University of Potsdam
Faculty of Human Sciences

Bachelor Thesis

in Computational Linguistics

Submitted in 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:	March 10, 2020
1. Supervisor:	Dr. Thomas Hanneforth
2. Supervisor:	Dr. Uladzimir Sidarenka

Summary

Abstract

Zusammenfassung

Contents

List of Figures

List of Tables

1 Introduction

In the 2010s, seemingly every major technology company developed its own "personal assistant" system, a program that allows the end-user to interact with the company's services more intuitively by interpreting spoken natural language commands. Apple's Siri, Amazon's Alexa and Google's succinctly named Assistant have been irrevocably ingrained in day-to-day life. While the ethical and data security concerns raised by this development are still a point of contention, it is clear that Natural Language Processing (NLP) applications have boomed from a niche field to a rapidly growing multi-million dollar industry[1]. Despite state-of-the-art performance on NLP tasks such as machine translation, text classification, sentiment analysis and speech recognition having made leaps and bounds in the past decade, these systems are still far from acquiring a perfect understanding of natural language. Recently, many new Recurrent Neural Network (RNN) model ideas have been experimented with, like the Clockwork RNN (Koutník et al. (2014)) or the Recurrent Unit with a Stack-like State (RUSS) (Bernardy (2018)), often designed to excel at a specific task. To showcase the new model's superiority, its performance on a task is usually compared to that of a more well-known architecture, such as the Simple RNN (SRNN), the Long Short Term Memory (LSTM) or the Gated Recurrent Unit (GRU).

What is missing from the current state of literature is, however, a robust comparison of these three architectures on a task that adequately showcases their respective ability to perform well on natural language data. I seek to fill that gap with my work by trying to answer the following questions:

1. Can an SRNN, LSTM or GRU architecture learn the Dyck language with two pairs of brackets (D_2)?

2. If they cannot, what poses the highest difficulty in doing so?

3. What influence, if any, does corpus construction have on model performance, specifically generalizability?

The following work consists of five main parts, each of which will be briefly summarized hereunder:

In Chapter 2, I introduce the core concepts relevant for this thesis: formal languages, the complexity of Natural Language, Dyck languages, three neural network architectures and an overview of related works to contextualize my work within the current state of research. I describe model design and training, corpus construction and the two experiments I conduct in this work in Chapter 3. I report the results for each of the experiments in Chapter 4 and discuss them in detail in Chapter 5. Finally,

[1] https://www.tractica.com/newsroom/press-releases/natural-language-processing-market-to-reach-22-3-billion-by-2025/

I seek to answer the research questions posed above with my experimental results in Chapter 6 and suggest further avenues of research on this topic.

2 Theoretical Background

2.1 Formal Languages and Formal Grammars

A formal language $L(G)$ is defined as a subset of all words Σ^* over an alphabet Σ, where all words need to comply with the formal grammar G. As per Jurafsky and 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. Following the two definitions, $L(G)$ consists of all strings w that can be derived from the start symbol S in a finite number of steps, formally $\{w \in \Sigma^* | S \overset{*}{\underset{G}{\Rightarrow}} w\}$. 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 four 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.

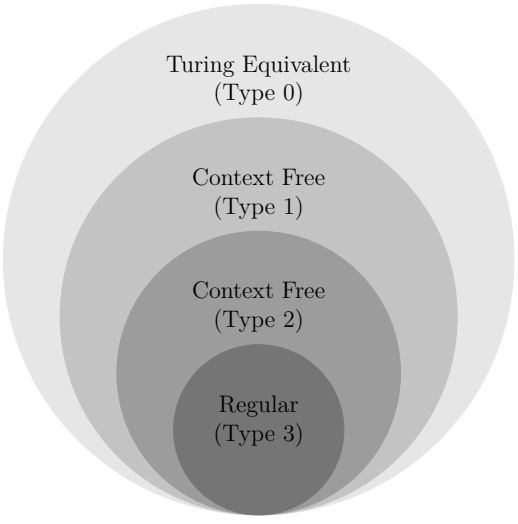

Figure 1: A visual representation of the Chomsky Hierarchy.

The four types of formal grammars can be defined by the form their rules can

take. An overview over these rules as per Jurafsky and 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 (i.e. Kleene's Theorem for regular languages and finite 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 modelled, assessing the precise complexity of Natural Language is the subject of ongoing investigation (Fitch et al. (2012), Petersson and Hagoort (2012), Newmeyer and Preston (2014)). Arguments answering that question usually seek to establish lower bounds: If there is a phenomenon in a natural language that cannot be described with a given type of grammar, natural language must be - however slightly - 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.

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 and Martin (2009), augmented with corresponding automata.

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

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.[2]

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 faced 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 and 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.

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 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.'

[2]The proofs for regular languages being closed under intersection and E' not being regular can be found in Hopcroft et al. (2006) and Sipser (2013).

Four finite sets can be constructed from these examples: accusative noun phrases ($A = \{\text{d'chind}, \dots\}$), dative noun phrases ($B = \{\text{em Hans}, \dots\}$), verbs taking accusative objects ($C = \{\text{laa}, \dots\}$) and 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 context-free.[3]

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.

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

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

[3]The respective proofs for S' not being context-free and context-free sets being closed under intersection with regular sets can be found in Hopcroft et al. (2006) and Sipser (2013).

