Logo 1

Logo2

University of Potsdam

Faculty of Human Sciences

Bachelor Thesis

in Computational Linguistics

Submitted as Partial Fulfillment of the Requirements for the Degree of Bachelor of Science

Topic:	Investigating the ability of RNN architectures to learn context-free grammars by example of Dyck(2)
Author:	Fynn Dobler <fynndobler@gmail.com> Matr.-Nr. 775710
Version of:	January 13, 2020
1. Supervisor:	Prof. Dr. Thomas Hanneforth
2. Supervisor:	Dr. Uladzimir Sidarenka

Summary

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Abstract

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Contents

Index of Figures 4

Index of Tables 4

List of Abbreviations 5

1 Introduction 6

2 Theoretical Background 6

2.1 Formal Languages and Formal Grammars 6

2.2 Formal Grammars and Natural Language 7

2.2.1 Natural Language as supra-regular 7

2.2.2 Natural Language as supra-context-free 8

2.3 Dyck Languages 9

2.4 Neural Network Architectures 10

2.4.1 Simple RNN 10

2.4.2 LSTM 11

2.4.3 GRU 12

2.5 Related Works 13

3 Experiment Setup 15

3.1 Models 15

3.2 Corpus Construction 16

3.3 Evaluation 18

3.3.1 Experiment 1: Long-Range Dependency 19

3.3.2 Experiment 2: Deeper Nesting 19

3.3.3 Experiment 3: Generating Depth 20

4 Results 20

5 Discussion 20

6 Conclusion 20

Bibliography 21

Appendix 23

Eidesstattliche Erklärung 24

List of Figures

1 Chomsky Hierarchy 6

2 Unfolded RNN 11

3 LSTM Memory Cell 12

4 Illustration of a GRU 13

List of Tables

1 Formal grammar properties. 7
2 Corpus sizes in current works 14
3 Overview of investigated models 14
4 Reported values for performance in previous works 15
5 Training corpora properties 16

1 Introduction

2 Theoretical Background

2.1 Formal Languages and Formal Grammars

A formal language L is defined as a subset of Σ^* over an alphabet Σ, with Σ^* being the set of all words over the alphabet. For the purpose of formal language theory, the subset of Σ^* that constitutes the language $L(G)$ needs to be well-formed given the formal grammar G. As per Jurafsky und Martin (2009), the definition of a formal grammar is $G = \{N, \Sigma, R, S\}$, where N is a set of non-terminal symbols, Σ is a set of terminal symbols (alphabet), R is a set of rules of the form $\alpha \rightarrow \beta$ (where α and β are strings of symbols from $(\Sigma \cup N)^*$) and S is a designated start symbol.

$L(G)$ consists of all strings w that can be derived from the start symbol S in a finite number of steps, formally $\{w \in \Sigma^* | S \overset{*}{\underset{\text{G}}{\Rightarrow}} w\}$. As such, a word $w \in \Sigma^*$ that cannot be derived from S in a finite number of steps is not part of $L(G)$.

Formal grammars differ in terms of complexity and can be described in a hierarchical manner. Grammars of higher complexity have a greater generative power than grammars of lower complexity. The most commonly used hierarchy of grammars is the Chomsky hierarchy (Chomsky (1959)). In this hierarchy, formal grammars are classified into three types, sorted from most powerful to least powerful: Turing equivalent (Type 0), Context Sensitive (Type 1), Context Free (Type 2) and Regular (Type 3).

The difference in generative power and complexity stems from increasing restrictions imposed on the rules of the grammar - a Type 3 grammar is more restrictive than a Type 0 grammar. As such, every grammar of a higher type is a subset of the previous type of grammar. A visual representation of this property can be found in Figure 1.

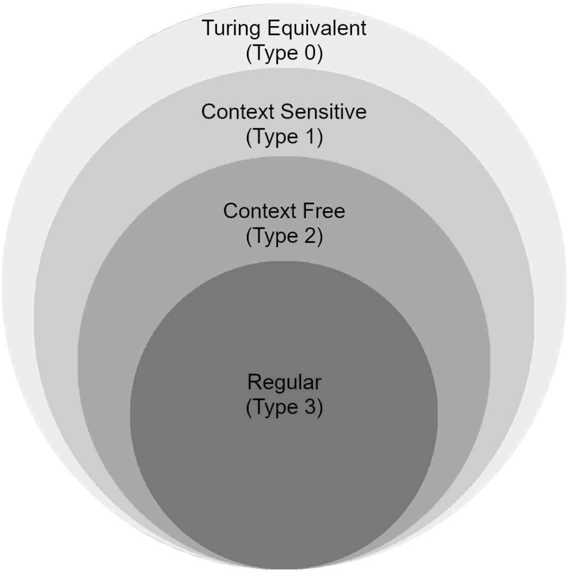

Figure 1: A visual representation of the Chomsky Hierarchy.

