Learning Hierarchical Structure with LSTMs

Tomas Paris (he/him)

May 12, 2021
Version: Final Draft
Advisor: Alvin Grissom II

Abstract

Recurrent Neural Networks (RNNs) are successful at modeling languages because of their ability to recognize patterns over an undefined input length using their internal memory. However, the data kept in their memory decays over time due to a problem called vanishing gradients. Long Short-Term Memory (LSTM) units mitigate this problem with forget gates which help reserve its memory for only important data. This model has thus become very popular in natural language processing (NLP), because they are able to model context.

Compared to ealier models used in NLP, LSTMs excel at modeling a language modeling. However, some aspects of their success in the field have surprised researchers. Their apparent ability to model syntax suggests that they use mechanisms of learning which we do not yet fully understand. Research has been done on LSTMs and language syntax, in an effort to potentially further the field. Yet, an exhaustive account of how the inside an LSTM works and what needs improving has yet to be compiled. Here, we hope to use previous research and some final experiments to provide a clear picture of how LSTMs model hierarchical syntax.

Contents

1 Introduction **2**

1.1 Background 2

1.1.1 Linguistics 2

1.1.2 Machine learning 3

1.2 Previous Work 6

1.3 Problem Introduction 6

2 Literature Review **7**

2.1 Motivation 7

2.2 Previous Work Overview 7

2.3 Previous Work Detail 8

2.3.1 Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies 8

2.3.2 Evaluating the Ability of LSTMs to Learn Context-Free Grammars 11

2.3.3 LSTMs Compose—and Learn—Bottom-Up 13

2.4 Problem Approach 16

3 Conclusion and Future Work **17**

1

Introduction

Because machine learning is autonomous, sometimes models arrive at correct results before researchers can fully explain how the model succeeded. Computer scientists have run into this issue in natural language processing and the use of LSTM models in this field (Hochreiter and Schmidhuber 1997). LSTMs are currently some of the most successful models used in NLP. Understanding why LSTMs perform the way they do would help researchers develop even better models for the future. In section 1.1, we explain various linguistic and machine learning ideas used throughout this literature review. Section 1.2 introduces the papers that will be reviewed. In section 1.3, we begin to describe our future work.

Section 2.1 describes the motivation behind the choice of papers we review. Section 2.2 summarizes what each paper will cover. In section 2.3, we go through each paper in detail. Section 2.4 covers how we will use the information from each paper in our future work. Chapter 3 gives a conclusion to the ideas described in this literature review.

1.1 Background

1.1.1 Linguistics

Syntax Trees

English syntax can be formalized with a hierarchical tree structure (Chomsky 1957). Each word is a terminal node and has a part of speech (POS). Their parts of speech

then combine to form constituents (phrases and clauses), until they come together at the root, S, which stands for the whole sentence. Syntax trees show how each constituent combines together under the rules of the grammar to build a sentence.

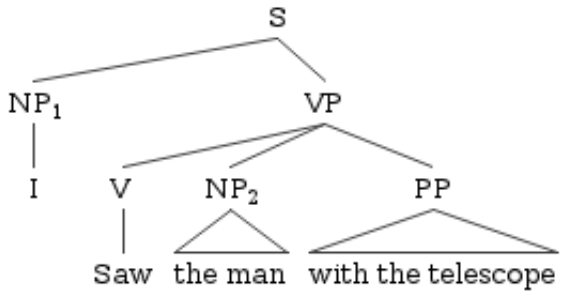

Fig. 1.1: Syntax Tree [1]

Dyck Languages

Sennhauser and Berwick (2018) use Dyck languages when studying the learning process of LSTMs. Dyck languages are formal languages which use brackets as terminals. These brackets have to be balanced, which means that every open bracket '(' has to eventually be closed with a matching ')' bracket. Brackets are not balanced if they are out of order, like in ')(', or if there are unbalanced brackets between them, like in '({)'. The range of Dyck languages is all the languages with balanced brackets and an indeterminate number of bracket types; for example, 2dyck contains only two types of brackets: '(,)' and '{,}', so an example of a valid 2dyck word would be: '({}()){}', and an example of an invalid 2dyck word would be: '({}{)'.

