Evaluating the Ability of LSTMs to Learn Context-Free Grammars

Luzi Sennhauser
Federal Institute of Technology
Zurich, Switzerland
Massachusetts Institute of Technology
Cambridge, MA, USA
luzis@student.ethz.ch

Robert C. Berwick
LIDS, Room 32-D728
Massachusetts Institute of Technology
Cambridge, MA, USA
berwick@csail.mit.edu

Abstract

While long short-term memory (LSTM) neural net architectures are designed to capture sequence information, human language is generally composed of hierarchical structures. This raises the question as to whether LSTMs can learn hierarchical structures. We explore this question with a well-formed bracket prediction task using two types of brackets modeled by an LSTM.

Demonstrating that such a system is learnable by an LSTM is the first step in demonstrating that the entire class of CFLs is also learnable. We observe that the model requires exponential memory in terms of the number of characters and embedded depth, where a sub-linear memory should suffice.

Still, the model does more than memorize the training input. It learns how to distinguish between relevant and irrelevant information. On the other hand, we also observe that the model does not generalize well.

We conclude that LSTMs do not learn the relevant underlying context-free rules, suggesting the good overall performance is attained rather by an efficient way of evaluating nuisance variables. LSTMs are a way to quickly reach good results for many natural language tasks, but to understand and generate natural language one has to investigate other concepts that can make more direct use of natural language's structural nature.

1 Introduction

Composing hierarchical structure for natural language is an extremely powerful tool for human language generation. These structures are of great importance in order to extract semantic interpretation (**?**) and enable us to produce a vast repertoire of sentences via a very small set of rules. Having acquired such a set of rules, it is easy to construct new structures without having previously seen similar examples.

For purposes of external communication, the syntactic structures generated by grammars must be "flattened" or linearized into a sequential output form (e.g. written, signed, or spoken). When reading such a (linearized) text, hearing a spoken sentence or observing a signed language, the structure has to be recovered implicitly to recover the original meaning (i.e., parsing).

In this study, we investigate whether Long Short-Term Memory (LSTM) models (**?**) possess this same ability as humans do: inferring rule-based structure from a linear representation. **?** (**?**) show clearly that there are phenomena in human language that can only be understood by taking the underlying hierarchical structure into account. For neural networks to do the same, it is therefore essential to acquire the underlying structure of sentences.

Recurrent neural networks are often used for tasks like language modeling (**??**), parsing (**???**), machine translation (**?**), and morphological compositions (**?**). LSTMs are inherently sequential models. Since the hierarchical structures appearing in natural language often correlate with sequential statistical features, it can be difficult to evaluate whether an LSTM learns the underlying rules of the sentence's syntax or alternatively simply learns sequential statistical correlations. In this paper we carry out experiments to determine this.

We set up our experiments by posing the LSTM with a bracket completion problem having two possible bracket types, a so-called Dyck Language. A model which recognizes this language has to infer rules of the underlying structure. Furthermore, a system that can solve this task is able to recognize every context-free grammar (see section 3 regarding Dyck Languages via the Chomsky-Schützenberger theorem for why this is

so).

By analyzing the intermediate states of the corresponding LSTM networks, observing generalization behaviours, and evaluating the memory demands of the model we investigate whether LSTMs acquire rules as opposed to statistical regularities.

2 Related work

It has been shown that LSTMs are able to count and partly acquire for context-free languages like $a^n b^n$ and simple context-sensitive languages (**??**). We note that in contrast to the language we investigate here, $a^n b^n$ may be considered the "simplest" context-free language, since it can be generated by a grammar with just one transition.

The question as to whether LSTMs can infer rules on a natural language corpus, e.g., for subject-verb agreement, was initially explored by others such as (**?**). **?** (**?**) investigated the memorization vs. generalization issue for LSTMs for function composition: they showed that if an LSTM learns the mapping from a string-set A to B and from B to C, then the direct mapping from A to C can partly be learned. We use the same method and model for a different task – instead of function composition we evaluate it for bracket matching.

Since most of the time it is challenging to determine what is actually going on with respect to the neural network's internal state, several attempts have been made to visualize a neural network's intermediate states with the goal of making them interpretable (**???**). For several simple copy and palindrome language tasks, it has been shown that RNNs learn a fractal encoding similar to a binary expansion of the input (**???**). With the same objective we use another, recently introduced method to investigate the internal states.