The four types of formal grammars can be defined by the form their rules can take. An overview over these rules as per Jurafsky und Martin (2009) can be found in Table 1, where A is a single non-terminal, α, β, γ are strings of terminal and non-terminal symbols, and x is a string of terminal symbols. α, β and γ may be empty unless specifically disallowed. The table is supplemented with a column describing the corresponding automaton capable of accepting or recognizing the grammar.

2.2 Formal Grammars and Natural Language

The correspondence of formal grammars to automata and Computational Complexity Theory lends itself to consider natural languages under the same lense. While formal grammars constitute powerful tools with which phenomena in natural language can be described, assessing precisely where natural languages lie within the Chomsky hierarchy is the subject of ongoing research. The usual form arguments answering that question take is one of finding lower bounds: If there is a phenomenon in a natural language that cannot be described with a given grammar, natural language must be - however slighty - more complex than that type allows. Such arguments increase in credibility the more frequently they can be replicated for phenomena in multiple languages. The arguments establishing natural languages as supra-context-free (i.e. more complex than CFGs) as well as contrary evidence from empiric research shall be presented here.

2.2.1 Natural Language as supra-regular

English, as well as several other languages (Hagège (1976)) allow for center embedding, the embedding of a phrase into another phrase of the same type.

(1) The man eats.

(2) The man the boss fired eats.

(3) The man the boss the investor distrusted fired eats.

(4) The man the boss the investor the police investigated distrusted fired eats.

Type	*Name*	*Rule Skeleton*	*Automaton*
0	Turing Equivalent	$\alpha \rightarrow \beta$, s.t. $\alpha \neq \epsilon$	Turing Machine (recognized)
1	Context Sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$, s.t. $\gamma \neq \epsilon$	Linear Bound Automata (accepted)
2	Context Free	$A \rightarrow \gamma$	Push Down Automata (accepted)
3	Regular	$A \rightarrow xB$ or $A \rightarrow x$	Finite-State Automata (accepted)

Table 1: Overview of formal grammar properties according to Jurafsky und Martin (2009), augmented with corresponding automata.

Let the set E contain all grammatical sentences of English, and let the noun phrases and transitive verbs constitute following sets:

$$A = \{\text{the boss, the investors, the police,} \dots\}$$
$$B = \{\text{fired, distrusted, investigated,} \dots\}$$

Then the following two sets can be defined.

$$E' = \{\text{the man } a^n b^n \text{ eats} \mid n \geq 0\}$$
$$R = \{\text{the man } a^* b^* \text{ eats}\}$$

a^n and b^n are finite sequences of size n of elements of sets A and B, respectively. E' describes a subset of E, namely $E \cap R$. Since regular languages are closed under intersection and E' is not regular, E is not regular.[1]

While this proof is correct under the framework of Formal Language Theory, the validity of claiming that it shows natural language to be supra-regular is debatable. Research in psycholinguistics shows that native speakers have face severe problems processing center embeddings of depth two or higher, yielding long processing times, an incomplete understanding of the presented sentence or leading the participants to judge the sentence as ungrammatical (Hamilton und Deese (1971), Frank et al. (2016)). Furthermore, the corpus-driven analysis by Karlsson (2007) suggests an upper limit of center embedding depth three in the seven investigated languages.

The proof concerns linguistic competence, whereas empirical studies provide an insight into linguistic performance. Usually, the distinction between competence and performance provide an adequate framework for both theoretical and empirical work, but when discussing the complexity of natural language neither side should be discarded. A linguistic competence framework cannot appropriately declare natural language as supra-regular if natural language is produced and processed according to regular rules. Conversely, for the purposes of computational linguistics, exclusively assessing linguistic performance by way of corpus analysis is insufficient, as a corpus of any size pales in comparison to the potential for infinitely many combinations that characterizes language. For the purposes of NLP, it is recommendable to err on the side of overestimating the complexity of natural language - so long as the resulting models are sufficiently efficient and accurate.

2.2.2 Natural Language as supra-context-free

Similarly to the proof given in Section 2.2.1, an argument characterizing natural language as supra-context-free can be brought forth. It is based on embedded infinitival

[1] The proofs for regular languages being closed under intersection and E' not being regular can be found in the appendix.

verb phrases found in Swiss German (Shieber (1987)).

(5) Jan säit das mer em Hans es huus haend wele hälfe
Jan said that we the Hans-DAT the house-ACC have wanted help
aastriiche.
paint
'Jan said that we have wanted to help Hans paint the house.'

(6) Jan säit das mer d'chind em Hans es huus haend
Jan said that we the children-ACC the Hans-DAT the house-ACC
wele laa hälfe aastriiche.
have wanted let help paint
'Jan said that we have wanted to let the children help Hans paint the house.'

