UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

COMPUTERVISUALISTIK

Deep Learning techniques applied to Constituency parsing of German

Master's Thesis
in partial fulfillment of the requirements for the degree of
Master of Science (M.Sc.) in Web Science

Submitted by
Kandhasamy Rajasekaran

Betreuer: Prof. Dr. Karin Harbusch, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Erstgutachter: Prof. Dr. Karin Harbusch, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Zweitgutachter: Denis Memmesheimer, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau

Koblenz, im Januar 2020

Statement

I hereby certify that this thesis has been composed by me and is based on my own work, that I did not use any further resources than specified – in particular no references unmentioned in the reference section – and that I did not submit this thesis to another examination before. The paper submission is identical to the submitted electronic version.

	Yes	No
I agree to have this thesis published in the library.	☐	☐
I agree to have this thesis published on the Web.	☐	☐
The thesis text is available under a Creative Commons License (CC BY-SA 4.0).	☐	☐
The source code is available under a GNU General Public License (GPLv3).	☐	☐
The collected data is available under a Creative Commons License (CC BY-SA 4.0).	☐	☐

. . . . . .

(Place, Date) (Signature)

Note

- If you would like us to contact you for the graduation ceremony,

  please provide your personal E-mail address: .....

- If you would like us to send you an invite to join the WeST Alumni

  and Members group on LinkedIn, please provide your LinkedIn ID : .....

I would like to express my gratitude to my advisors Prof.Dr.Karin Harbusch and Denis Memmesheimer for their support throughout the course of my Master thesis. Thank you.

My mother has been very supportive of me and all those weekly phone calls have helped me to put things in perspective. Thank you.

Abstract

Constituent parsing attempts to extract syntactic structure from a sentence. These parsing systems are helpful in many NLP applications such as grammar checking, question answering, and information extraction. This thesis work is about implementing a constituent parser for German language using neural networks. Over the past, recurrent neural networks have been used in building a parser and also many NLP applications. In this, self-attention neural network modules are used intensively to understand sentences effectively. With multi-layered self-attention networks, constituent parsing achieves 93.68% F1 score. This is improved even further by using both character and word embeddings as a representation of the input. An F1 score of 94.10% was the best achieved by constituent parser using only the dataset provided. With the help of external datasets such as German Wikipedia, pre-trained ELMo models are used along with self-attention networks achieving 95.87% F1 score.

Zusammenfassung

Konstituenten-Parsing versucht, syntaktische Struktur aus einem Satz zu extrahieren. Diese Parsing-Systeme sind in vielen maschinellen Sprachverarbeitungsanwendungen hilfreich, wie z.B. bei der Grammatikprüfung, der Beantwortung von Fragen und der Informationsextraktion. In dieser Masterarbeit geht es um die Implementierung eines Konstituentenparsers für die deutsche Sprache mit Hilfe von neuronalen Netzen. In der Vergangenheit wurden wiederkehrende neuronale Netze beim Aufbau eines Parsers und auch bei vielen maschinellen Sprachverarbeitungsanwendungen verwendet. Dabei werden Module des neuronalen Netzes mit Selbstaufmerksamkeit intensiv genutzt, um Sätze effektiv zu verstehen. Bei mehrschichtigen Selbstaufmerksamkeitsnetzwerken erreicht das konstituierende Parsen 93,68% F1-Scoret. Dies wird noch weiter verbessert, indem sowohl Zeichen- als auch Worteinbettungen als Darstellung des Inputs verwendet werden. Ein F1-Score von 94,10% wurde am besten durch den Konstituenten-Parser erreicht, der nur den bereitgestellten Datensatz verwendet. Mit Hilfe externer Datensätze wie der deutschen Wikipedia werden vortrainierte ELMo-Modelle zusammen mit Selbstbeobachtungsnetzwerken verwendet, die einen F1-Score von 95,87% erreichen.

Contents

List of Figures

List of Tables

1 Introduction

Constituency or Syntactic parsing is the process of determining the syntactic structure of a sentence by analyzing its words based on underlying grammar. Constituency parsing is very important in natural language processing (NLP) because it plays a substantial role in mediating between linguistic expression and meaning (Socher, Bauer, Manning, & Ng, 2013).

Say for e.g. the sentence 'Hans isst einen Apfel' along with parts of speech (POS) tags, the parse tree is as follows:

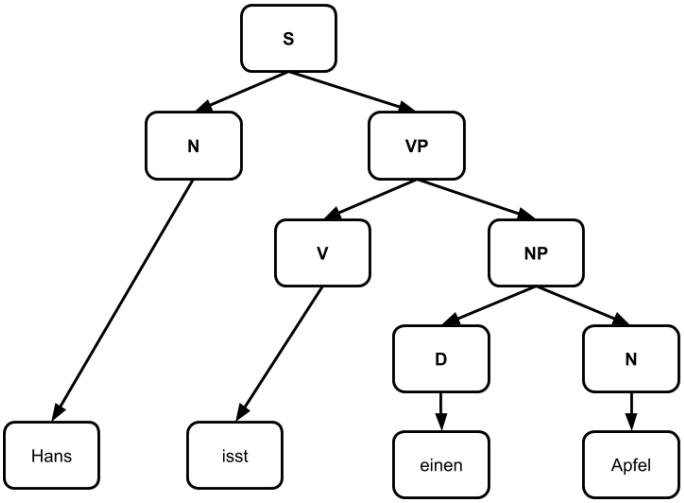

Figure 1: An example parser tree

In this tree, the abbreviations S, D, N, V, NP, and VP refer to sentence, determinant, noun, verb, noun-phrase, and verb-phrase respectively. The words of the sentence are referred as leaf nodes of the tree: Hans, isst, einen and Apfel. The parts of speech (POS) tags associated with the above words are noun(N), verb(V), determinant(D), and noun(N) respectively. These are referred by the immediate parent of each word. The rest of the nodes are formed out of parsing. From the diagram, it is clear that the sentence followed three rules:

$$\mathbf{S} \rightarrow \mathbf{N}\ \mathbf{VP}$$
$$\mathbf{VP} \rightarrow \mathbf{V}\ \mathbf{NP}$$
$$\mathbf{NP} \rightarrow \mathbf{D}\ \mathbf{N}$$

Jurafsky et al. (Jurafsky & Martin, 2008) in their 'Speech and Language processing' book stated that the parse tree is useful in many applications related to NLP. They are directly useful in grammar checking in word processing systems; a sentence that cannot be parsed is supposed to have grammatical errors. Most of the time, it is also an important intermediate representation for the semantic analysis of a sentence. Thus it plays an important role in applications such as relation extraction, semantic role labeling (Gildea & Palmer, 2002), question answering and paraphrase detection (Callison-Burch, 2010).

In this thesis, we will apply state of the art techniques to perform constituent parsing for the German language.

2 Related Work

In the past, handwritten grammar played a central role in parsing. Context-free grammars (CFG) were used to model the rules and patterns of natural language. But given the complexity of a natural language, neither simple, broad-coverage grammar rules nor complex, constrained grammars were able to optimally accommodate sentences. A specific complex grammar with a lot of constraints was failing to parse a lot of sentences. At the same time, a simple and generic grammar was making it possible to have multiple ways of parsing even for a simple sentence. Statistical parsing systems offer a lot of possibilities of many rules to cope up with the flexibility of a language and at the same time has the predictive capabilities to figure out most likely parsing tree for a sentence. To build a statistical parsing system, the need for a versatile dataset is very critical.

Marcus et al.(Marcus, 1993) prepared the Penn Treebank dataset for the English language which consists of syntactic structures for sentences along with POS tags. It was revised multiple times and now it consists of more than 39,000 sentences with syntax trees and 7 million words with POS tags (Taylor, Marcus, & Santorini, 2003). This annotated information includes text from different sources such as IBM computer manuals, nursing notes, Wall Street Journal articles, and transcribed telephone conversations. This served as a good corpus for building models to do parsing based on Machine Learning (ML). This leads to an empirical/data-driven approach to parsing than a rule-based approach. Some of the advantages of using treebanks are reusability of a lot of other subsystems such as POS taggers, parsers, and a standard way to evaluate multiple parsing systems.

Similarly, for the German language, Telljohann et al. associated with the University of Tübingen, prepared a syntactically annotated German newspaper corpus TüBa-D/Z treebank (Telljohann & Hinrichs, 2004), (Telljohann, Hinrichs, Kübler, Kübler, & Tübingen, 2004). These annotations are done manually and consist of 3,816 articles, 104,787 sentences, and 1,959,474 tokens. It mainly distinguishes four levels of syntactic constituency: the lexical level, phrasal level, topological fields level, and clausal level. In addition to that, Brants et al. also prepared a 35,000 syntactically annotated German newspaper sentences named TIGER treebank (Brants, Dipper, Hansen, Lezius, & Smith, 2002). It also contains different levels of annotation such as POS tags, phrase categories, and syntactic functions. These two data sources are very important in the field of NLP for the German language.

Probabilistic context-free grammars (PCFG) (Booth, 1969) contains a probability score assigned to each production rule. These probabilities are assigned based on various methods and the simplest of it is considering the statistical properties of word combinations. They help to quantify the likelihood of different possibilities of parsing a sentence. The Chart based CKY (Younger, 1967) algorithm is a bottom-up parsing methodology that uses these probability scores to decide what combination of constituents is very likely. It only takes cubic time complexity rather than exponential by using dynamic programming techniques.

The head word of a phrase gives a good representation of the phrase structure and meaning (Charniak, 1997). The Lexicalization of PCFG (Charniak, 2000) will create more rules of specific cases and the probability scores can be trained for each production rule. This indeed improves the accuracy of parsing, but it increases production rules exponentially which results in sparseness. Kelin and Manning (Klein & Manning, 2003) found that lexicalized PCFG does not capture lexical selection between content words but just basic grammatical features like verb form, verbal auxiliary, finiteness, etc. This kind of splitting can be done horizontally while binarizing the tree and vertically by parent annotation. This alternative method achieved 85.7% F1 score, but at the same time, does not lead to too many production rules. Until now the rules of grammar were handwritten which were picked carefully by experts. Petrov and Klein (Petrov, Barrett, Thibaux, & Klein, 2006) came up with an unsupervised ML approach that uses base categories and learns subcategorizing and splitting using treebank data. They used expectation maximization (EM) algorithm, like Forward-Backward for HMMs but constrained

by a tree structure and achieved 90.2% F1 score.

Neural networks have been raising and becoming a prominent architecture to build machine learning (ML) systems. They were giving state of the art results for many NLP applications. They are also used to develop a better alternative for the representation of words or other features of text.