While here we investigate the ability of how well structural information can be stored in originally sequential models, other approaches are currently being taken to move from sequential models to structural ones, e.g. to hardwire structural properties into the model's architecture (**???**); to make a larger external memory available to the network (**??**); or to make the network architecture dynamic (**?**).

Finally, we note that thanks to careful reviewing, we were made aware of **?**'s work (**?**), that addresses essentially the same task as the one we tackle: He investigated also the generalization behaviour of LSTMs for a Dyck-language corpus with several bracket types. He investigated generalization for sentences by concatenating several training sentences; or embedding training sentences in a centrally embedded bracket string. In contrast, we evaluate generalization by training sentences on a certain feature (number of characters, embedded depth) and testing the resulting model on the out-of-sample sentences. By this method, we strive to reduce the probability of similar sub-strings in the training versus the test set.

3 Corpus

When dealing with natural language, there are many side effects or nuisance variables – e.g. words occurring more often in certain correlative contexts or clusters than others. These can influence any classification and experimental result. To minimize such effects, we conducted all experiments on artificial corpora.

The Chomsky-Schützenberger theorem (**??**) about representing context-free language (CFL) states the following: "For each context-free language L, there is a positive integer n, a regular language R, and a homomorphism h such that $L = h(D_n \cup R)$." where D_n is a Dyck language with n different bracket pairs. As described by **?** (**?**), it follows that the Dyck language D_2 essentially covers the entire class of CFLs. Every model which recognizes or generates well-formed Dyck words with two types of brackets should be powerful enough to handle any CFL when intersected with a relabeling (homomorphism of a constructed regular language).

The synthetic corpus we use consists of such a Dyck language with two types of brackets (`[]` and `{}`). Sentences are generated according to the following grammar:

```
S  -> S1 S | S1
S1 -> B | T
B  -> [ S ] | { S }
T  -> [ ] | { }
```

The probabilities of the rules are defined in a way that the entropy – in terms of the number of characters between an opening and its corresponding closing bracket and the depth of embedding at which a bracket appears – is larger than if the rules had all equal probabilities. Formally, the branching probability $P_b = P[\texttt{S1 -> B}]$ and the concatenation probability $P_c = P[\texttt{S -> S1 S}]$ are

defined as follows:

$$
\begin{aligned}
s(l) &= min(1, -3 \cdot \frac{l}{n} + 3) \\
P_b &= r_b \cdot s(l) \quad \text{where} \quad r_b \sim \mathcal{U}(0.4, 0.8) \\
P_c &= r_c \cdot s(l) \quad \text{where} \quad r_c \sim \mathcal{U}(0.4, 0.8)
\end{aligned} \tag{1}
$$

where r_b and r_c are sampled once per sentence and l is the number of already generated characters in the sentence. All 1M generated sentences have a length n of 100 characters.

In this paper, we check whether an LSTM can be trained to recognize this grammar.

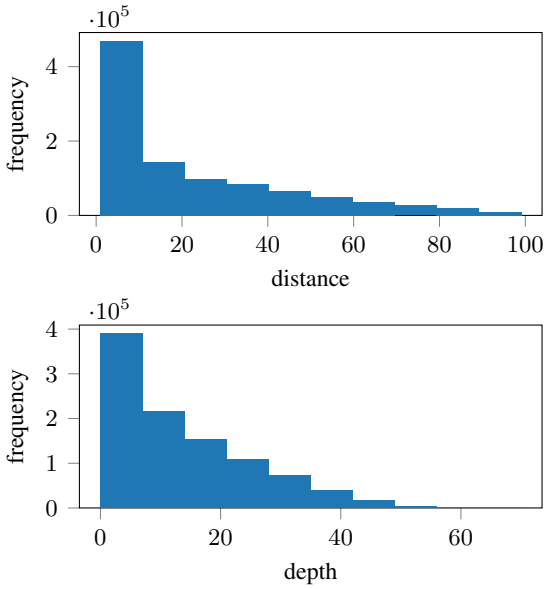

Figure 1: Corpus frequencies

4 Model

To check if we can train a neural network to accept the language generated by the grammar above, an LSTM is used.