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 and 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. 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 comparing neural network performance.

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 design: the RNN's output at a time step t is fed back as its input at the following time step $t+1$. Not only does this enable RNNs to process sequences of arbitrary length, it also makes every output dependent on the previous computation as well as the current input. This property equips RNNs with a "memory" for 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:

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

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

The function f constitutes a non-linear transformation, like tanh or ReLU. $\boldsymbol{W_{xh}}$, $\boldsymbol{W_{hh}}$, $\boldsymbol{W_{hy}}$ 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. 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 and 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.

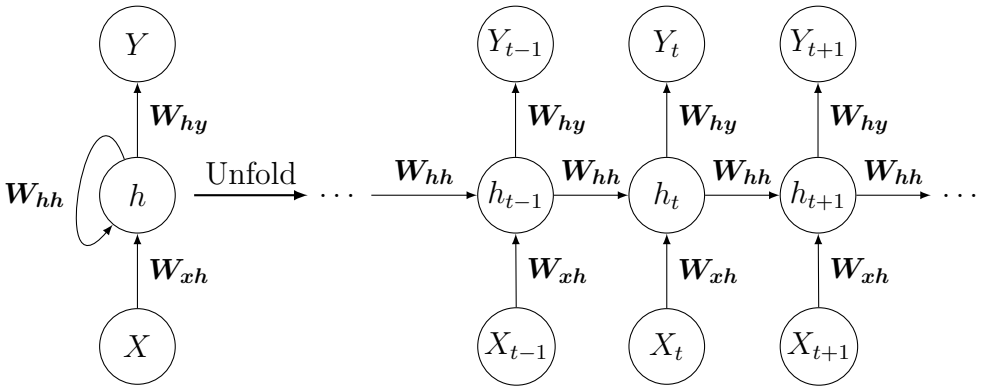

Figure 2: An RNN, unfolded through time.

2.4.2 LSTM

Long-Short Term Memory networks (LSTM) were designed by Hochreiter and 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 gates consisting of a sigmoid layer σ each: input (in), output (out) and forget (forget). 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. The sigmoid layer transforms every value in the concatenated vector to a value in range $[0, \ldots, 1]$ - a 0 translates to forgetting the information, while a 1 passes it through completely. Thus, the output of the gates determines what information is let through the input gate, passed through the output gate or forgotten by the self-connected linear unit.

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

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

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

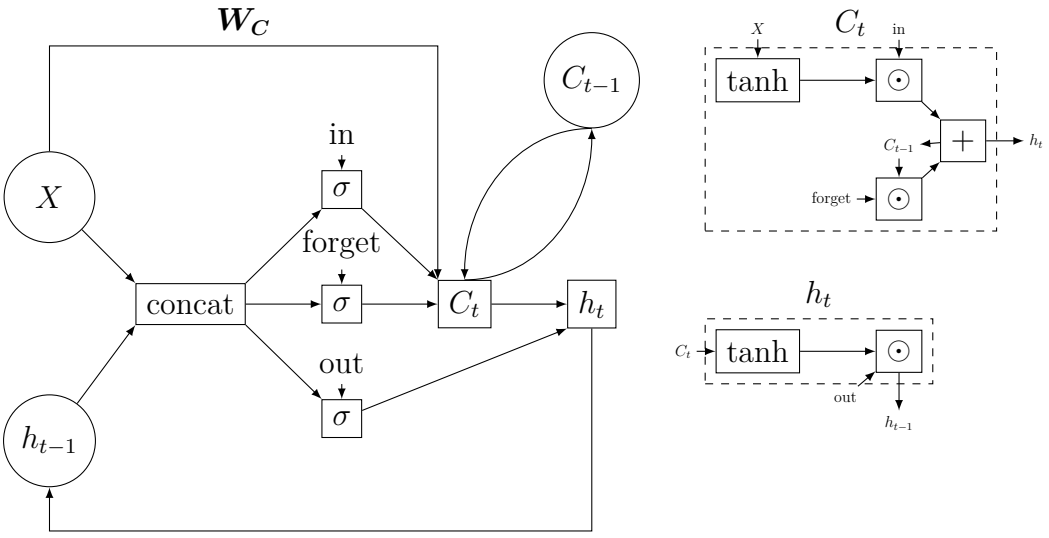

Figure 3: An LSTM memory cell.

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.

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

2.5 Related Works

Currently, NLP models are trained and tested on vast datasets, such as the CoNLL Shared Tasks. By evaluating a variety of different models and approaches on the same data, it is possible to easily assess which one poses the current state-of-the-art for any given NLP task.