The constituent parsing of a given sentence involves outputting a tree which is fundamentally made of recursive structures of branches or merges of words or phrases. Socher et al. [(2011), (2013), (2010)] implemented a neural network-based parsing which is specialized in learning recursive structures in a sentence and simultaneously learning compositional vectors for phrases. Their work achieved an accuracy of 90.4% F1 score for the Penn Treebank and it is 20% faster than Stanford factored parser. The system involves two bottom-up parsing; the first parsing done by a base Probabilistic Context-Free Grammar(PCFG) parser using CKY dynamic programming and select 200 best parses; the second pass is done by their best recursive neural network model with expensive matrix calculations on those 200 best results.

Stern et al. (2017) implemented a minimum span neural network-based constituency parser and achieved an accuracy of 91.79% F1 score on the Penn Treebank dataset. They used an encoder-decoder architecture where the encoder converts the input into a different representation with context information and the decoder uses this augmented information to build a tree and get trained to become better. Bidirectional RNN with LSTM (Schuster & Paliwal, 1997) modules were used to encode the words in a sentence. This encoding gives out two vectors for each word containing the sentential context from the left and right direction. This encoded output is used by two independent scoring systems; one to label the span and the other to choose the split. Their research work involved experimenting with the above computed encoded output with chart-based bottom-up and greedy top-down parsing. Surprisingly the results of using top-down with run time complexity $O(n^2)$ is as good as using a bottom-up approach wherein a global optimized tree is chosen with run time complexity $O(n^3)$. The bottom-up parsing was making decisions at every stage with the already computed values for the whole tree beneath; whereas the top-down parsing can refer to only local information. These results credit the ability of bidirectional RNN modules in capturing a lot of complex relations among words in a sentence. By using an encoder that produces rich and expressive word vector representations, the decoding architecture is made simpler.

Kitaev et al. (2019) also implemented an encoder-decoder based neural network architecture for constituency parsing and they achieved 95.13% F1 score on the Penn Treebank dataset. Their parser also outperforms all the previous parsers on 8 of the 9 languages in SPMRL dataset which includes German. In this, instead of LSTM modules, attention modules (2017) are used. Attention modules are better in expressing how two sentences or inputs are relevant to each other by expressing them in a matrix with each row/column as a word of a sentence. At the same time, attention modules can also be applied to a sentence itself to express which part of the sentence is dependent on which other words. These matrices capture the relationships of words in a sentence among themselves. Several self-attention modules are used in combination to encode the information and context in each sentence. The use of attention makes explicit how information is propagated between different locations in the sentence. A chart-based decoder implemented by Stern et al. (2017) is used along with the modifications from Gaddy et al. (2018). They also found that positional information plays a very important role in parsing and inclusion of ELMo(Embeddings from Language Models) (Peters et al., 2018) improved their accuracy by 3%.

3 Background Study

In this thesis, we develop a neural network based constituent parser for the German language. The required information to understand the different components of the proposed system is explained briefly in this section.

3.1 Neural Networks and its Components

Recently, neural networks have become the most popular approach for building machine learning systems. Neural networks compose many interconnected, fundamental, functional units called neurons. They are loosely inspired from the field of neuroscience. There are different ways by which these neurons can be networked with each other. That defines its ability to learn.

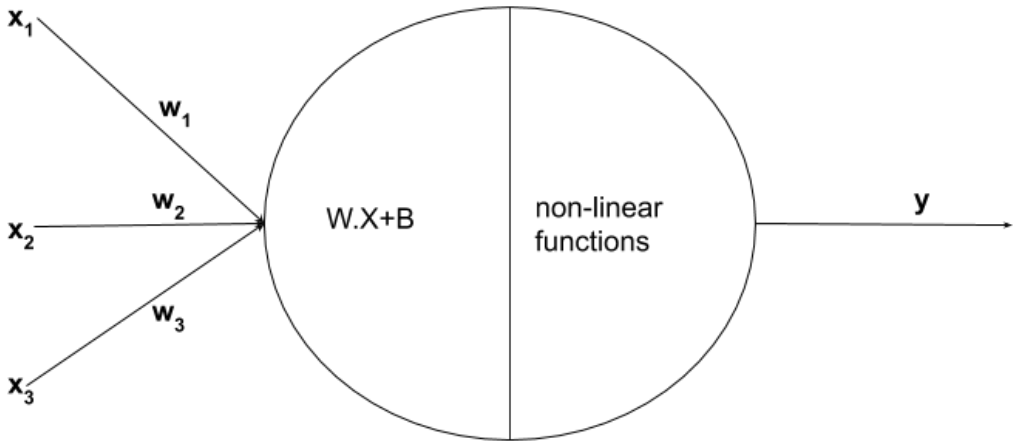

Figure 2: Single Neuron

As shown in figure 2, each neuron in the network takes in multiple scalar inputs and gives out a scalar output. It uses parameters (weights W and bias B) to perform linear transformation of the input and most often applies non-linear function subsequently.

The important components of any machine learning approach are model, data, loss function, and optimizer. The model defines the architecture of system, data helps the model to get trained, loss function helps the system to understand how has it performed during its learning, and optimizer helps the system to fine-tune its parameters of model to get better results.