4.1 Long Short-Term Memory

Long-Short-Term-Memory networks (LSTM) (**?**) are a variant of recurrent neural networks (RNNs). Both of them possess a memory state that is updated in the process of reading a time series. Many RNNs suffer from the problem of vanishing gradients (**?**): The recurrent activation functions of RNNs are often set to be $tanh$ or the sigmoid function. Since their gradients are most of the times smaller than 1 (for $tanh$ it is upper bounded by 1, and for the sigmoid function even by 0.25), the gradient cannot be conserved during extense back-propagation and approaches 0. LSTMs deal with this issue by containing three multiplicative gates controlling what proportion of the input to pass to the memory cell (input gate), what proportion of the previous memory cell information to discard (forget gate) and what proportion of the memory cell to output (output gate). In the recurrency of the LSTM the activation function is the identity function, which has gradient 1.0. This means that if the forget gate is open, the gradient is fully passed on to previous time steps, and long term dependencies can be learned.

The LSTM reads each input x_i consecutively and updates its memory state c_i accordingly. After each step, an output h_i is generated based on the updated memory state. More specifically, the LSTM solves the following equations in a forward pass:

$$
\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{ix}\mathbf{x}_t + \mathbf{W}_{ih}\mathbf{h}_{t-1} + \mathbf{b}_i) \\
\mathbf{f}_t &= \sigma(\mathbf{W}_{fx}\mathbf{x}_t + \mathbf{W}_{fh}\mathbf{h}_{t-1} + \mathbf{b}_f) \\
\mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} \\
&\quad + \mathbf{i}_t \odot tanh(\mathbf{W}_{cx}\mathbf{x}_t + \mathbf{W}_{ch}\mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{ox}\mathbf{x}_t + \mathbf{W}_{oh}\mathbf{h}_{t-1} + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot tanh(\mathbf{c}_t)
\end{aligned} \tag{2}
$$

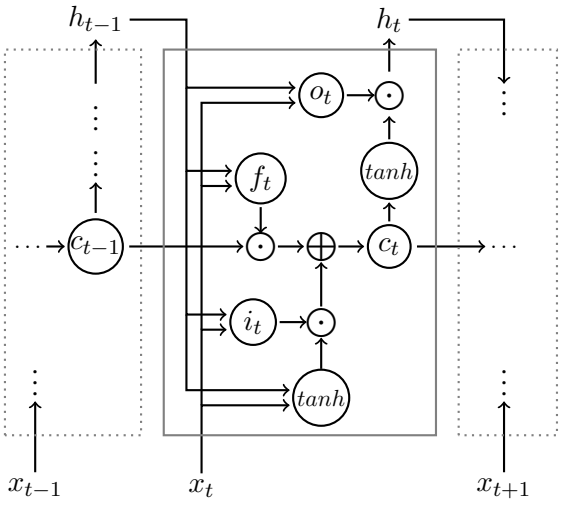

Figure 2: schematic model of an LSTM-cell. $\odot$ stands for element-wise multiplication and $\bigoplus$ for vector addition.

4.2 Basic Model

Now let us turn to the details of the model implementation. We begin with the basic formulation.

Let B_{open} and B_{close} be the sets of opening and closing brackets and $B = B_{open} \cup B_{close}$ the set

of all brackets. Given the beginning of a sentence $w_1, w_2, ..., w_{k-1}, w_k$ with $w_1, ..., w_{k-1} \in B$ and $w_k \in B_{close}$, the LSTMs tries approximate the function:

$$F\colon B^{k-1} \to B_{close}$$
$$w_1, w_2, ..., w_{k-1} \mapsto w_k.$$

The substring (clause) between the corresponding opening bracket of w_i and w_i will be referred to as the *relevant clause* in the remainder of this paper. Likewise, by *distance* we denote the number of characters of the relevant clause. Note that this distance is always a multiple of 2, since the relevant clause is well-balanced. The *depth* at a certain position i is the number of unclosed brackets in the first i characters. The *embedded depth* of a sentence is the maximum *depth* when processing the relevant clause.

To read the input characters, an embedding layer with 5 output dimensions precedes the LSTM. Together they build the encoder, which will read the input sequentially. The decoder, mapping the internal representation to a probability of predicting `}` or `]` is a dense layer with one output variable.

We have compared different initialization methods. It turns out that the initialization of the model is crucial to avoid bad local minima. The following initialization method results in consistently good solutions: To initialize the weights, the model is trained with sentences of length 50 and only afterwards on the actual corpus with sentence length 100.