The earliest research on neural network architectures was done before such datasets were widely available and easy to process (Cleeremans et al. (1989), Elman (1990),

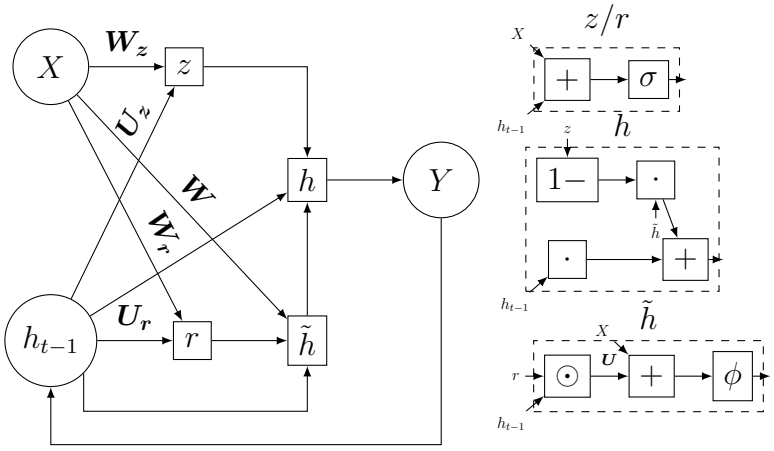

Figure 4: Illustration of a GRU.

Paper	D_n	*Grammar Probability*	*Training Corpus Size*
Deleu and Dureau (2016)	1	equal	unclear
Bernardy (2018)	1, 5	equal	102,400
Li et al. (2018)	1	modified	200 - 20,000
Skachkova et al. (2018)	1-5	modified	131,072
Sennhauser and 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.

Zeng et al. (1994), Hochreiter and Schmidhuber (1997), Rodriguez and Wiles (1998), Gers and Schmidhuber (2001)). During that time, novel architectures and algorithms were mostly scored on formal language datasets, with the test set containing longer words than the training set to assess learning success. However, evaluating on formal languages comes with its own advantages and challenges.

Primarily, it is undeniably cheaper than scoring on a natural language dataset. By deriving words from the grammar, datasets of arbitrary length with arbitrary properties can be generated. However, performance on a formal grammar dataset should always be understood as a simplified benchmark. As mentioned in Sections 2.2.1 and 2.2.2, the formal complexity of natural language is debatable, limiting the significance of formal benchmark performance for NLP tasks. Nevertheless, formal language datasets are still used to evaluate the performance of novel architectures to this day, as done by Joulin and Mikolov (2015), Bernardy (2018), Deleu and Dureau (2016), Li et al. (2018) and Yu et al. (2019).

In addition to the exploration of new architectures, formal languages are also still used to investigate particular behaviours of well-established architectures, such as LSTMs (Sennhauser and Berwick (2018)), or to compare several established models on a specific set of tasks (Skachkova et al. (2018), Suzgun et al. (2019)).

Paper	*Accuracy*	*Perplexity*	*Cell State*	*AUC*	*Error Rate*
Deleu and Dureau (2016)	No	No	No	Yes	No
Bernardy (2018)	Yes	No	No	No	No
Skachkova et al. (2018)	Yes	Yes	No	No	No
Sennhauser and 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 3: 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.

Paper	*Architectures*
Deleu and Dureau (2016)	Neural Turing Machine, LSTM
Bernardy (2018)	GRU, LSTM, RUSS
Skachkova et al. (2018)	SRNN, GRU, LSTM
Sennhauser and Berwick (2018)	LSTM
Suzgun et al. (2019)	SRNN, GRU, LSTM
Yu et al. (2019)	seq2seq

Table 4: Overview of investigated models.

While all these papers use a formal language to evaluate models, several factors prevent them from forming a solid basis upon which to compare their respective results: First, there is neither a benchmark train/dev/test set for Dyck languages (as is standard for most machine learning tasks) (Table 2), nor a set of measures that is reported consistently throughout the literature (Table 3). Additionally, only two papers compare the three well-established architectures (SRNN, GRU, LSTM) directly (Table 4). Finally, the employed training and test measures are not unified - in Bernardy (2018), for example, the models are trained to predict the next letter in words of variable length at any given time step, while in Suzgun et al. (2019), they predict the final letter of a word. Further model details, such as the inclusion of an Embedding and/or a Dropout layer or the number of hidden units also vary.

In conclusion, despite formal languages having been used to assess neural network model performance for decades, there is little to no comparative studies of SRNN, LSTM and GRU models performing on D_2. Any research comparing new architectures to any of these models does so with both varying training and testing methods, as well as with vastly differing corpora and as such cannot be directly compared with each other. This allows for no conclusive statement on the relative performance of these popular RNN architectures based on the current literature.

3 Experiment Setup

3.1 Evaluation

Generating the words for the two experiments follows the procedure described by Bernardy (2018) and will be explained in depth in the coming sections. Whereas the research put forth in his paper scrutinized the generative abilities of RNNs, I am investigating the models performing a classification task. As such, my training, test, validation and experiment data all consist of the same 1:1 ratio of correct-to-incorrect words. Within the incorrect words, a distinction between superfluous opening or closing brackets is made, also at a ratio of 1:1.

As such, a random guessing strategy would yield a baseline accuracy of 50%. A model is considered as having learned useful features from the training data if it scores above the baseline accuracy in the experiments. Furthermore, if the model has learned the underlying grammar of D_2, there should be no difference in accuracy for the two classes of incorrect words, as both of them do not belong to D_2, regardless of which bracket is replaced.