3.1.1 Feed-forward Network

There are different architectures of neural networks that vary mostly in how the neurons are connected and how the weights are managed. Feed-forward neural network (Svozil, Kvasnieka, & Pospichal, 1997) is a basic neural network architecture which arranges neurons in layers that reflect the flow of information. They are used to perform supervised machine learning wherein label data regarding classification is mandatory. These networks should contain an input layer that takes in data and an output layer which represents prediction of classification. It can have one or many hidden layers and each neuron in one layer is connected with every other neuron in the subsequent layer as given in Figure 3

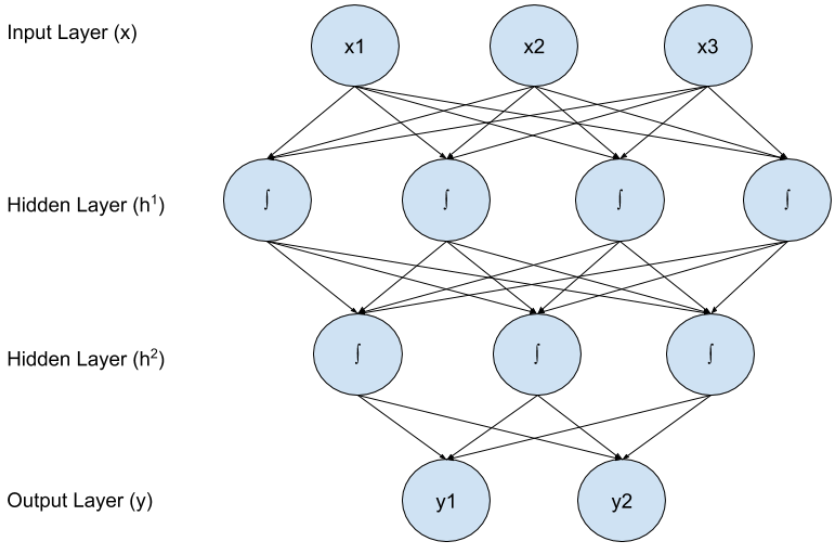

Figure 3: A Feed-Forward neural network.

There are 4 layers in figure 3. Each circle is a neuron with incoming lines as inputs and outgoing lines as outputs to the next layer. Each line carries weight and the input layer has no weights since it has no incoming lines. The input layer consists of 3 neurons and the extracted features of raw data will be sent through these neurons. The first hidden layer consists of 4 neurons, of which each neuron takes 3 inputs from input layer. Each input to the neuron in the first hidden layer is multiplied by a unique weight variable and added together. Finally, the output is added with a bias variable and will be passed to a non-linear activation function as shown in equation 1. The same process is carried out for the second hidden layer except that it has 3 neurons and each neuron will take 4 inputs from the first hidden layer. The output layer consists of 2 neurons and each will take 3 inputs from the second hidden layer as shown in equation 3.

$$h^1 = g^1(xW^1 + b^1) \tag{1}$$

$$h^2 = g^2(h^1W^2 + b^2) \tag{2}$$

$$\mathrm{NN}_{\mathrm{MLP2}}(x) = y = g^3(h^2W^3 + b^3) \tag{3}$$

$$\begin{aligned} x \in \mathbb{R}^{d_{in}}, y \in \mathbb{R}^{d_{out}} \\ W^1 \in \mathbb{R}^{d_{in} \times d_1}, b^1 \in \mathbb{R}^{d_1}, h1 \in \mathbb{R}^{d_1} \\ W^2 \in \mathbb{R}^{d_1 \times d_2}, b^2 \in \mathbb{R}^{d_2}, h2 \in \mathbb{R}^{d_2} \\ W^3 \in \mathbb{R}^{d_2 \times d_{\mathrm{out}}}, b^3 \in \mathbb{R}^{d_{out}} \end{aligned}$$

Here $\mathbf{W^1}, \mathbf{W^2}, \mathbf{W^3}$, and $\mathbf{b^1}$, $\mathbf{b^2}$ and $\mathbf{b^3}$ are matrices and bias vectors for first, second and third linear transforms, respectively. The functions g^1, g^2 and g^3 are activation functions and they are almost always non-linear. With respect to figure 3, the values of d_{in}, d_1, d_2 and d_{out} are 3, 4, 3, and 2, respectively.

3.1.2 Activation Function

The activation functions help the neural network models to approximate any non-linear function. Different activation functions have different advantages. Some popular activation functions are sigmoid, hyperbolic tangent, and rectifiers (Goldberg, 2016):

1. The sigmoid activation function is a S-shaped function which transforms any value into the range between 0 and 1.

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

2. The hyperbolic tangent function is also a S-shaped function, but it transforms any value into the range between -1 and 1.

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

3. The rectifier activation function clips values lesser than 0

$$\mathrm{ReLU}(x) = \max(0, x)$$

The sigmoid activation function is not used in internal layers of neural networks since other functions have been giving better results empirically. The rectifier activation function is commonly used since it performs faster and better than sigmoid and hyperbolic tangent functions.