For backpropagation, the Adam (**?**) optimizer was used. Furthermore, to ensure faster and more consistent convergence, at the beginning of the training, the batch size is gradually increased, which has a similar effect as reducing the learning rate (**?**). In all experiments (and for all models), the corpus is split into 50% training sentences and 50% test sentences. The reported results always refer to the results on the test set.

4.3 Analysis Model

The analysis model is used to analyze what information is stored in the internal representation of the LSTM. In a Push-Down-Automaton model, this internal representation would conceptually correspond to the entire stack.

To analyze the internal representation $[h_i, c_i]$ of the LSTM after having read the input or part of it, we use a method already developed by **?** (**?**) and **?** (**?**): After having trained the basic model, the weights of the encoder are fixed and the labels (previously y) are replaced by some feature z_i of the input $x_1, \ldots, x_i$.

This feature z_i can either be a scalar or a vector. If z_i is a scalar, a dense layer (scalar analysis decoder) is trained to predict z_i. On the other hand, if z_i is a vector (sequence analysis decoder), another LSTM is trained to predict $z_{i,1}, \ldots, z_{i,j}$.

Analyzing the performance of the analysis network shows us how accurately a feature z_i is preserved in $[h_i, c_i]$. One can assume that the LSTM uses its limited memory "efficiently" and therefore discards irrelevant information. Hence, the performance of the analysis decoder shows whether z_i is contained in the information that is relevant for the original classification task.

To begin, two of the experiments which were conducted are presented in the following section to test the trained model performance. For the first experiment z_i is the *depth* (nesting level) after i characters.

Example: For the sequence `{[{}[[]`, z is $(1, 2, 3, 2, 3, 4, 3)$.

For the second experiment we note that theoretically, at any time t, no information about a closed clause in $w_1, \ldots, w_t$ has to be stored, since it is irrelevant for any eventual future prediction of $w_{t+1}, w_{t+2}, \ldots$. When reading from left to right, as soon as a closing bracket is processed, the corresponding clause becomes irrelevant. Therefore, the relevant information is simply the list of bracket types of unclosed clauses. In this experiment we investigate how well the previous characters are preserved in the intermediate representation and evaluate if this correlates with the recovered characters being relevant or not.

Example: after having processed `{[{}][`, only the first and the last characters are relevant, since they are the only ones that could matter for a future classification task. On the other hand, the sub-string `[{}]` is irrelevant. In this example we would evaluate whether the first and last character are better preserved in the intermediate state than the irrelevant sub-string.

To set up the experiment, we set $z_{i,k}$ to be equal to x_{i-k+1}, corresponding to predicting the previous characters of a given intermediate state.