Four finite sets can be constructed from these examples: accusative noun phrases ($A = \{\text{d'chind}, \dots\}$), dative noun phrases ($B = \{\text{em Hans}, \dots\}$), transitive verbs taking accusative objects ($C = \{\text{laa}, \dots\}$) and transitive verbs taking dative objects ($D = \{\text{hälfe}, \dots\}$). Let the set S then be the set of all grammatical sentences of Swiss German. Again, the two following sets can be defined:

$$S' = \{\text{Jan säit das mer } a^n b^m \text{ es huus haend wele } c^n d^m \text{ aastriiche} \mid n, m \geq 0\}$$

$$R = \{\text{Jan säit das mer } a^* b^* \text{ es huus haend wele } c^* d^* \text{ aastriiche}\}$$

S' is not context-free and results from $S \cap R$. Since context-free sets are closed under intersection with regular sets, G cannot be not context-free.[2]

Curiously enough, empirical research into the matter of processing similar cross-serial dependencies in Dutch suggests them to be generally easier to process than nested dependencies (i.e. the ones used to prove natural language to be supra-regular) (Bach et al. (1986)).

2.3 Dyck Languages

Whether natural language is regular, context-free, supra-regular or supra-context-free is a distinction of only tangential relevance for this work. The first two cases are fully covered by CFGs, while the other two leave room for some natural language productions outside of the scope of CFGs. The characteristics of supra-context-free examples in natural language show a *weak* non-context-freeness, making CFGs sufficient for covering the vast majority of natural language productions. With this assumption, an appropriate CFG for a model to learn must be found. The most important property of this grammar is that model performance on its language must allow for strong conclusions about the learnability of any other CFG. In doing so, one can make reasoned assumptions about potential model performance on natural language data.

[2] The respective proofs for S' not being context-free and context-free sets being closed under intersection with regular sets can be found in the Appendix.

One such a grammar is the Dyck Grammar, which can produce an array of Dyck Languages. Let $D_n = \{N, \Sigma, R, S\}$ with

$$\begin{aligned}
&N = \{S\} \\
&\Sigma = \{\epsilon, O_1, O_2, \dots, O_n, C_1, C_2, \dots, C_n\} \\
&R = \{ \\
&\quad S \to \epsilon \\
&\quad S \to O_n\, S\, C_n\, \},
\end{aligned}$$

where O_n represents an opening parenthesis, C_n represents a closing parenthesis and n denotes the number of distinct pairs of parentheses. D_1, then, denotes the Dyck Language with $\Sigma = \{\epsilon, (,)\}$, D_2 the Dyck Language with $\Sigma = \{\epsilon, (, [,],)\}$, et cetera.

Within the family of Dyck Languages, D_2 is of particular interest. According to the Chomsky-Schützenberger Representation Theorem (Chomsky und Schützenberger, 1963), for every context-free language L there exists a positive integer n, a regular language R, and a homomorphism h so that $L = h(D_n \cap R)$. Following the proof in Autebert et al. (1997), a homomorphism g_n can be constructed so that $D_n = g_n^{-1}(D_2)$. It follows that every context-free language can be represented as $L = h(g_n^{-1}(D_2) \cap R)$. As such, every CFL could be represented via homomorphisms on D_2 and intersections with a regular language[3]. Assuming natural languages to be context-free and bearing in mind that using a formal language is a choice of abstraction which allows for precise control over corpus composition, this makes D_2 the language of choice when evaluating neural network performance on a comparative scale.

2.4 Neural Network Architectures

2.4.1 Simple RNN

Recurrent Neural Networks (RNNs) (Elman (1990)) are a neural network architecture particularly suited to processing sequential information by virtue of them being recurrent: the RNN's output at a time step t is fed back as its input at the following time step $t + 1$. As such, every output is dependent on the previous computation as well as the current input. This property equips RNNs with a "memory" with regards to previous inputs, allowing them to capture context dependencies a context agnostic model cannot adequately learn.

Within the frame of this work, the specific case of the Simple RNN (SRNN) is considered. It is a three layer networks, consisting of an input layer, a hidden layer and an output layer. The hidden state h_t at time step t given the input vector x_t and the output vector y_t are calculated as per the following equations:

[3] NDFL vs DFL!

$$h_t = f(\boldsymbol{W_{xh}} x_t + \boldsymbol{W_{hh}} h_{t-1}) \tag{1}$$

$$y_t = \boldsymbol{W_{hy}} h_t \tag{2}$$

The function f constitutes a non-linear transformation, like tanh or ReLU. $\boldsymbol{W_{xh}}$, $\boldsymbol{W_{hh}}$, $\boldsymbol{W_{yh}}$ are matrices of the weights connecting the input layer to the hidden layer, the hidden layer to itself and the hidden layer to the output layer, respectively.