Instead of an activation function, the function in output layer g^3 can be a transformation function such as softmax to convert values to represent a discrete probability distribution. Each of the converted values will be between 0 and 1 and sum of all of them will be 1.

$$y = \begin{bmatrix} y_1 & y_2 & \dots & y_k \end{bmatrix}$$
$$s_i = \mathrm{softmax}(\mathrm{y_i}) = \frac{e^{y_i}}{(\sum_{j=1}^{k} e^{y_j})}$$

3.1.3 Training Neural Networks

Training is an essential part of learning and like many supervised algorithms, a loss function is used to compute the error for the predicted output against the actual output. The gradient of the errors is calculated with respect to each weight and bias variable by propagating backward using the chain rule of differentiation. The values of the weights and bias are adjusted with respect to the gradient and a learning parameter. Typically a random batch of inputs is selected and a forward pass is carried out which involves multiplying weights, adding bias and applying an activation function to predict outputs. The average loss is computed for that batch and the parameters are adjusted accordingly. This optimization technique is called stochastic gradient descent (Bottou, 2012). A number of extensions exist, such as Nesterov Momentum (Sutskever, Martens, Dahl, & Hinton, 2013) or AdaGrad (Duchi, Hazan, & Singer, 2011). Some loss functions that exist are hinge loss (binary and multiclass), log loss and categorical cross-entropy loss (Goldberg, 2016).

The categorical cross-entropy loss is used when predicted output refers to a probability distribution. This is typically achieved by using a softmax activation function in the output layer. Let $y = y^1, y^2, \dots, y^n$ be representing the target multinomial distribution over the labels $1, 2, \dots, n$ and let $\hat{y} = \hat{y}_1, \hat{y}_2, \dots, \hat{y}_n$ be the network's output which is transformed by a softmax function. The

categorical cross-entropy loss measures the difference between the true label distribution y and the predicted label distribution $\hat{y}$.

$$L_{\text{cross-entropy}}(\hat{y}, y) = -\sum_i y_i \cdot \log(\hat{y}_i)$$

For hard classification, y is a one-hot vector representing the true class. Here t is the correct class assignment. Training will attempt to set the correct class t to 1 which inturn will decrease the other class assignment to 0.

$$L_{\text{cross-entropy(hardclassification)}}(\hat{y}, y) = -\log(\hat{y}_t)$$

Training the neural network for a longer duration, will cause the trained system to perform well only for trained data but not on the test data. This is called as overfitting. This can be minimized by using regularization techniques such as L_2 regularization and dropout (Hinton, Srivastava, Krizhevsky, Sutskever, & Salakhutdinov, 2012). L_2 regularization works by adding a penalty term equal to sum of the squares of all the parameters in the network to the loss function which is being minimized. Dropout, instead, works by randomly ignoring half of the neurons in every layer and corrects the error only using the parameters of another half of neurons. This helps to prevent the network from relying on only specific weights.

3.2 Convolution Neural Networks (CNN)

Feed-forward networks do not take into account the order of input and so they are not good at representing text data. According to it, words w_i and w_j are independent and do not influence each other. Convolution neural networks (LeCun & Bengio, 1995) evolved initially in the vision community where they showed tremendous success in object detection regardless of position in an image. For an image, convolutions are 2-dimensional matrix whereas for text it is implemented by a 1-dimensional vector. These are slid over the input taking into account of their spatial dependencies, resulting in capturing the local important characteristics irrespective of its location in the input. These are then fed into a pooling layer which combines all these characteristics computed from different convolutions either by taking maximum or average of them.

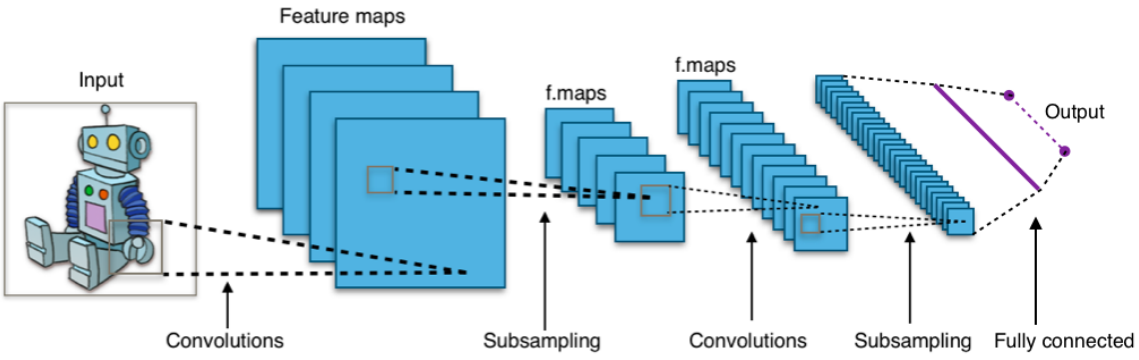

Figure 4: CNN architecture (Alphex34, 2015)

In figure 4, many 2-dimensional convolutional filters are slid over X and Y axes of an input which is a robot image. It outputs a 3-dimensional tensor whose dimensions are influenced by the number and dimensions of filters and length of stride. This output captures many local features of an image and each one is represented by a matrix. This is followed by a max-pooling layer which filters the

features by allowing only the most dominant ones. This combination of convolution and pooling is applied many times and finally passed onto a feed-forward network with softmax at the end.

In the case of text data, convolutional filters are 1-dimensional and they are applied over a k-word sliding window. Convolution neural networks consider ordering to some extent based on the value of k which helps to capture local dependencies. In figure 5, words in a sentence is referred as $x_0, x_1, \ldots, x_n$; the value of k is 2 and so it convolves consecutive words. It is then passed to a neural network module and the outputs $y_0, y_1, \ldots, y_n$ are obtained. Each vector y_i embodies the context of x_i and x_{i+1}. If each of the characters in a word is represented by a vector then a 2-dimensional convolutional filter can be used and this will be same as convolutions applied on an image.