1.1.2 Machine learning

Introduction

Natural languages can be described by a set of recursive grammar rules and lexical items which together can produce infinitely many unique strings (Chomsky 1957). However, the number and complexity of these rules and items makes it difficult to write a program that would recognize natural language appropriately. NLP uses machine learning models to try to solve this issue. In supervised learning, these

[1] hugh, Bishop of (2011). URL: https://commons.wikimedia.org/wiki/File:Syntax_ tree.png.

models try to develop a function $F()$ from data where each example is a set of feature values and a label. The feature values of each example are the variables $x_1, ...x_i$ inputted into the function $F()$, and the label of each example is the output of $F(x_1, ..., x_i)$. The model goes through the examples iteratively to find a function that fits the given data.

Feed-forward Neural Networks

Feed-forward neural networks are a class of machine learning model. They train on a table of features where each row is an individual example and each column is a feature of those examples (with the last column being all the labels for the examples). The model iterates through each row of the table, and multiplies each feature in this row by a weight w_i.

$$x_1 w_1 + x_2 w_2 + ... + x_i w_i \tag{1.1}$$

The result then needs to be scaled to a probability between 0 and 1; the logistic function is often used for this.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.2}$$

This whole process is repeated over several epochs, changing the weights each time, until the model reaches convergence. The product of each epoch is called a hidden layer. The structure is commonly visualized like this:

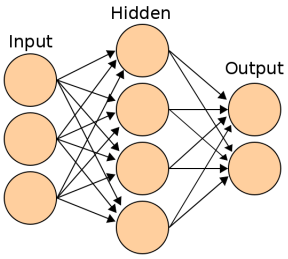

Fig. 1.2: Feed-forward NN [2]

[2] Cburnett (2006). Illustration of the topology of a generic Artificial Neural Network (ANN). URL: https://commons.wikimedia.org/wiki/File:Artificial_neural_network.svg.

Feed-forward neural networks can have any number of hidden layers. The final layer undergoes the same weight multiplication and sigmoid function, but only repeats it for every class of label, so that the number of nodes in the output layer equals the number of possible labels. Then, the predicted output for the function *F()* is the highest output node. During training, the feed-forward neural network takes into consideration whether its prediction is the same as the attached label for each example and uses that information to improve its function design. In testing, the neural network calculates its accuracy and error rate based off of how often its predicted labels are the same as the given labels.

Recurrent Neural Networks

A recurrent unit feeds information from its output layer back into itself as part of the next input. This allows RNNs to use information from previous context during training. The data passed between each example builds up during training. Then, because every output is a small number between 0 and 1, outputs from earlier examples become vanishingly small as they are multiplied together. This can cause old outputs to have a negligible influence on the model's training. This effect, called vanishing gradients, causes RNNs to perform with lower accuracy when their input reaches a high length.

Long Short-Term Memory RNNs

LSTMs are designed to avoid the effect of vanishing gradients. LSTMs do this by using a forget gate which purposefully disposes of data in an output when it is being passed into the next example's input. This way, there is less information carried between loops, which decreases number of small numbers being multiplied together. This change allows LSTMs to accurately handle much larger inputs without running into the problem of vanishing gradients. The below diagram shows the overall structure of an LSTM:

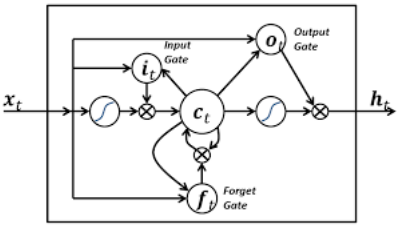

Fig. 1.3: LSTM [3]

1.2 Previous Work

Linzen et al. (2016) describe the success of LSTMs at modeling English syntax. However, when Sennhauser and Berwick (2018) reasoned that LSTMs succeeded in NLP because they modeled syntactic hierarchy with generated grammar rules, their research results showed that their estimation was right for the wrong reasons. While LSTMs do model syntactic hierarchy, they don't generate any grammar rules. In an attempt to better understand the learning process of LSTMs, Saphra and Lopez (2020) developed new tools to analyse their LSTM model as it trained on a formal language. Their results helped to further our understanding of LSTMs and their relation with natural language structure.

1.3 Problem Introduction

While much progress has been made in the research of LSTMs and their abilities, their learning style has not yet been fully documented. More research is needed on the processes that take place inside an LSTM as it learns English syntax. An important part of this research is to make sure that the LSTM is trained with appropriate supervision and a well-filtered data set. The analysis done by Saphra and Lopez (2020) in section 2.3.3 proves to be very useful and should be replicated during this training. The desired result is to find conclusive explanations for the capabilities of LSTMs that can lead to the development of better NLP models for the future.