When training RNNs, it is beneficial to think of the network as unfolding into an architecture with one layer per time step. A visualisation is provided in Figure 2. As such, the RNN can be treated as a deep feedforward network. These conceptual layers share their parameters - if any weight changes at time step t, the weight also changes at $t+1, t+2, \ldots, t+n$. Isolated changes are not possible. A popular training algorithm for RNNs is Backpropagation Through Time (BPTT) (Williams und Zipser (1998)), a gradient based algorithm designed for recurrent rather than feedforward networks. However, as Bengio et al. (1994) and Hochreiter (1998) show, RNNs suffer from a fundamental flaw: the aptly named vanishing gradient problem, in which the training gradient diminishes to zero throughout the layers.

Figure 2: An RNN, unfolded through time.

2.4.2 LSTM

Long-Short Term Memory networks (LSTM) were designed by Hochreiter und Schmidhuber (1997) as an RNN architecture which preserves the RNN capabilities of processing sequential data of arbitrary length and capturing context dependencies, while circumventing the vanishing gradient problem.

LSTMs are based on self-connected linear units which are regulated by three multiplicative gates: input (in), output (out) and forget (φ). At every time step, the concatenated vector of the previous hidden state h_{t-1} and the current input x_t are received by all three gates. Given this input, the gates apply individual multiplicative factors $m_{\text{Gate}} \in [0, \ldots, 1]$, determining what information is let through the input gate, passed through the output gate or forgotten by the self-connected linear unit.

$$
\begin{aligned}
\mathrm{in}_t &= m_{\mathrm{in}}(\boldsymbol{W}_{\mathrm{in}} \cdot [h_{t-1}, x_t] + b_{\mathrm{in}}) \\
\mathrm{out}_t &= m_{\mathrm{out}}(\boldsymbol{W}_{\mathrm{out}} \cdot [h_{t-1}, x_t] + b_{\mathrm{out}}) \\
\varphi_t &= m_{\varphi}(\boldsymbol{W}_{\boldsymbol{\varphi}} \cdot [h_{t-1}, x_t] + b_{\varphi})
\end{aligned}
$$

Finally, the cell state C_{t-1} is updated to C_t and h_t is set.

$$
\begin{aligned}
C_t &= \mathrm{forget}_t \odot C_{t-1} + \mathrm{in}_t \odot \tanh(\boldsymbol{W}_{\boldsymbol{C}} \cdot [h_{t-1}, x_t] + b_{\mathrm{C}}) \\
h_t &= \mathrm{out}_t \odot \tanh(C_t)
\end{aligned}
$$

Figure 3: An LSTM memory cell, bla, bla bla.

2.4.3 GRU

A less complex alternative to LSTMs, the Gated Recurrent Unit (GRU) was developed by Cho et al. (2014). The information flow within the GRU is handled by just two gates: reset (r) and update (z). The update gate determines how much information from previous time steps is passed along for further time steps, while the reset gate enables the model to drop irrelevant information and only consider the current input rather than the previous hidden state, as described in the equations below, where j is the j-th hidden unit, σ is the squashing sigmoid function, $\boldsymbol{W}$ and $\boldsymbol{U}$ are learned gate-dependent weight matrices and ϕ is a non-linear function.

$$
\begin{aligned}
r_j &= \sigma\Big([\boldsymbol{W}_{\boldsymbol{r}} x]_j + [\boldsymbol{U}_{\boldsymbol{r}} h_{t-1}]_j\Big) \\
z_j &= \sigma\Big([\boldsymbol{W}_{\boldsymbol{z}} x]_j + [\boldsymbol{U}_{\boldsymbol{z}} h_{t-1}]_j\Big) \\
h_j^t &= z_j h_j^{t-1} + (1 - z_j)\tilde{h}_j^t \\
\tilde{h}_j^t &= \phi\Big([\boldsymbol{W} x]_j + [\boldsymbol{U}(r \odot h_{t-1}]_j)\Big)
\end{aligned}
$$

Figure 4: Illustration of a GRU.

2.5 Related Works

Formal languages have been used to evaluate the performance of neural network architectures for decades (Cleeremans et al. (1989), Zeng et al. (1994), Hochreiter und Schmidhuber (1997), Rodriguez und Wiles (1998), Gers und Schmidhuber (2001), Joulin und Mikolov (2015)). These papers frequently determine how well the architecture can generalize from training on a relatively small subset of the language to testing on longer unseen words. (Example results)

While these results make for compelling cases in favour of the architectures proposed by the authors, the methodology has recently come under criticism by researchers like Bernardy (2018), Sennhauser und Berwick (2018). The former goes as far as to say that none of the previous research compellingly shows RNNs to be capable of learning nested hierarchical structures.