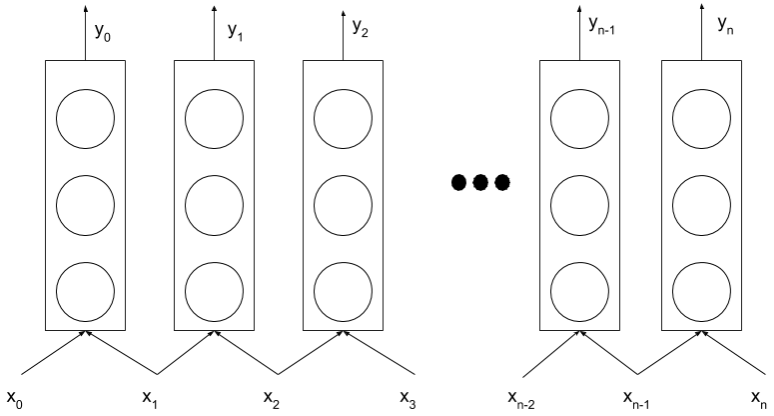

Figure 5: CNN architecture on words

3.3 Recurrent Neural Networks (RNN)

For text data, the input is sequential and of unknown length, where the ordering of words is important. Techniques such as continuous bag of words (Mikolov, Chen, Corrado, & Dean, 2013b) can be used with feed-forward networks to convert sequential input into fixed-length vectors but it will discard the order of words. Convolutional neural networks (CNN) (Y Bengio, 1997) are good at capturing the local characteristics of data irrespective of its position. In this, a nonlinear function is applied to every k-word sliding window and captures the important characteristics of the word in that window. All the important characteristics from each window are combined by either taking maximum or average value from each window. This captures the important characteristics of a sentence irrespective of its location. However, because of the nature of CNNs, they fail to recognize patterns that are far apart in the sequence.

Recurrent neural networks (RNN) accept sequential inputs and are often able to extract patterns over long distances (Elman, 1990). A RNN takes input as an ordered list of input vectors $\mathbf{x_1}, \ldots, \mathbf{x_n}$ with initial state vector $\mathbf{h_0}$ and returns an ordered list of state vectors $\mathbf{h_1}, \ldots, \mathbf{h_n}$ as well as an ordered list of output vectors $\mathbf{o_1}, \ldots, \mathbf{o_n}$. At time step t, a RNN takes as input a state vector $\mathbf{h_{t-1}}$, an input vector $\mathbf{x_t}$ and outputs a new state vector $\mathbf{h_t}$ as shown in figure 6. The outputted state vector is used as input state vector at the next time step. The same weights for input, state, and output vectors are used in each time step.

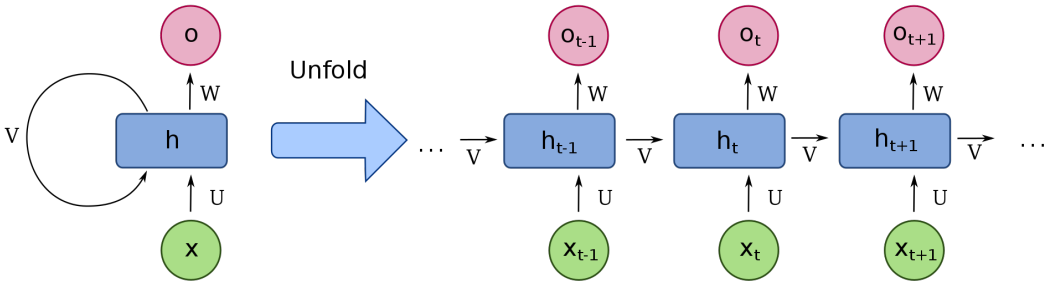

Figure 6: A basic example of RNN architecture (Deloche, 2017b).

$$\text{RNN}(h_0, x_{1:n}) = h_{1:n}, o_{1:n}$$
$$h_i = g^1(h_{i-1}V + x_iU + b^1)$$
$$o_i = g^2(h_iW + b^2)$$

$$x_i \in \mathbb{R}^{d_x}, U \in \mathbb{R}^{d_x \times d_h}$$
$$h_i \in \mathbb{R}^{d_h}, V \in \mathbb{R}^{d_h \times d_h}, b^1 \in \mathbb{R}^{d_h}$$
$$o_i \in \mathbb{R}^{d_o}, W \in \mathbb{R}^{d_h \times d_o}, b^2 \in \mathbb{R}^{d_o}$$

Here the functions g^1 and g^2 are non-linear activation functions; $\mathbf{W}$, $\mathbf{V}$, and $\mathbf{U}$ are weight matrices and $\mathbf{b^1}$, $\mathbf{b^2}$ are bias vectors.

To train a RNN, the network is unrolled for a given input sequence and the loss function is used to compute the gradient of error with respect to parameters involved in every time step by propagating backward through time. After that, the parameters are adjusted to reduce the error in prediction (Werbos, 1990).

3.4 Neural networks with memory

While training RNNs, a common problem that especially occurs with long input sentences is that the error gradients might vanish (become too close to zero) or explode (become too large) which results in numerical instability during the backpropagation step. The gradient explosion can be handled by clipping a given gradient when it goes beyond the threshold. Long short-term memory (LSTM) networks (Hochreiter & Schmidhuber, 1997) solve the vanishing gradient problem by introducing memory cells that are controlled by gating components. These gating components decide at each time step, what parts of the hidden state should be forgotten and what parts of new input should be included in the memory cells. These memory cells are involved in the computation of hidden states, which in turn are used to compute the output states. This technique has been shown to provide good results in practice, in capturing the dependency between words even though separated by a long distance.