[3] BiObserver (2015). URL: https://commons.wikimedia.org/wiki/File:Long_Short_Term_Memory.png.

2

Literature Review

2.1 Motivation

Every year thousands of papers are published which detail new ways for machines to learn natural language. There is plenty of research on how effective these RNNs are. The papers selected for this literature review focus on LSTMs. We learn about the capabilities of various LSTM models trained on English sentences or trained on formal languages which emulate aspects of English grammar. We especially focus on papers which not only showcase successful LSTMs, but also try to explain *why* their LSTM models are so successful through various means of analysis.

2.2 Previous Work Overview

Linzen et al. (2016) provides an overview of the capabilities and limits of an LSTM trained directly on English sentences. This paper shows why LSTMs are so popular in NLP right now. Sennhauser and Berwick (2018)'s model is trained on the formal language 2dyck; overall, the paper leaves much about LSTMs unexplained. Saphra and Lopez (2020) is one of the most recent papers analyzing LSTMs, and uses several new analytical methods on their model. This paper has a lot of promise, but ultimately only explores a small portion of the new avenues its discoveries open up.

2.3 Previous Work Detail

2.3.1 Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies

Why LSTMs?

RNNs succeed in natural language processing because they can recognize patterns of arbitrary length within inputs. Patterns of arbitrary length appear commonly in natural language, whose recursive structure allows any number of words to separate syntactically associated words. For example:

The cat eats. → The cat with the hat eats. → The cat with the hat on the mat eats.
→ The cat with the hat on the mat ... using the bat eats. → etc.

As the model needs to recognize these long-distance syntactic word dependencies, it also needs to ignore any intervening words that are not important to its prediction. The forget gates in LSTMs suggest that they are the most promising RNN architecture for this job.

Linzen et al. (2016) choose the number prediction task as an initial test for their LSTM model. The number prediction test just asks a model to predict the number of a verb (either singular or plural) given every word in the sentence leading up to that verb. The most obvious, and most commonly correct, solution to this problem is to make the verb the same number as the noun immediately preceding it. However, as suggested above, there can be any number of words between a subject and its verb. Among these intervening words, there can be nouns that distract from the subject. When these nouns are the opposite number of the subject, they are called agreement attractors. Linzen et al. (2016) make the case that a model which can successfully find a subject among agreement attractors has shown a fundamental ability to model English syntax structure.

Testing Set-up

The number prediction task is the most structured and supervised test which Linzen et al. (2016) use on their model. They proceed to give their model a grammaticality judgment training objective; this test gives their model full sentences and an annotation reporting whether the sentence's verb correctly agrees with its subject or not. The grammar annotation does not specify where the specific ungrammaticality is in the sentence; it just gives a binary judgement of whether the sentence is grammatical or not. The final and most difficult task given to the LSTM model is the language modeling objective. This test gives the LSTM one word of a sentence at a time and asks the model to predict the next word at each step.

The data used for these tests are English sentences taken from Wikipedia. The corpus totals at about 1.35 million sentences; 9% of them are used for training, 1% for validation, and all the remaining data are used for testing. The encoding of each input stores words as one-hot vectors, which means that the characters that spell each word are ignored completely by the model. Linzen et al. (2016) use a baseline for the number prediction task which only keeps the nouns of its inputs and removes every other word. This baseline is meant to show how much the model relies on the syntactic information of function words (words like 'the' or 'an') and other parts of speech to parse its input. The baseline has two categories: the first only keeps common nouns, while the second keeps common nouns and pronouns and proper nouns. These baselines are both referred to as noun-only baselines.

The Number Prediction Task Results

Of all the tests given, the model is most successful in the number prediction task, boasting a 0.83% error rate. While the noun-only baselines also succeed at this task, they have a higher error rate, with the common-nouns case having 4.2% errors and the all-nouns case having 4.5% errors. Linzen et al. (2016) run several modified versions of this test on their model in order to measure its ability in different areas. In one version, they replace words in their data with parts of speech. They only replace words that have a single POS in more than 90% of their occurrences, to avoid any ambiguities. The model's results for this test are similar to its unfiltered baseline results, suggesting that these baseline results correctly classify each word's POS.