Formal languages making up early performance benchmarks for neural networks was, in the early years of RNN research, at least partially caused by the lack of comparable natural language corpora and of reliable hardware with sufficient computational power. By now, of course, these restrictions are far above the levels of 1990 (Elman RNN). This lead to a sizeable corpus of research on NLP, i.e. Karpathy et al. (2015).

There are several recent works investigating neural architectures and their capability of learning Dyck languages specifically. [They feature a variety of ways of sampling their corpora, evaluate models on different tasks and report a varying subset of relevant performance measures.]

Deleu und Dureau (2016) evaluate the capability of a Neural Turing Machine (NTM) to deal with long-term dependencies. The model performs a membership classification task on words belonging to D_1 and non-words using the same alphabet, but not belonging to D_1. The NTM shows strong generalization by correctly recognizing well over 90% of D_1 words with length 180, despite only being trained on words of length < 12. The authors do not report on any other measures of performance.

Li et al. (2018) deserve a mention in this section for evaluating their nonlinear weighted finite automata on a corpus generated by a probabilistic D_1 grammar and reporting on word error rate (WER) relative to the size of the training set. The size of the training set ranges from 200 to $20,000$, with a fixed test set size of 250.

Skachkova et al. (2018) compare the accuracy and perplexity of Elman-RNNs, GRUs and LSTMs learning D_1, D_2, D_3, D_4 and D_5. The respective corpora are generated by

Paper	D_n	*Grammar Probability*	*Training Corpus Size*
Deleu und Dureau (2016)	1	equal	unclear
Li et al. (2018)	1	modified	200 - $20,000$
Skachkova et al. (2018)	1-5	modified	$131,072$
Sennhauser und Berwick (2018)	2	modified	$1,000,000$
Suzgun et al. (2019)	1-2	modified	$10,000$
Yu et al. (2019)	2	modified	$1,000,000$

Table 2: Overview of corpus sizes in current works.

Paper	*Architectures*
Deleu und Dureau (2016)	Neural Turing Machine, LSTM
Skachkova et al. (2018)	Elman-RNN, GRU, LSTM
Sennhauser und Berwick (2018)	LSTM
Suzgun et al. (2019)	Elman-RNN, GRU, LSTM
Yu et al. (2019)	seq2seq

Table 3: Overview of investigated models.

a probabilistic Dyck grammar, with the given rule probabilities controlling the average length of the generated sequences.

Sennhauser und Berwick (2018) choose an approach that demonstrates whether an LSTM can learn hierarchical structures by training the models for a prediction task on one million D_2 words with length 100. Their training corpus has been sampled from a probabilistic D_2 grammar. They report on two further corpus properties: distance, which is the number of characters between an opening bracket and its corresponding closing bracket, and embedded depth, which is the maximum number of unclosed brackets in a processed word. They report error rate, predictable distances and embedded depth in relation to number of hidden units. Furthermore, they perform an Intermediate State Analysis. Their results suggest the trained models fail at correctly learning the rules of the D_2 grammar.

Suzgun et al. (2019) offer an overview of small-sized Elman-RNNs, GRUs and LSTMs learning D_1, D_2 and corresponding shuffle languages. They report accuracy and provide an analysis of the LSTM cell state dynamics. They find that none of their models are capable of learning D_2, with the highest scoring model (LSTM) achieving a 48.24% accuracy.

The topic of investigation of Yu et al. (2019) lies outside of the scope of this thesis, as it is predominantly concerned with the seq2seq framework and attention mechanisms. However, their data set consists of largely the same settings as Sennhauser und Berwick (2018) have used. Additionally, they report many of the same measures of performance.

An overview of investigated Dyck languages, corpus sizes and reported measures of performance can be found in Tables 2, 3 and 4.

Paper	*Accuracy*	*Perplexity*	*Cell State*	*AUC*	*Error Rate*
Deleu und Dureau (2016)	No	No	No	Yes	No
Skachkova et al. (2018)	Yes	Yes	No	No	No
Sennhauser und Berwick (2018)	No	No	Yes	No	Yes
Suzgun et al. (2019)	Yes	No	Yes	No	No
Yu et al. (2019)	No	Yes	No	No	Yes

Table 4: Overview of reported values for performance. Cell State Analysis does not refer to a unified method, it merely means the paper investigates cell states at all. AUC refers to the area under the curve for an increasing length of Dyck words the model was able to generalize.

In conclusion, there is currently no benchmark Dyck corpus within the literature upon which to evaluate model performance. Furthermore, there is also no consensus on which measures of performance to report. Considering how wildly the corpora and the reported measures of performance differ, the results of the publications mentioned here cannot be directly compared with each other. This allows for no conclusive statement on the relative performance of various popular RNN architectures based on the current literature.