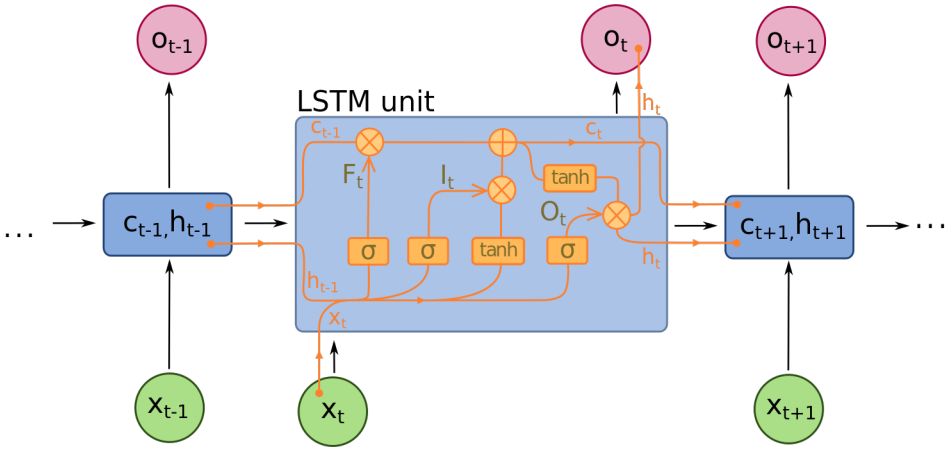

Figure 7: LSTM architecture (Deloche, 2017a).

$$
\begin{aligned}
s_j = R_{LSTM}(s_{j-1}, x_j) = [c_j; h_j] \\
c_j = c_{j-1} \odot f + g \odot i \\
h_j = tanh(c_j) \odot o \\
i = \sigma(x_j W^{xi} + h_{j-1} W^{hi}) \\
f = \sigma(x_j W^{xf} + h_{j-1} W^{hf}) \\
o = \sigma(x_j W^{xo} + h_{j-1} W^{ho}) \\
g = \sigma(x_j W^{xg} + h_{j-1} W^{hg}) \\
y_j = O_{LSTM}(s_j) = h_j \\
s_j \in \mathbb{R}^{2.d_h}, x_i \in \mathbb{R}^{d_x}, c_j, h_j, i, f, o, g \in \mathbb{R}^d_h, \\
W^{xo} \in \mathbb{R}^{d_x * d_h}, W^{ho} \in \mathbb{R}^{d_h * d_h}
\end{aligned}
$$

Here the symbol $\odot$ denotes component-wise product. The LSTM network uses 3 gating components such as input gate (i), forgot gate (f), and output gate (o) respectively. They contain a sigmoid function to convert the values between 0 and 1 which is used to decide how much to keep. The memory cell state c_j is obtained by controlling the previous memory cell state with forgot gate and the new input state (g) with input gate. The obtained memory cell state is controlled by the output gate to get the hidden state h_j.

A similar type of architecture with fewer elements is also effective in handling long-range dependencies is Gated Recurrent Unit (GRU) (Cho et al., 2014).

3.5 Bidirectional RNN

A RNN computes the state of current word $\mathbf{x_i}$ only based on the words in the past, i.e. $\mathbf{x_1}, \ldots, \mathbf{x_{i-1}}$. However, the following words $\mathbf{x_{i+1}}, \ldots, \mathbf{x_n}$ will also be useful in computing the hidden state regarding the current word. The bidirectional RNN (biRNN) (Schuster & Paliwal, 1997) solves the problem by having two different RNNs. The first RNN is fed with the input sequence $\mathbf{x_1}, \ldots, \mathbf{x_n}$ and the second RNN is fed with input sequence in reverse. The hidden state representation $\mathbf{h_i}$ is then composed of both the forward and backward states. Each state representation consists of the token information along with sentential context from both directions which has shown better results than classical uni-directional RNNs in practice.

In the figure 8, the inputs $x_0, x_1, \ldots, x_n$ are passed through two RNNs; one process it from left to right with hidden state s_i and the other from right to left with hidden state $s_i^{'}$. The outputs of each of them are concatenated with other at every time step to form $y_0, y_1, \ldots, y_n$.