Other Results

The grammaticality judgement task has 2.5% errors. Although this is a higher error rate than the number prediction task, it still suggests that LSTMs are largely successful at modeling language even when working under a very indirect learning structure. The language modeling objective gives the LSTM the most trouble, causing a worse-than-chance prediction rate when the data is confined to sentences with agreement attractors. This result suggests that the language modeling objective is not structured enough to teach an LSTM how to model language.

Over-sampling Difficult Sentences

Linzen et al. (2016) note that, although they had spent a lot of time researching the effect of agreement attractors on their LSTM models, sentences that actually had agreement attractors took up a vanishingly small portion of their data. In response, they suggested that their model might have performed better if they had cut the number of simpler sentences down so that the difficult agreement attractor sentences took up a larger percentage of their data. They tried repeating the number prediction task with a new data set where the proportion of sentences with agreement attractors was doubled during training. During testing, the resulting model showed a 27% decrease in its error rate on agreement attractor sentences.

Conclusion

The successes that their models display exemplify why there is so much attention on the use of LSTMs in the field of NLP right now. In the course of building so many varying LSTM models on English sentences, Linzen et al. (2016) also provide many suggestions regarding how to best train an LSTM to process language. Perhaps the most important of these is that unfiltered English data causes the model to overfit the simpler sentences, while using a data set that over-samples difficult examples notably improves the model's performance.

2.3.2 Evaluating the Ability of LSTMs to Learn Context-Free Grammars

Motivation

In natural language production, people use a finite set of recursive rules to produce an infinite number of sentences (Chomsky 1957). These sentences, however, are linearized when we use them in speech. Sennhauser and Berwick (2018) aim to determine whether LSTMs learn the structural rules behind a language or simply learn to recognize sequential patterns in the language's linear form. They test LSTMs on a Dyck language to find out.

Testing Set-up

The Chomsky-Schutzenberger theorem (Chomsky and Schutzenberger 1963) states that for each context-free language L, there is a positive integer n, a regular language R, and a homomorphism h such that $L = h(D_n \cup R)$. D_n here is a Dyck language with n types of brackets. Thus the LSTM model is tested on a Dyck language of two types of brackets, since any model which can recognize D_2 will also be able to recognize virtually any context-free language. The language uses brackets '[', ']' and '{', '}'. Its grammar is as follows:

S $\rightarrow$ S_1 S | S_1
S_1 $\rightarrow$ B | T
B $\rightarrow$ [S] | { S }
T $\rightarrow$ [ ] | { }

Some examples of words in this language are: '{[{}[[]' (invalid), and '{[]{[]}}' (valid). The generated corpus has one million sentences which are 100 characters long. 50% of this corpus is used for training, and the other 50% is used for testing.

The basic LSTM model is given various numbers of hidden layers. At 20 hidden layers, the error rate has dropped to 1%; at 50, the error rate is 0.38%. Sennhauser and Berwick (2018) use 50 hidden layers as the maximum number in their models.

Memory Demand

The **depth** at a certain position in a sentence is the number of unclosed brackets that have been read up to that point. One test given to the LSTM is to report the depth of the input at each step. The embedded depth of a sentence is the highest depth that is reached in that sentence. LSTM models of various sizes are tested on sentences of increasing length and embedded depth to determine how these variables increase memory demand. If a corpus causes a model's error rate to increase above 5%, then testing moves to a model with a higher number of hidden units (thus the model demands more memory to retain its accuracy). The results suggest that LSTMs require exponential increases in size as the inputted sentences grow more difficult.

Internal Representations

One possible explanation of this exponential increase in memory demand is that the LSTM is not properly forgetting unimportant data. Thus, a test is given to the LSTM model that is designed to check whether the LSTM recognizes that closed brackets are no long relevant to the output. Sennhauser and Berwick (2018) designed an analytical tool which checks how accurately a bracket can be recovered from a hidden layer. This tool is used to check whether closed brackets are being forgotten by the LSTM while unclosed brackets are being kept track of. The model's recovery of closed brackets has a 33% error rate, while recovery of unclosed brackets has less than 1.8% errors, showing that the LSTM's forget gate handles the 2dyck language efficiently.