3.2 Models

For the following experiments, the three RNN architectures described in Section 2.4 have been used. All models consist of an embedding layer, a single layer of size $n = \{2^1, 2^2, \ldots, 2^n\}$ and a dense layer of size 1 with sigmoid activation. The activation value of the neuron in the dense layer acts as the output of the model: a value ≥ 0.5 means the model classified the input as a correct word. The models were implemented in Tensorflow 2.0.[4]

All models were trained with the same parameters. The training data was received one word at a time, in batches of 512. The loss was computed by binary cross-entropy, as is current standard for binary classification tasks. Furthermore, the Adam optimizer (Kingma and Ba (2014)) was applied with a learning rate of 0.0001. At the end of a training epoch, the models were evaluated for loss and accuracy on a validation set of 120,000 words. The models were trained until their loss on the validation set did not lower by more than 0.0001 for three consecutive epochs or for at most 100 epochs. The models with the lowest validation loss were used for all experiments.

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 $9 \times 3 \times 3 = 81$ (number of different hidden units $\times$ number of training corpora $\times$ number of different architectures) evaluated models.

[4]The source code can be found at `https://github.com/FyDob/BSc-Thesis`.

Corpus	*Word Length*	*maxND*	*maxBD*
Baseline	18.37 (6.36)	4.31 (1.22)	13.00 (16.02)
High LRD	18.67 (4.75)	5.12 (1.04)	16.67 (4.75)
Low LRD	17.54 (8.04)	3.92 (0.98)	10.58 (9.02)

Table 5: Properties of the three corpora the models were trained on, reported in averages (variance in brackets).

3.3 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, maximum nesting depth (ND) and the maximum distance between a pair of opening and closing brackets (BD). ND is, in this case, defined as the highest number of unresolved open brackets preceding an open bracket in a given word (i.e. in the word `{[{}]}`, the square open bracket is at ND= 1, and the curly open bracket is at ND= 2, making the maximum ND of the word 2). Maximum BD, then, is the highest number of characters between a pair of brackets in a word. In the previous example word, the maximum BD would be 4. These measures are reported in Table 5 in terms of averages and variance.

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 known fact about the size of D_n subsets was utilized: a Dyck language D_n contains $n^m C_m$ 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 and 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, 500,000 words in $D_2^{\leq 20}$ were generated. These words served as the basis for creating the three corpora. To create the Low LRD corpus, all words with a maximum bracket distance higher than 10 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 19 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.

Finally, the corpora were filled with 500,000 non-words obtained by corrupting the correct words in $D_2^{\leq 20}$. For one half of the words, a random opening bracket was

[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.

replaced with a random closing bracket, while a random closing bracket was replaced with a random opening bracket for the other half.

In total, all corpora consist of 1,000,000 samples, of which 50% are incorrect.

3.4 Experiment 1: Long-Range Dependency

For this experiment, the test set consisted of 1,000,000 samples of length 1+18+18+1 = 38, half of which were correct Dyck words. They were created by picking two random Dyck words $w_1, w_2 \in D_2^{=18}$ from the base corpus, concatenating them and wrapping the result in a randomly selected pair of matching brackets as follows:

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

To generate incorrect samples, the generated correct LRD words were corrupted in the same way as for the training corpora, yielding 250,000 incorrect LRD words with a superfluous opening or closing bracket each.

While w_1 and w_2 might have been seen in training (for models trained on the base corpus), 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 classification accuracy serves as a strong indication of the model having learned to generalize to longer, non-concatenated Dyck words.[7] I report model performance on Experiment 1 in terms of accuracy, precision, recall and F1 score.

3.5 Experiment 2: New Depths

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. Since Experiment 1 already investigates Long-Range Dependency (LRD), this corpus was designed so its results are confounded as little as possible by LRD performance.

For this task, the test set consisted of 1,000,000 samples of length 30, half of which were correct Dyck words. First, $500,000$ correct words were chosen at random from the base corpus. Then, they were wrapped by a prefix of five randomly chosen opening brackets and a suffix of the corresponding closing brackets as follows:

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

Generation of incorrect samples was done in accordance to Experiment 1 and corpus creation.

[7] While the infixed sub-words are indeed concatenated, w_{LRD} cannot be created by concatenating two shorter words due to being wrapped by a matching bracket pair.

This process still has the model extrapolate beyond the length of the training words, while increasing all present nesting depths by 5. This is analogous to center embedding in natural language - processing increasing nesting levels is more complicated than processing a flat structure. A high classification accuracy in Experiment 2 indicates a capability to generalize to repeated application of grammar rules beyond what was seen in the training set. As such, it implies an understanding of the D_2 grammar. I report model performance on Experiment 2 in terms of accuracy, precision, recall and F1 score.

4 Results

During training, almost all 81 trained models achieved a validation accuracy significantly above random guessing, except for the SRNN-2 models trained on the base and low LRD corpus, which scored 75.0% and 50.4% respectively. I have included them in the experiments regardless of their low validation accuracy, since it was unclear whether validation accuracy was a strong predictor for a network's performance on the experiment data. I present my results with regard to three focus points: First, the overall performance of different architectures with respect to which corpus they were trained on, then the individual model performances on each of the two experiments, and finally a closer look at classifications made by outlier networks - networks which drastically over- or underperformed in either of the experiments - with regards to word features.

4.1 Architecture/Training Data

As can be seen in Table 6, none of the architectures consistently achieved an accuracy far above the random guessing baseline of 50.0%. However, there was still a notable difference in performance between architectures: on average, the GRU networks scored the highest on accuracy and precision, while the SRNN networks achieved the best recall and F1 score. With 51.5%, LSTMs scored an average accuracy between SRNNs (50.0%) and GRUs (53.3%), but they underperformed in all other experiment measures.