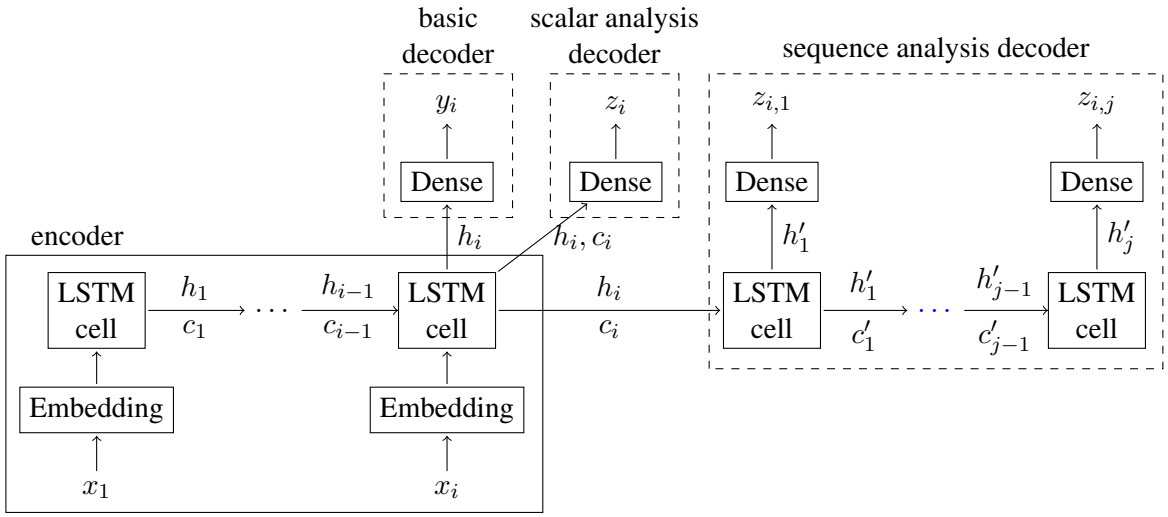

Figure 3: Network architecture of the model. The basic end-to-end model consists of the encoder and the basic decoder. The analysis model fixes the weights for the encoder and uses the scalar (if z_i is a scalar) or sequence analysis decoder (if z_i is a sequence).

4.4 Varying hidden units

The basic model is evaluated with $2, 4, 6, ..., 50$ hidden units. The error rate with 50 hidden units is 0.38% and an error rate of 1% is reached around 20 hidden units. Thus, the error seems to converge with increasing hidden units to a fairly small value. As a result, in all further experiments, the maximum number of hidden units the models are tested against was set to 50.

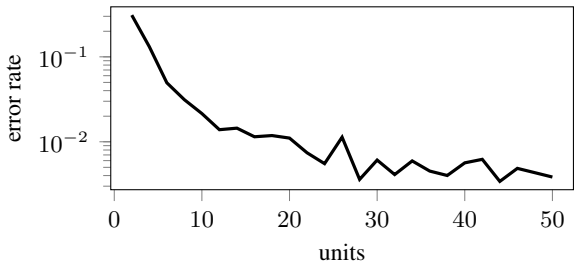

Figure 4: Overall error rate of the basic model with respect to the number of hidden units of the LSTM.

4.5 Memory demand

In this section we evaluate how "difficult" sentences can be with respect to the memory demand of the model, while still reaching an error tolerance of 5%. We have to work with tolerances, because 100% accuracy is not reached. Since it can be challenging measuring how difficult a sentence is to predict, we use the *distance* and the *embedded depth* of a sentence as defined above as metrics.

The resulting values (figure 5) demonstrate that memory demand grows exponentially with respect to the *distance* of sentences that can be predicted. The same behaviour can be observed with respect to the *embedded depth*.

4.6 Generalization

To evaluate the model's generalization performance, training was done only on a systematically chosen subset of sentences (in-sample). To avoid adding additional nuisance variables in this selection, the training sentences are selected according to one of the following rules:

regular interpolation: the sentence has *distance* $2, 6, 10, 14, \ldots$ / odd *embedded depth*.

random interpolation: the *distance* / *embedded depth* of the relevant clause belongs to a set D. D is a random subset consisting of half of all *distances* / *embedded depths* present in the corpus.

extrapolation: the *distance* / *embedded depth* of the relevant clause is smaller than a certain threshold (11 for *distance* and 13 for *embedded depth*).

Running the experiment 100 times – each one with a different random weight initialization – has shown (figure 6) that the results are consistent with respect to the weight initialization. The best out-of-sample accuracy is still worse than almost all in-sample accuracies.

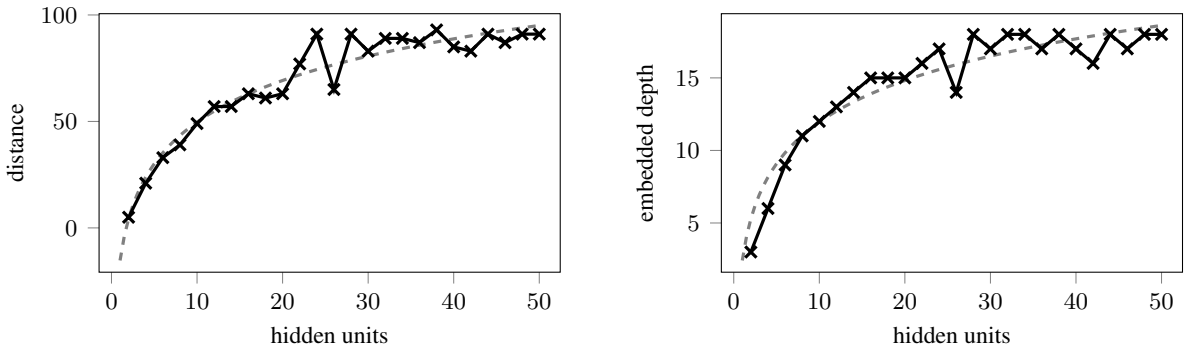

Figure 5: *Distances* (left figure) and *embedded depth* (right figure) that can be predicted with a given number of hidden units and 5% error tolerance. The dashed line is a logarithmic approximation.