3 Experiment Setup

3.1 Models

For the following experiments, the three RNN architectures described in Section 2.4 have been used. All models consist of a single layer of size $n = \{2^1, 2^2, \ldots, 2^n\}$, followed by a dense layer of size 5 with softmax activation. Each neuron in the dense layer corresponds to one character in the training data - `{`, `[`, `]`, `}` and $ as an end-of-word marker. As such, the activations of the dense layer serve as interpretable output.

All models were trained with the same parameters. The training data was received one character at a time, in batches of 512. The loss was computed by categorical cross-entropy. Furthermore, the Adam optimizer (Kingma und Ba (2014)) was applied with a learning rate of 0.0001. The models were trained for 50 epochs each. From those 50 epochs, the model with the lowest loss was chosen to perform on the test data.[4]

The models were trained on the same training data for both experiments. To answer the question of training data influence on model performance, three distinct sets of training data were used, yielding a total of $9 \times 3 \times 3 = 81$ evaluated models.

[4] The models were implemented in Tensorflow 2.0. The source code can be found at https://github.com/FyDob/BSc-Thesis.

Corpus	*Word Length*	*maxND*	*maxBD*
Baseline	18.38	4.31	13.03
Low LRD	BLA	BLA	BLA
Low LRD	BLA	BLA	BLA

Table 5: Properties of the three corpora the models were trained on.

3.2 Corpus Construction

To investigate the influence of corpus composition on model performance, three corpora were created: a baseline corpus which is directly sampled from a subset of D_2, as well as two modifications of the baseline corpus: one impoverishing the training data from long-range dependencies (Low LRD) and one enriching the training data with more long-range dependencies (High LRD). The sampling and modification processes will be explained later in this section.

The experiments were explicitly designed to test the models' abilities to generalize based on the training data they encounter. As such, it is prudent to give consideration to which properties the training data might possess to facilitate or inhibit generalizability - properties such as length, nesting depth (ND) and the distance between a pair of opening and closing brackets (BD). These measures are in terms of averages and variance in Table 5.

Furthermore, the training corpora were chosen to be a small slice of a comparatively large subset of D_2. To facilitate generalization, the training corpora consist of words of varying length. As discussed in Section 2.5, previous works largely utilized similarly small language subsets and achieved encouraging results. For a discussion of Experiment 1 and 2 on a training corpus consisting of a majority of the target language, see Bernardy (2018).

In determining an eligible maximum length, a fact about the size of D_n subsets ways used: a Dyck language D_n contains $n^m C_n$ words of length $2m$, where C_m is the m-th Catalan number (Skachkova et al. (2018)). It follows that a maximum length limit of $2m$ produces a set of size $\sum_{i=2}^{2m} n^i C_i$. For example, a maximum length of 20 in D_2 ($D_2^{\leq 20}$) yields $20,119,506$ words, which is a sufficiently large subset to sample from. The words were generated following the probabilistic grammar set forth by Sennhauser und Berwick (2018).

$$
\begin{aligned}
&S \rightarrow Z\ S \mid Z \\
&Z \rightarrow B \mid T \\
&B \rightarrow [\ S\ ] \mid \{\ S\ \} \\
&T \rightarrow [\ ] \mid \{\ \}
\end{aligned}
$$

The production $Z \rightarrow B$ branches, whereas $S \rightarrow Z\ S$ concatenates two smaller Dyck words. This representation provides a good intuition for understanding the merit of Experiment 1. The probabilities with which the rules were applied are calculated as follows, with alternative rules of course being applied with the complementary probability:

$$\begin{aligned} P_{\text{branch}} &= r_{\text{branch}} \cdot s(l) \quad \text{with } r_{\text{branch}} \sim U(0.7, 1.0) \\ P_{\text{concat}} &= r_{\text{concat}} \cdot s(l) \quad \text{with } r_{\text{concat}} \sim U(0.7, 1.0) \\ s(l) &= \min(1, -3 \cdot \frac{l}{n} + 3) \end{aligned}$$

with l being the number of already generated non-terminal characters and n the maximally desired length of the word. r_{branch}, r_{concat} and l were sampled at every step of word generation.

Following this process, $140,000$ words in $D_2^{\leq 20}$ were generated. These words served as the basis for creating the three corpora. To created the Low LRD corpus, all words with a maximum bracket distance higher than 14 were modified[5] by first identifying the bracket pair with the highest bracket distance, then simply moving the opening bracket from its original position to the position right before the closing bracket. (i.e. `{[{}]}` becomes `[{}]{}`). This has the largest impact on bracket distance throughout the corpus, while ensuring grammaticality of the resulting word. The resulting set of long-range impoverished words was merged with all unmodified words, deleting all duplicates.

The High LRD corpus was created in a similar way: First, all words with a bracket distance lower than 18 were identified.[6] Then, the first pair of neighbouring closing brackets is found and deleted. The remaining word is wrapped in a randomly chosen pair of brackets, creating the longest possible bracket distance between the two (i.e. `{[{}]}` becomes `{{[]}}`). The resulting set was merged with the unmodified words the same way as the Low LRD set.