Furthermore, the choice of training data had a notable effect on overall model performance: SRNNs and GRUs received a boost in performance in all measures when comparing the Base to the Low LRD models, elevating SRNNs from an accuracy below random guessing to 51.4%. While LSTMs lost 1.4% in terms of accuracy, all other performance measures improved significantly for the Low LRD models. Training on the High LRD corpus aided SRNNs in terms of accuracy, precision and F1 score, but worsened accuracy, recall and F1 score for LSTMs and GRUs.

	Accuracy	*Precision*	*Recall*	*F1 Score*	*Val Acc*
SRNN			Base		
Mean	*0.476*	*0.278*	*0.181*	*0.193*	0.972
Variance	0.064	0.235	0.257	0.220	0.081
SRNN			Low LRD		
Mean	**0.514**	**0.443**	**0.288**	**0.316**	*0.932*
Variance	0.055	0.176	0.259	0.213	0.158
SRNN			High LRD		
Mean	0.511	0.415	0.153	0.200	**0.981**
Variance	0.043	0.211	0.171	0.187	0.054
SRNN			Complete		
Mean	*0.500*	0.379	**0.208**	**0.236**	*0.962*
Variance	0.056	0.217	0.236	0.211	0.107
LSTM			Base		
Mean	**0.543**	*0.219*	0.148	0.154	0.999
Variance	0.173	0.365	0.335	0.341	0.003
LSTM			Low LRD		
Mean	0.529	**0.391**	**0.176**	**0.195**	0.999
Variance	0.158	0.295	0.311	0.302	0.002
LSTM			High LRD		
Mean	*0.472*	0.258	*0.036*	*0.059*	**1.000**
Variance	0.075	0.278	0.057	0.090	0.000
LSTM			Complete		
Mean	0.515	*0.289*	*0.120*	*0.136*	**0.999**
Variance	0.143	0.318	0.268	0.269	0.002
GRU			Base		
Mean	0.531	*0.371*	0.147	0.185	0.999
Variance	0.097	0.297	0.226	0.247	0.001
GRU			Low LRD		
Mean	**0.554**	**0.507**	**0.206**	**0.243**	0.999
Variance	0.133	0.198	0.302	0.285	0.001
GRU			High LRD		
Mean	*0.514*	0.439	*0.126*	*0.170*	*0.972*
Variance	0.082	0.223	0.198	0.193	0.055
GRU			Complete		
Mean	**0.533**	**0.439**	0.160	0.200	0.990
Variance	0.106	0.245	0.244	0.242	0.034

Table 6: Performance measures of all architectures across both experiments depending on which corpus they were trained on, as well as the compounded measures for all networks regardless of training data.

Network	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*Val Acc*
GRU-2	0.890	**0.982**	0.794	0.878	0.995
GRU-4	0.487	0.413	0.060	0.105	1.000
GRU-8	0.488	0.331	0.023	0.043	1.000
GRU-16	0.550	0.713	0.168	0.272	1.000
GRU-32	0.434	0.267	0.075	0.118	1.000
GRU-64	0.510	0.537	0.149	0.234	1.000
GRU-128	0.553	0.611	0.293	0.396	0.999
GRU-256	0.497	0.364	0.009	0.018	1.000
GRU-512	0.500	0.487	0.007	0.015	1.000
LSTM-2	0.500	0.000	0.000	0.000	0.999
LSTM-4	0.451	0.000	0.000	0.000	1.000
LSTM-8	**0.910**	0.959	**0.857**	**0.905**	**1.000**
LSTM-16	0.343	0.001	0.000	0.000	1.000
LSTM-32	0.500	0.000	0.000	0.000	1.000
LSTM-64	0.505	0.596	0.030	0.057	1.000
LSTM-128	0.347	0.001	0.000	0.000	1.000
LSTM-256	0.455	0.128	0.016	0.028	1.000
LSTM-512	0.499	0.468	0.009	0.017	0.991
SRNN-2	0.465	0.357	0.087	0.140	*0.750*
SRNN-4	0.488	0.334	0.023	0.043	1.000
SRNN-8	0.461	0.037	0.003	0.006	1.000
SRNN-16	*0.272*	0.021	0.010	0.014	1.000
SRNN-32	0.498	0.492	0.141	0.219	1.000
SRNN-64	0.504	0.505	0.366	0.424	1.000
SRNN-128	0.484	0.040	0.001	0.003	1.000
SRNN-256	0.503	0.503	0.455	0.478	1.000
SRNN-512	0.484	0.017	0.001	0.001	1.000
Mean	0.503	0.339	0.132	0.163	0.990
Variance	0.130	0.294	0.233	0.253	0.048

Table 7: Performance measures for experiment 1 of all networks that were trained on the Base LRD corpus.

4.2 Experiment 1: Long-Range Dependency

I report results for Experiment 1 in Tables 7, 8 and 9, which include the performance measures for all networks trained on the Base, Low and High LRD corpus respectively, as evaluated on Experiment 1.