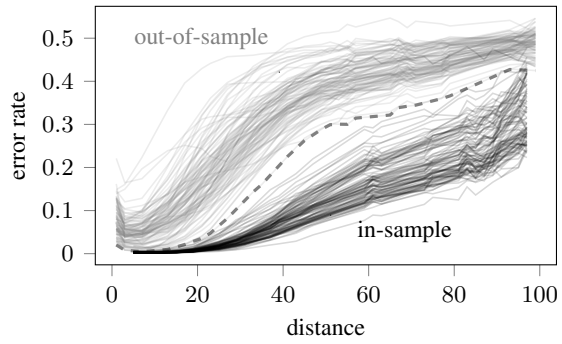

Figure 6: Generalization behaviour of a model with 10 hidden units with respect to the *distance* for 100 runs. Half of the corpus is systematically selected for training (in-sample) – for testing also the left out *distances* are considered (out-of-sample). The bold dashed line is the minimal out-of-sample error out of all 100 runs.

The results (figure 7) demonstrate a large discrepancy between the performance on in-sample (training) and the out-of-sample (testing) accuracy. The experiment was evaluated for different numbers of hidden units. On the one hand, with a large number of hidden units, the generalization error is similarly large (the out-of-sample error rate for interpolation was already between 8.1 and 14.3 times larger than the in-sample error). On the other hand, models with a small number of hidden units did not even converge. The reason for no convergence can be reasonably be explained by the sparse data set, that might lead to more local minima. The maximum generality – especially for smaller *distances* – is observed at around 10 hidden units.

Generalization was evaluated with respect to *distance* and with respect to the *embedded depth*.

For regular interpolation the out-of-sample error for 10 hidden units was on average 5.4 (*distance*) and 5.9 (*embedded depth*) times higher than the in-sample error. Figure 7 shows also that for random interpolation and extrapolation, the model generalizes much worse or not at all.

4.7 Intermediate State Analysis

For the first experiment analyzing intermediate states, recovering the *depth* as defined in section 4.2 from intermediate states shows that the *depth* is only marginally conserved in the intermediate state (figure 8). For a small number of units, the model is only able to distinguish whether a *depth* is either close to 0 or close to the mean *depth* (see figure 8 with two hidden units). When increasing the number of hidden units, the distribution of predictions gets closer to the real distribution of *depths*.

While for two hidden units the prediction of the *depth* is on average off by 7.04, it decreases until it reaches a value of 1.34 for 50 hidden units.

Figure 9 shows how accurately a past character can be recovered from an intermediate state. There is a large discrepancy between the accuracy of relevant and irrelevant characters: If the 4th-to-last character is an irrelevant one, the model is only able to recover the type of bracket with a 33% error rate; whereas if it is a relevant character, it reaches an error below 1.8%. As the number of past characters k approaches 10, the irrelevant information cannot be recovered anymore. Note that an error rate of 0.5 amounts to a random guess, since we evaluate only if it can predict the type of bracket (square or curly) correctly, and not whether it was an opening or a closing one.

