sentences.

The primary limitation of this work is that it is limited to LSTM architectures. Ideally, we would have compared a second-order RNN model to a baseline RNN as well as the baseline LSTM. This would allow us to assess whether the second-order model is useful for the parenthesis prediction task; this was difficult to assess with only an LSTM because the baseline LSTM works so well in practice.

Future work in this area could explore how training data size, model size, and parameter tuning (particularly the number of LSTM units) impact model performance.

References

[1] Ben Kraus, Iain Murray, Steven Renals, and Liang Lu. Multiplicative lstm for sequence modeling. In *International Conference on Learning Representations (ICLR)*, 2017.

[2] Ilya Sutskever, James Martens, and Geoffrey Hinton. Generating text with recurrent neural networks. In *nternational Conferenceon Machine Learning*, 2011.

[3] Anonymous ACL Submission. Lstms can provably capture bounded hierarchical structure by building a stack. 2020.

[4] Shayne O'Brien. Language modeling. `https://github.com/shayneobrien/language-modeling`, 2018.

[5] Stephen Merity. The wikitext long term dependency language modeling dataset. `https://www.salesforce.com/products/einstein/ai-research/the-wikitext-dependency-language-modeling-dataset/`, 2016.

[6] Rebecca Martin and Tal Linzen. Targeted syntactic evaluation of language models. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 2018.