When trained on the Base corpus, 8 of 27 models (29.6%) achieved an accuracy above random guessing. Among those 8, only 2 reached an accuracy above 60%: LSTM-8 and GRU-2. The vast majority of models - 20 in total - reached an accuracy of $50 \pm 5\%$. 4 performed even worse than that: the worst model (SRNN-16) only achieved 27.2% accuracy on the experiment data. There was no apparent correlation between validation accuracy and model performance in Experiment 1 - indeed, SRNN-2 with the lowest validation accuracy at 75.0% evaluated at below chance, but so did

Network	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*Val Acc*
GRU-2	0.583	0.582	0.589	0.585	1.000
GRU-4	0.505	0.518	0.135	0.214	1.000
GRU-8	0.498	0.491	0.106	0.175	0.997
GRU-16	0.503	0.528	0.060	0.107	1.000
GRU-32	0.505	0.525	0.100	0.168	1.000
GRU-64	**0.890**	**0.869**	**0.918**	**0.893**	**1.000**
GRU-128	0.484	0.324	0.030	0.054	1.000
GRU-256	0.500	0.496	0.039	0.072	1.000
GRU-512	0.500	0.435	0.002	0.004	0.997
LSTM-2	0.500	0.500	0.475	0.487	0.996
LSTM-4	*0.278*	*0.000*	*0.000*	*0.000*	*1.000*
LSTM-8	0.500	0.000	0.000	0.000	1.000
LSTM-16	0.881	0.853	0.920	0.885	1.000
LSTM-32	0.491	0.092	0.002	0.004	1.000
LSTM-64	0.500	0.505	0.010	0.020	1.000
LSTM-128	0.496	0.374	0.011	0.021	0.998
LSTM-256	0.500	0.535	0.007	0.014	1.000
LSTM-512	0.507	0.566	0.058	0.106	1.000
SRNN-2	0.501	0.501	0.366	0.423	0.*504*
SRNN-4	0.708	0.646	0.922	0.760	0.930
SRNN-8	0.508	0.527	0.150	0.234	1.000
SRNN-16	0.500	0.000	0.000	0.000	1.000
SRNN-32	0.486	0.388	0.050	0.088	1.000
SRNN-64	0.487	0.375	0.040	0.072	1.000
SRNN-128	0.492	0.484	0.236	0.318	0.953
SRNN-256	0.501	0.501	0.412	0.452	1.000
SRNN-512	0.503	0.503	0.497	0.500	1.000
Mean	0.530	0.449	0.227	0.246	0.977
Variance	0.120	0.217	0.304	0.280	0.096

Table 8: Performance measures for experiment 1 of all networks that were trained on the Low LRD corpus.

several models with a validation accuracy of 100%.

12 of 27 Low LRD-trained models (44.4%) scored an accuracy higher than the random guessing baseline. Among those, 3 models - SRNN-4, LSTM-16 and GRU-64 - reached an accuracy above 60%. While 22 models fell within the $50 \pm 5\%$ belt of accuracy, only 1 model performed significantly worse: LSTM-4 with 27.8%. Validation accuracy was entirely unrelated to model performance, with LSTM-4 having achieved a perfect score on the validation data, but completely failing at Experiment 1. On average, all measures have improved when compared to the Base models: accuracy improved by +2.7 percentage points (p.p.), precision by +11.0 p.p., recall by +9.5 p.p. and F1 score by +8.3 p.p.

9 of 27 High LRD-trained models (33.3%) achieved an accuracy above baseline,

Network	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*Val Acc*
GRU-2	0.494	0.480	0.136	0.212	1.000
GRU-4	0.499	0.482	0.036	0.066	1.000
GRU-8	0.342	0.000	0.000	0.000	1.000
GRU-16	0.523	0.590	0.153	0.242	1.000
GRU-32	0.504	0.521	0.093	0.158	1.000
GRU-64	0.504	0.522	0.088	0.150	1.000
GRU-128	0.489	0.419	0.056	0.098	1.000
GRU-256	0.503	0.549	0.035	0.066	0.902
GRU-512	**0.800**	**0.768**	**0.861**	**0.812**	0.849
LSTM-2	0.483	0.390	0.060	0.104	1.000
LSTM-4	*0.282*	*0.000*	*0.000*	*0.000*	1.000
LSTM-8	0.379	0.000	0.000	0.000	1.000
LSTM-16	0.502	0.640	0.007	0.015	1.000
LSTM-32	0.498	0.475	0.038	0.070	1.000
LSTM-64	0.500	0.000	0.000	0.000	1.000
LSTM-128	0.492	0.365	0.022	0.041	1.000
LSTM-256	0.497	0.367	0.008	0.016	1.000
LSTM-512	0.500	0.000	0.000	0.000	0.999
SRNN-2	0.500	0.499	0.042	0.078	*0.833*
SRNN-4	0.502	0.523	0.046	0.085	0.996
SRNN-8	0.539	0.622	0.201	0.304	1.000
SRNN-16	0.500	0.000	0.000	0.000	0.999
SRNN-32	0.500	0.497	0.037	0.070	1.000
SRNN-64	0.503	0.504	0.360	0.420	1.000
SRNN-128	0.497	0.494	0.235	0.318	1.000
SRNN-256	0.496	0.495	0.340	0.403	1.000
SRNN-512	0.490	0.459	0.112	0.180	1.000
Mean	0.493	0.395	0.110	0.145	0.984
Variance	0.083	0.230	0.180	0.182	0.045

Table 9: Performance measures for experiment 1 of all networks that were trained on the High LRD corpus.

but only GRU-512 by a significant margin with 80%. 23 models performed within the ±5% margin around the baseline, and 3 models (LSTM-4, LSTM-8 and GRU-8) underperfomed significantly. There was no relation between validation accuracy and performance on experiment data for High LRD models, either. Compared to the Base models, High LRD models almost consistently scored worse: accuracy, recall and F1 score went down by −1.0, −2.2 and −1.8 p.p. respectively, but precision was improved by +5.6 p.p.