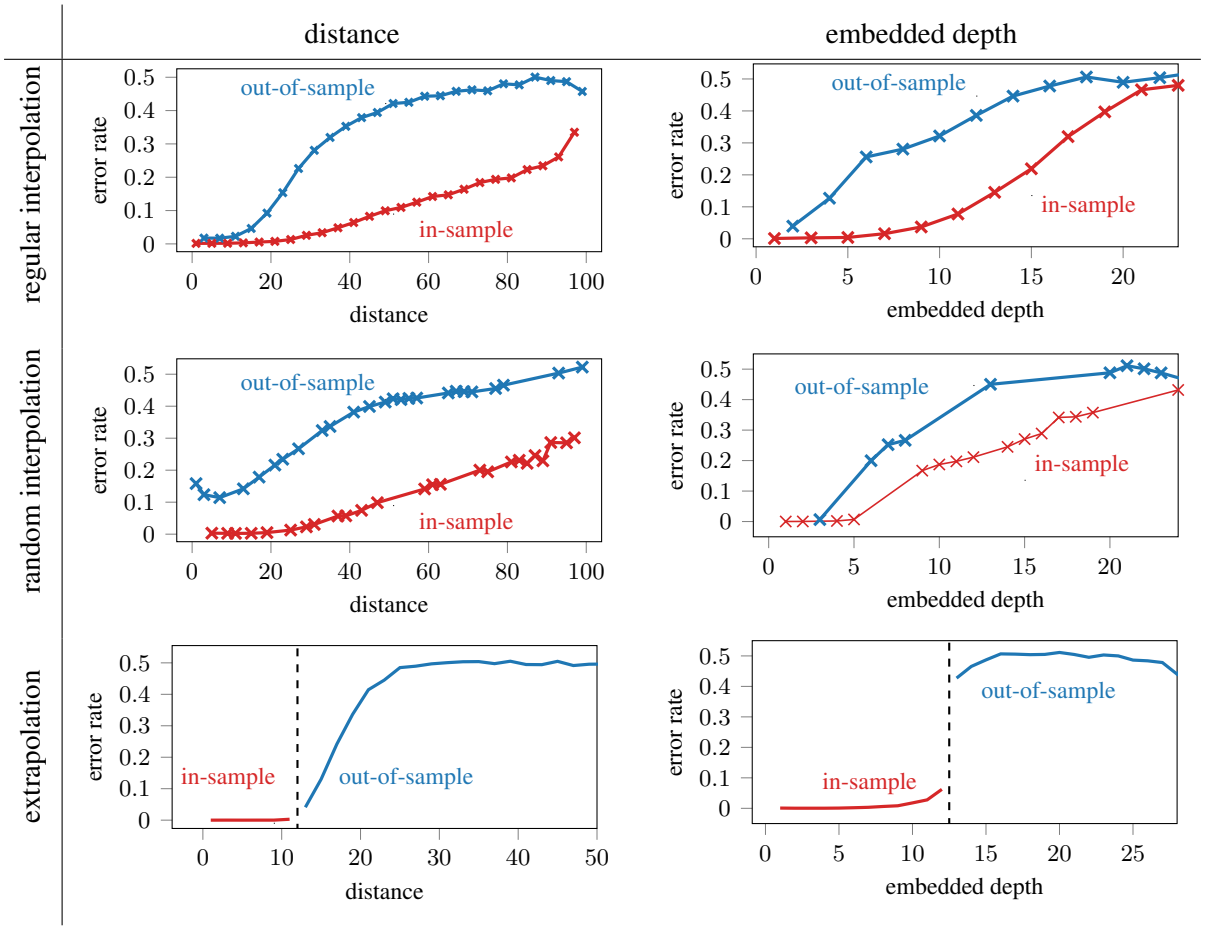

Figure 7: Test for generalization: The error rate of the model with 10 hidden units if only half of the corpus is systematically selected for training (in-sample), while during testing also the left out *distances* / *embedded depths* were considered (out-of-sample).

5 Discussion

We now consider the results of the various experiments, some of which might be considered as controversial on first sight. On the one hand, we see that the LSTM exhibits an exponential memory demand as sentences grow longer, while theoretically, a sub-linear memory ought to be sufficient (**?**). On the other hand, we see that the model has successfully sorted out irrelevant information: the intermediate state analysis shows that irrelevant characters are very quickly forgotten. So, the exponential memory space is not needed for storing irrelevant information for the original classification task.

The strength of structural rules is that they generalize well. In human language this enables humans to create new sentences which have never been heard before. But also for the Dyck language being used, the 4 rules defining the language are enough to generate sentences of arbitrary *distance* and *(embedded) depth*. The only constraint is the memory to store intermediate results while streaming the input. Assuming the model had in fact learned the underlying grammatical rules correctly, an upper bound for the memory required is 50 bits. The model we are using has up to 50 hidden units which corresponds to 11,200 trainable parameters. **?** (**?**) showed that LSTMs can store up to 5 bits of information per parameter and one real number per hidden unit. So we can assume that memory to store the values to process the corpus-defining rules sequentially is not an issue. To partially answer the question of whether LSTM can learn rules we follow a proof by contradiction: if the LSTM learns rules and if these rules are the correct ones, the model would generalize. What we observe is that the LSTM generalizes poorly. Therefore we conclude that the model is not able to learn the right rules.