The deletion of duplicates naturally leads to not all $140,000$ initially created words being used for training. Indeed, the chosen cut-off points resulted in a much smaller Low LRD set of $87,311$ words. For the sake of keeping training in full batches of 512, the closest multiple of the batch size is $512 \times 170 = 87,040$, leading to a training corpus size of 512×170 words. Furthermore, since the investigated manipulations were the changes in ND and BD and not the effects of corpus size, the Baseline and High LRD corpus were truncated to size 512×170 by randomly removing surplus words.

[5] This cut-off point was chosen as it significantly reduces the average maximum bracket distance without creating too many duplicates.

[6] The same considerations as for the Low LRD corpus cut-off apply.

3.3 Evaluation

The designs for the first two experiments closely follow the procedure described by Bernardy (2018) - and the evaluation procedure was inspired by the discussion in his work as well. As he notes, it is indeed impossible to achieve a full 100% accuracy on prediction tasks for Dyck languages. This is owed to the fact that at any position within a word, the valid options for the next character encompasses all opening brackets in addition to whichever closing bracket might be appropriate (i.e. the sequence `[[{}]` can be followed by either `]`, `{` or `[`). Indeed, this makes deciding how to score accuracy for the following experiments rather difficult. Obviously, an assessment of accuracy only ever makes sense when the next character in the test word is a closing bracket - when there is no prescribed limit to the length of a Dyck word, opening brackets are never a wrong prediction. But how to score accuracy on the position of the closing bracket? Either, one only assesses the correct closing bracket as the one correct prediction, or one allows all predictions except the wrong closing bracket and the end-of-word marker to be seen as correct.

In a series of experiments designed to assess how well - if at all - neural network models can learn the underlying grammar of a training corpus, choosing the first option seems counter-intuitive. After all, prediction an opening bracket instead of the expected closing bracket could be an indication for appropriate rule application. However, accepting all open bracket predictions to be correct does not contribute to an accurate assessment of grammaticality either: Once the model has predicted an opening bracket in place of a closing bracket, it is assuming a structurally very different word than is actually present in the test set. This predicted word would have a deeper nesting depth and a longer-running range dependency than the test word. A much more insightful measure, then, is to ask whether the model actually closes its predicted opening bracket.

Since there is no elegant way to unite both approaches, the idea of assessing a model's capability of closing a self-predicted bracket will be pursued in Experiment 3. For Experiments 1 and 2, accuracy will be calculated more strictly. If, and only if the next character in the test word is a closing bracket will the model's prediction be evaluated. If the model predicted the correct closing bracket, it has made a correct prediction. Every other prediction is rated as false. As such, the model accuracy by character index is calculated as follows:

$$\mathrm{acc}_i = \frac{\#\text{ of correct predictions up to index } i}{\#\text{ of closing brackets up to index } i}$$

A random guessing strategy choosing from all brackets in the alphabet yields a baseline 25% accuracy. A perfect predictor, then, could score 100% if it both perfectly resolves all opening and closing bracket dependencies and never predicts an open bracket instead of a closed one. This indeed would make a 100% accuracy both undesirable and puzzling - and an indication that the model has not actually learned a generalizable grammar resembling D_2. In reality, a prefect predictor that has learned D_2 should score around 33.33% accuracy. The remaining 66.66% are split equally on the two opening brackets.

3.3.1 Experiment 1: Long-Range Dependency

For this experiment, the test set consisted of 512×17 Dyck words with length $1+18+18+1 = 38$. They were created by picking two random Dyck words $w_1, w_2 \in D_2^{=18}$ from the previously generated $140,000$ words $\in D_2^{\leq 20}$, concatenating them and wrapping the result in a randomly selected pair of matching brackets as follows:

$$w_{\text{LRD}} = O_n w_1 w_2 C_n$$

While w_1 and w_2 might have been seen in training, the resulting word most certainly has not been observed. Neither could the model possibly have encountered a long-range dependency spanning 36 characters between the opening and closing bracket. As such, a high accuracy on the final character of the word serves as a strong indication for the model having not just learned the necessity of closing open brackets to form an acceptable Dyck word, but also for the model having retained information on the first opening bracket during all processing steps. For further insight in model behaviour, the accuracy is reported for every processed character.

3.3.2 Experiment 2: Deeper Nesting

To investigate how well a model performs on predicting brackets on a nesting level deeper than anything included in training, another test set was constructed. Generalizing to extreme long-range dependencies as in Experiment 1 is not necessary for this task, leading to the words being significantly shorter.