4.3 Experiment 2: New Depths

All results for Experiment 2 can be found in Tables 10, 11 and 12. When comparing mean performances across all models, they largely scored higher on Experiment 2 than Experiment 1. This suggests that Experiment 2 was easier regardless of training data. As with Experiment 1, a model's validation accuracy did not correlate with its performance on the experiment data.

Among the Base models, 8 of 27 (29.6%) performed above guessing baseline in Experiment 2 - the same ratio as for Experiment 1, though there was minimal overlap in the best performers. Only GRU-64 and SRNN-64 performed above 50% accuracy for both experiments. 20 of 27 models stayed in the ±5% margin of the baseline, with only 2 (SRNN-128 and SRNN-256) dropping below that. The best performing network - LSTM-128 - scored the highest accuracy across all models and all experiments with 99.3%.

The same number of Low LRD models performed above chance for Experiment 2 as for Experiment 1 (44.4%), with 6 models (SRNN-2, SRNN-8, LSTM-16, LSTM-512, GRU-4 and GRU-64) occuring in both groups. LSTM-512 and GRU-64 have both achieved an accuracy $> 90\%$. 19 models performed within the ±5% margin around the baseline, only 2 dropped below that. Compared to the Base models, low LRD models also performed better on Experiment 2: mean accuracy is up by +0.4 p.p., precision by +20.6, recall by +3.5 and F1 score by 6.5 p.p.

While the highest number of High LRD models have achieved an accuracy above random guessing - 10 of 27, or 37.0% - only SRNN-16 crossed the 60% threshold at all. Indeed, aside from SRNN-16, only one other model lay outside of the ±5% margin around the baseline - LSTM-64, with an accuracy of 28.5%. When comparing with the Base models, all measures except for precision, which improved by +10.7 p.p. Accuracy went down by −2.7 p.p., recall by −7.5 and F1 score by −5.0 p.p.

Network	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*Val Acc*
GRU-2	0.500	0.000	0.000	0.000	0.995
GRU-4	0.563	0.655	0.265	0.377	1.000
GRU-8	0.472	0.164	0.014	0.025	1.000
GRU-16	0.500	0.000	0.000	0.000	1.000
GRU-32	0.500	0.000	0.000	0.000	1.000
GRU-64	0.515	0.550	0.160	0.248	1.000
GRU-128	0.496	0.000	0.000	0.000	0.999
GRU-256	0.500	0.000	0.000	0.000	1.000
GRU-512	0.601	0.597	0.620	0.608	1.000
LSTM-2	0.500	0.000	0.000	0.000	0.999
LSTM-4	0.500	0.000	0.000	0.000	1.000
LSTM-8	0.500	0.000	0.000	0.000	1.000
LSTM-16	0.500	0.000	0.000	0.000	1.000
LSTM-32	0.778	0.792	0.755	0.773	1.000
LSTM-64	0.500	0.000	0.000	0.000	1.000
LSTM-128	**0.993**	**0.991**	**0.995**	**0.993**	**1.000**
LSTM-256	0.500	0.000	0.000	0.000	1.000
LSTM-512	0.499	0.000	0.000	0.000	0.991
SRNN-2	0.580	0.548	0.917	0.686	*0.750*
SRNN-4	0.488	0.000	0.000	0.000	1.000
SRNN-8	0.500	0.000	0.000	0.000	1.000
SRNN-16	0.500	0.000	0.000	0.000	1.000
SRNN-32	0.515	0.564	0.136	0.220	1.000
SRNN-64	0.511	0.529	0.192	0.281	1.000
SRNN-128	0.441	0.409	0.266	0.322	1.000
SRNN-256	*0.382*	0.156	0.053	0.080	1.000
SRNN-512	0.497	0.497	0.615	0.550	1.000
Mean	0.531	0.239	0.185	0.191	0.990
Variance	0.113	0.311	0.308	0.290	0.048

Table 10: Performance measures for experiment 2 of all networks that were trained on the Base LRD corpus.