Filtered Tests

Sennhauser and Berwick (2018) filter their training examples by their length and embedded depth in various different ways (for example, one training data set only had sentences of even length and odd embedded depth) in order to reduce overlap with the testing data. An LSTM model relying on predicting sequences based on their statistical likelihood will perform worse on this out-of-sample testing data, while a model which generalizes structural rules will not be affected by the change. The results showed that even an LSTM's best results for out-of-sample testing is

still worse than the worst results for in-sample testing, suggesting that the LSTM's predictions relied on the statistical likelihood of sequences.

Conclusion

The observation that LSTMs do reach exponential memory growth at a certain input length only highlights how important this research is so that we can make better models in the future. The affirmation that an LSTM's forget gate operates correctly on a formal language is an important confirmation to make as we strive to understand how an LSTM truly works.

2.3.3 LSTMs Compose—and Learn—Bottom-Up

Motivation

Saphra and Lopez (2020) aim to show that LSTMs learn **bottom-up**. This style of learning would mean that the model's internal representation of English syntax parallels actual syntax tree structures. The way they try to provide evidence for this is by showing that an LSTM model learns longer patterns faster if it has already learned the shorter sequences that show up inside the pattern. The specific example used to test their model is the words 'either' and 'or'. If LSTMs learn bottom-up, then they should recognize the relation between 'either' and 'or' faster if the phrases used between them have already been learned.

Analytical Tools

Saphra and Lopez (2020) deploy various analytical methods in order to closely follow the internal processes of LSTMs. The first of these analytical tools is called contextual decomposition (CD), which determines how much a hidden vector in an LSTM is affected by a preceding set of words. In addition to CD, Saphra and Lopez (2020) use decompositional interdependence (DI), which measures the associations between sets of words at a timestep.

CD and DI are then used to show how LSTMs learn syntactic structure. Specifically, they show that the LSTM model learns in a bottom-up manner, using shorter sequences to build up understandings of larger patterns. This type of learning is hierarchical in nature and explains why LSTMs can so successfully capture natural language syntax.

Contextual Decomposition

CD is a method of showing the impact that specific inputted words or phrases have on a model's output. First, the relevant part of the input must be selected for the CD to focus on. Then, the CD divides each hidden layer of the LSTM into two parts: a set of relevant hidden vectors, which were strongly impacted by the selected words, and a set of irrelevant hidden vectors, which the selected words did not affect. The relevant vectors are then converted using the softmax function:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.1}$$

This function is used to normalize the effect the relevant words had on each vector, so that they can be compared at a similar scale.

Decompositional Interdependence

DI is useful for phrases whose individual words cannot be analyzed separately. Phrases like "slice of cake" need to be kept whole for the proper meaning to be communicated. DI measures how much a word or words rely on other words for their impact. So, when analyzing a phrase like "slice of cake", each individual word would have a high DI, because the meaning of 'slice' depends on 'cake'. Besides syntactic phrases, idioms are also highlighted by DI. DI is given two sets of words which have already been run through CD, and the timestamp which CD outputted with them. It then computes the strength of the nonlinear interactions between the two sets of words.

Testing Set-up

A short span of words needed to predict a long relation, such as the span of words between 'either' and 'or', is called a **scaffold**. Saphra and Lopez (2020) use a synthetic corpus to test whether familiar scaffolds, which are scaffolds the LSTM has already learned to recognize, cause faster learning of a long-distance relation than unfamiliar scaffolds. The synthetic corpus they use is in a language defined by the long-distance rule called the $\alpha - \omega$ relation. This relation is defined: $\alpha\Sigma^{\kappa}\omega$, where α is the start symbol which needs to be followed by ω after a span of unique characters Σ^{κ} that is κ characters long. The results of testing with this synthetic corpus show that an LSTM model learns the $\alpha - \omega$ relation fastest if the scaffold Σ^{κ} is familiar and reused from the training data. However, this may be due to vanishing gradients instead of hierarchical learning.

Vanishing gradients occur when older inputs affect the resulting output exponentially less than newer inputs. LSTMs greatly reduce this problem with their forget gates, but it still arises in cases of especially long input. The faster learning of familiar scaffolds may occur because a familiar scaffold takes up less memory in an LSTM, and therefore avoids vanishing gradients much better than unfamiliar scaffolds.

Results

Saphra and Lopez (2020) attempt to use DI to isolate the effect that the open symbol α has on the output. DI shows that the open symbol has a strong relation to familiar scaffolds; in fact, the DI grows higher and higher as the scaffold is read and continues to be familiar. This relation persists even though most cases of the familiar scaffold in the training data appear outside of the $\alpha - \omega$ relation, in unrelated context. This persistence of relation supports the idea that the LSTM learns bottom-up, or compositionally.