Combining the generalization results and the intermediate state analysis reveals that the model de-

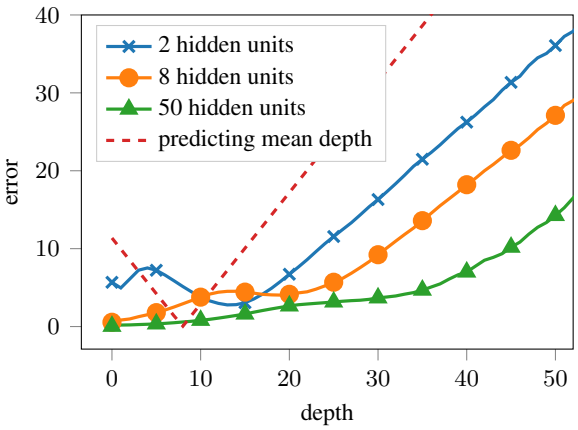

Figure 8: Error rates of predicting the *depth* given an intermediate state of the basic model. It is evaluated for $2, 8, 50$ hidden units. The dashed line indicates the baseline always predicting the mean *depth*. The error indicates how far the prediction is off from the true *depth*.

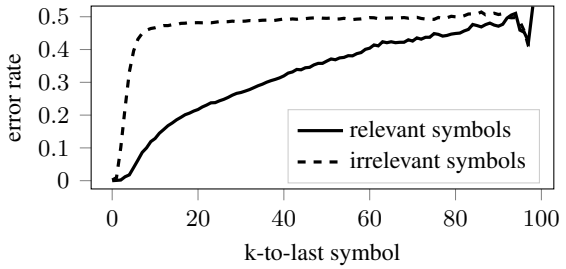

Figure 9: Error rate of predicting the previous characters given an intermediate state of the basic model with 20 hidden units. Characters are grouped as being either relevant or irrelevant for the basic classification task.

termines each character's relevance – but it has learned this without resorting to hierarchical rules. As LSTMs are known to have the ability to capture statistical contingencies, it suggests instead that rather than the "perfect" rule-based solution, what the LSTM has in fact acquired is a sequential statistical approximation to this solution.

The large effect of initialization to a good local minimum suggests that the underlying function may well have many local minima as on reviewers noted. Indeed, **?** (**?**) has already concluded that the memory in LSTMs is mainly used for training effectiveness rather than to increase the storage capacity. Therefore, the large memory demand in our experiments suggests that the LSTM memory is needed to avoid such local minima.

6 Conclusion

At heart, neural networks are statistical models, performing well at capturing and combining correlations of the output variable values and the corresponding component values in the training input. In particular, LSTMs are constructed such that they capture sequential information. Hence, due to the design of their architecture, LSTMs perform very well on statistically-oriented, sequential tasks.

As a result, in experiments like this one that examine whether LSTMs can acquire hierarchical knowledge, one has to pay close attention to nuisance variables like sequential statistical correlations that might be hard to detect and confounded with true hierarchical information.

The bottom line that emerges from this experiment is that the range of rules that an LSTM can learn is very restricted: even a context-free grammar with four simple rules apparently cannot be appropriately learned by an LSTM.

According to most linguistic accounts, natural language syntax relies heavily on hierarchical rules. It enables humans to compose new sentences with relatively little memory capacity and training data. Furthermore, there are sentences that have the same linear representation but differ in structure – syntactically ambiguous sentences. From this perspective, it seems not only more efficient to directly infer structures and rules, but also useful to use rules to understand sentences correctly. The bracket completion task presented here can be understood by a human after only a few training sentences, though online processing of the rules themselves may be difficult. This result invites the conclusion that it will be very challenging for LSTMs to understand natural language as humans do. While LSTMs remain good engineering tools to approximate certain language features based on statistical correlations, the exploration of fundamentally new models and architectures seems a valuable direction to explore on the way to developing methods for understanding human language in the way that people do.

Acknowledgments

We want to thank Beracah Yankama for his help, valuable discussions and the machines to run the experiments on. We thank Prof. Thomas Hofmann for the fast and easy administrative process at ETH Zurich and also for granting access to

high-computing clusters. Additionally we are very grateful for the financial support provided by the Zeno Karl Schindler Foundation.