For this task, 512×17 words $w \in D_2^{=20}$ have been chosen at random. They were then wrapped by a prefix of five randomly chosen opening brackets and a suffix of the corresponding closing brackets as follows:

$$w_{\text{DN}} = O_n O_n O_n O_n O_n w C_n C_n C_n C_n C_n$$

This process still has the model extrapolate beyond the length of the training words, while not confounding the results with LRD performance too much. Indeed, when analysing the results, the main focus lies on the infixed word, since all its nesting depths are now increased by 5, possibly affecting model performance.

3.3.3 Experiment 3: Generating Depth

4 Results

5 Discussion

6 Conclusion

Bibliography

Autebert, J.-M., Berstel, J., und Boasson, L. (1997). *Context-Free Languages and Pushdown Automata*, pages 111–174. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bach, E., Brown, C., und Marslen-Wilson, W. (1986). Crossed and nested dependencies in german and dutch: A psycholinguistic study. *Language and Cognitive Processes*, 1:249–262.

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

Bernardy, J.-P. (2018). Can recurrent neural networks learn nested recursion? volume 16.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., und Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Chomsky, N. (1959). On certain formal properties of grammars. *Information and Control*, 2(2):137 – 167.

Chomsky, N. und Schützenberger, M. (1963). The algebraic theory of context-free languages*. In Braffort, P. und Hirschberg, D., editors, *Computer Programming and Formal Systems*, volume 35 of *Studies in Logic and the Foundations of Mathematics*, pages 118 – 161. Elsevier.

Cleeremans, A., Servan-Schreiber, D., und Mcclelland, J. (1989). Finite state automata and simple recurrent networks. *Neural Computation - NECO*, 1:372–381.

Deleu, T. und Dureau, J. (2016). Learning operations on a stack with neural turing machines. *CoRR*, abs/1612.00827.

Elman, J. L. (1990). Finding structure in time. *Cognitive Science*, 14(2):179–211.

Frank, S. L., Trompenaars, T., und Vasishth, S. (2016). Cross-linguistic differences in processing double-embedded relative clauses: Working-memory constraints or language statistics? *Cognitive Science*, 40(3):554–578.

Gers, F. A. und Schmidhuber, E. (2001). Lstm recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

Hagège, C. (1976). Relative clause, center-embedding, and comprehensibility. *Linguistic Inquiry*, 7(1):198–201.

Hamilton, H. W. und Deese, J. (1971). Comprehensibility and subject-verb relations in complex sentences. *Journal of Verbal Learning and Verbal Behavior*, 10(2):163 – 170.

Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116.

Hochreiter, S. und Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

Joulin, A. und Mikolov, T. (2015). Inferring algorithmic patterns with stack-augmented recurrent nets. *CoRR*, abs/1503.01007.

Jurafsky, D. und Martin, J. H. (2009). *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Karlsson, F. (2007). Constraints on multiple center-embedding of clauses. *Journal of Linguistics*, 43(2):365–392.

Karpathy, A., Johnson, J., und Li, F. (2015). Visualizing and understanding recurrent networks. *CoRR*, abs/1506.02078.

Kingma, D. und Ba, J. (2014). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Li, T., Rabusseau, G., und Precup, D. (2018). Nonlinear weighted finite automata. In Storkey, A. und Perez-Cruz, F., editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 679–688, Playa Blanca, Lanzarote, Canary Islands. PMLR.

Rodriguez, P. und Wiles, J. (1998). Recurrent neural networks can learn to implement symbol-sensitive counting. In *Proceedings of the 1997 Conference on Advances in Neural Information Processing Systems 10*, NIPS '97, pages 87–93, Cambridge, MA, USA. MIT Press.

Sennhauser, L. und Berwick, R. C. (2018). Evaluating the ability of lstms to learn context-free grammars. *CoRR*, abs/1811.02611.

Shieber, S. M. (1987). *Evidence Against the Context-Freeness of Natural Language*, pages 320–334. Springer Netherlands, Dordrecht.

Skachkova, N., Trost, T., und Klakow, D. (2018). Closing brackets with recurrent neural networks. pages 232–239.

Suzgun, M., Gehrmann, S., Belinkov, Y., und Shieber, S. M. (2019). LSTM networks can perform dynamic counting. *CoRR*, abs/1906.03648.

Williams, R. und Zipser, D. (1998). Gradient-based learning algorithms for recurrent networks and their computational complexity.

Yu, X., Vu, N. T., und Kuhn, J. (2019). Learning the Dyck language with attention-based Seq2Seq models. In *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 138–146, Florence, Italy. Association for Computational Linguistics.

Zeng, Z., Goodman, R. M., und Smyth, P. (1994). Discrete recurrent neural networks for grammatical inference. *IEEE transactions on neural networks*, 5 2:320–30.

Appendix

Eidesstattliche Erklärung

Eidesstattliche Erklärung zur <-Arbeit>

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Unterschrift : *Ort, Datum :*