Network	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*Val Acc*
GRU-2	0.494	0.488	0.222	0.306	1.000
GRU-4	0.561	0.651	0.262	0.374	1.000
GRU-8	0.499	0.492	0.056	0.101	0.997
GRU-16	0.497	0.465	0.039	0.072	1.000
GRU-32	0.500	0.000	0.000	0.000	1.000
GRU-64	0.935	0.910	0.966	0.937	1.000
GRU-128	0.516	0.560	0.144	0.229	1.000
GRU-256	0.499	0.486	0.043	0.078	1.000
GRU-512	0.497	0.305	0.004	0.008	0.997
LSTM-2	0.499	0.000	0.000	0.000	0.996
LSTM-4	0.480	0.328	0.037	0.067	1.000
LSTM-8	0.555	0.660	0.226	0.337	1.000
LSTM-16	0.566	0.662	0.270	0.384	1.000
LSTM-32	*0.327*	0.207	0.122	0.154	1.000
LSTM-64	0.491	0.301	0.014	0.026	1.000
LSTM-128	0.500	0.000	0.000	0.000	0.998
LSTM-256	0.502	0.538	0.027	0.051	1.000
LSTM-512	**0.953**	**0.917**	**0.996**	**0.955**	**1.000**
SRNN-2	0.501	0.501	0.585	0.540	*0.504*
SRNN-4	0.442	0.361	0.149	0.211	0.930
SRNN-8	0.529	0.582	0.206	0.304	1.000
SRNN-16	0.498	0.000	0.000	0.000	1.000
SRNN-32	0.580	0.561	0.738	0.637	1.000
SRNN-64	0.514	0.546	0.170	0.260	1.000
SRNN-128	0.498	0.495	0.178	0.262	0.953
SRNN-256	0.500	0.499	0.348	0.410	1.000
SRNN-512	0.501	0.505	0.136	0.214	1.000
Mean	0.535	0.445	0.220	0.256	0.977
Variance	0.126	0.246	0.281	0.263	0.096

Table 11: Performance measures for experiment 2 of all networks that were trained on the Low LRD corpus.

Network	*Accuracy*	*Precision*	*Recall*	*F1 Score*	*Val Acc*
GRU-2	0.535	0.580	0.253	0.353	1.000
GRU-4	0.502	0.518	0.070	0.123	1.000
GRU-8	0.500	0.000	0.000	0.000	1.000
GRU-16	0.492	0.269	0.010	0.019	1.000
GRU-32	0.511	0.552	0.114	0.189	1.000
GRU-64	0.502	0.512	0.081	0.140	1.000
GRU-128	0.541	0.601	0.244	0.347	1.000
GRU-256	0.503	0.530	0.048	0.089	0.902
GRU-512	0.500	0.000	0.000	0.000	0.849
LSTM-2	0.531	0.591	0.200	0.299	1.000
LSTM-4	0.500	0.000	0.000	0.000	1.000
LSTM-8	0.500	0.000	0.000	0.000	1.000
LSTM-16	0.517	0.658	0.070	0.127	1.000
LSTM-32	0.511	0.554	0.112	0.187	1.000
LSTM-64	*0.285*	0.000	0.000	0.000	1.000
LSTM-128	0.500	0.000	0.000	0.000	1.000
LSTM-256	0.521	0.608	0.121	0.202	1.000
LSTM-512	0.500	0.000	0.000	0.000	0.999
SRNN-2	0.491	0.333	0.017	0.033	*0.833*
SRNN-4	0.500	0.000	0.000	0.000	0.996
SRNN-8	0.500	0.000	0.000	0.000	1.000
SRNN-16	**0.675**	**0.691**	**0.633**	**0.661**	0.999
SRNN-32	0.507	0.557	0.072	0.128	1.000
SRNN-64	0.492	0.473	0.139	0.215	1.000
SRNN-128	0.485	0.281	0.019	0.036	1.000
SRNN-256	0.501	0.502	0.322	0.392	1.000
SRNN-512	0.512	0.534	0.183	0.273	1.000
Mean	0.504	0.346	0.100	0.141	0.984
Variance	0.056	0.267	0.141	0.164	0.045

Table 12: Performance measures for experiment 2 of all networks that were trained on the High LRD corpus.

5 Discussion

6 Conclusion

This work has set out to answer three questions, as posed in Chapter 1. Related literature has been consulted to choose a proper approach. However, current literature contains neither a benchmark dataset to train and test models on, nor a unified set of tasks and measures to do so. Due to these facts, most results in current literature discussing model performance on D_2 are incomparable to each other.

Bibliography

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

Bach, E., Brown, C., and 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., and 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., and 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. and Schützenberger, M. (1963). The algebraic theory of context-free languages*. In Braffort, P. and 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., and Mcclelland, J. (1989). Finite state automata and simple recurrent networks. *Neural Computation - NECO*, 1:372–381.

Deleu, T. and 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.

Fitch, W. T., Friederici, A. D., and Hagoort, P. (2012). Pattern perception and computational complexity: introduction to the special issue. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 367(1598):1925–1932.

Frank, S. L., Trompenaars, T., and 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. and 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. and 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. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. Addison-Wesley Longman Publishing Co., Inc., USA.

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

Jurafsky, D. and 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.

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

Koutník, J., Greff, K., Gomez, F., and Schmidhuber, J. (2014). A clockwork rnn.

Li, T., Rabusseau, G., and Precup, D. (2018). Nonlinear weighted finite automata. In Storkey, A. and 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.

Newmeyer, F. and Preston, L. (2014). *Measuring Grammatical Complexity*. Oxford linguistics. Oxford University Press.

Petersson, K. M. and Hagoort, P. (2012). The neurobiology of syntax: Beyond string sets. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, 367:1971–83.

Rodriguez, P. and 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. and 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.

Sipser, M. (2013). *Introduction to the Theory of Computation*. Course Technology, Boston, MA, third edition.

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

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

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

Yu, X., Vu, N. T., and 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., and 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.

$Unterschrift:$ $Ort, Datum:$