Conclusion

In this paper, Saphra and Lopez (2020) offer a concrete explanation for the positive performance of LSTMs. In the process, several important analytical techniques are developed which will help further our understanding of LSTMs even more.

2.4 Problem Approach

Saphra and Lopez (2020)'s CD and DI helped reveal a tangible connection between LSTM models and natural language syntax. However, this connection was found on an LSTM trained on a formal language. Even more information might be found if we use CD and DI on an LSTM trained on English sentences. Thus, we will train the LSTM on the Number Prediction Task from Linzen et al. (2016). We will also compare a baseline data set with a filtered data set which over-samples sentences that the LSTM finds more difficult. Linzen et al. (2016) found evidence that this change could prevent an LSTM model from over-fitting simple sentence structures; our tests will explore this possibility.

It will also help to use Sennhauser and Berwick (2018)'s analytical tools, which they used to track the effect of their LSTM's forget gate. We will thus be able to confirm that our LSTM is forgetting agreement attractors while remembering subjects. We will use the CD and DI on our model to check whether the DI of a sentence's subject grows as the model reads in the agreement attractors between it and the verb. This will provide a direct example of an LSTM modeling an English sentence in the same way a syntax tree does, from the bottom up.

3

Conclusion and Future Work

LSTMs have proven themselves to be an important tool for NLP in recent years. Linzen et al. (2016)'s different training tasks show that LSTMs can model some aspects of English syntax very well when given enough supervision. Sennhauser and Berwick (2018) provide evidence that the forget gate in LSTMs helps them to handle syntactic patterns spanning arbitrary distances. Their model was trained on the formal language of 2dyck, however, and we hope to provide further evidence for their conclusions by testing the performance of the forget gate on an LSTM learning English. Saphra and Lopez (2020) draw a clear parallel between the way LSTMs recognize sequences and the way humans parse English syntax, giving further evidence for why LSTMs can model language better than previous models.

LSTM's have several limitations, given the exponential memory demand that Sennhauser and Berwick (2018) discover and their failure to learn under less supervised conditions in Linzen et al. (2016)'s experiments. Yet, their use in NLP doesn't require unlimited input lengths or completely unsupervised testing. And as we learn more about LSTMs, we gain insights that could improve the use of many machine learning architectures in NLP. Linzen et al. (2016) discover that over-sampling difficult samples in a data set of English sentences is necessary to prevent the model from overfitting the simplest and most common cases. Saphra and Lopez (2020) and Sennhauser and Berwick (2018) both found ways to evaluate learning processes happening at inference time in their model. This internal evaluation helps the debugging and documentation of a model, and is an important process for any machine learning model.

The hierarchical structure of natural languages elevates LSTM RNNs above previous, non-recurrent machine learning models. Yet, the field of NLP is always improving, and other recurrent structures, such as the Transformers (Vaswani et al. 2017), are already being implemented in NLP following the success of LSTMs.

References

Chomsky, Noam (1957). “Logical Structures in Language”. In: *American Documentation* 8.Oct (cit. on pp. 2, 3, 11).

Chomsky, Noam and Marcel P Schutzenberger (1963). “The algebraic theory of context-free languages”. In: *Studies in Logic and the Foundations of Mathematics* 35, pp. 118–161 (cit. on p. 11).

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long short-term memory”. In: *Neural computation* 9.8, pp. 1735–1780 (cit. on p. 2).

Linzen, Tal, Emmanuel Dupoux, and Yoav Goldberg (2016). “Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies”. In: Transactions of ACL 2016. Vol. 4. Dec, pp. 521–535 (cit. on pp. 6–10, 16, 17).

Saphra, Naomi and Adam Lopez (2020). “LSTMs Compose - and Learn - Bottom-up”. In: Findings of EMNLP 2020. Oct (cit. on pp. 6, 7, 13, 15–17).

Sennhauser, Luzi and Robert C. Berwick (2018). “Evaluating the Ability of LSTMs to Learn Context-Free Grammars”. In: Proceedings of EMNLP 2018. Nov (cit. on pp. 3, 6, 7, 11, 12, 16, 17).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). “Attention is all you need”. In: Proceedings of NeurIPS. Vol. 30, pp. 5998–6008 (cit. on p. 17).