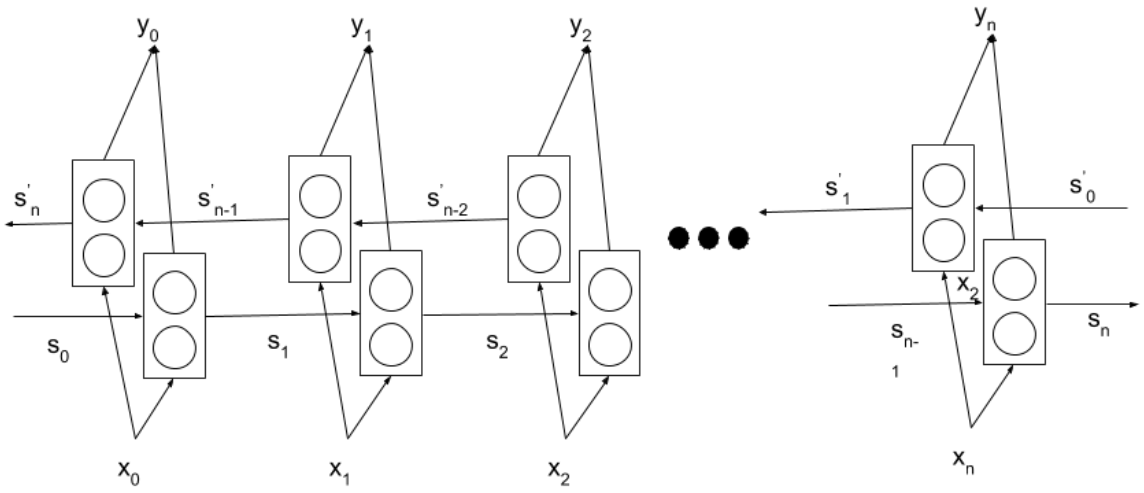

Figure 8: Bidirectional RNN

3.6 Data Representation

The representation of data is a very important factor in improving the performance of any machine learning application. In the past, inputs are differentiated using sparse vectors called one-hot vectors. If there are n inputs then there will be n one-hot vectors and each of its dimensions will be n. The input w_i will be represented by a vector whose elements will be 0 except that the value of i^{th} element to be 1. Each input is orthogonal to all others and the larger the number of inputs the higher the dimensions of vectors. It also increases the sparsity of representation which is a disadvantage.

3.6.1 Word Embeddings

Instead of representing each word by a vector of dimension n that is equal to the total number of words in a training dataset, a dense vector of dimension d where $d << n$ is used. The difference here is that all the elements of a vector can have any value whereas with one-hot vectors it is mostly 0 except one element with value 1. Unlike one-hot vectors, these dense vectors will be initialized randomly at the beginning and will be trained according to the application needs. These dense vectors are called as word embeddings. The dimensionality of these word embeddings typically ranges from 50 to a few hundred.

There are two advantages of using word embeddings and they are listed here:

1. Computation becomes efficient. It consumes less space and so the parameters that deal with will also get reduced. With the ability to have any values for each element, the representation becomes efficient

2. Model training will make the dense vectors to have similar values for similar features. This will make the training efficient

Many NLP applications were benefitted using word embeddings and they are initially made popular by few works. (Yoshua Bengio, Ducharme, Vincent, & Janvin, 2003), (Schwenk, Déchelotte, & Gauvain, 2006).

There are many unsupervised learning approaches to compute word embeddings and the most common ones are word2vec (Mikolov, Chen, Corrado, & Dean, 2013a), and Glove (Pennington, Socher, & Manning, 2014). These pre-trained word embeddings can be used directly or word embeddings with random initialization can be learned while performing task-specific training.

Mikolov et al. analyzed these dense vectors and figured out that they possess good syntactic and semantic word relationships (Mikolov, Yih, & Zweig, 2013). For instance the vector calculation $v_{madrid} - v_{spain} + v_{france}$ is closer to v_{paris}. This shows that it captures the capital to city relationship effectively. Similarly, man to woman, company to CEO, city to zip code, and comparative to superlative relations are captured very well. This shows that dense vectors inherently embody relationship attributes. They are much more than mere numbers. The figure 9 is an illustration of embeddings representing words such as man, woman, etc. The relationship between man and woman, uncle and aunt, and king and queen show similarity in their vector calculations.

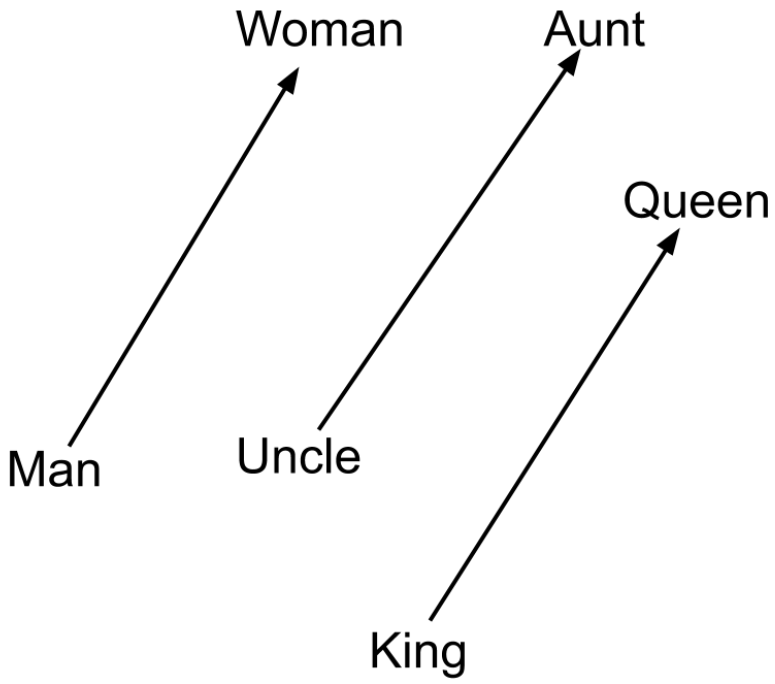

Figure 9: Word embeddings charcteristics

The word embeddings can be built over character embeddings. The representation of words can be built using a convolutional network over the words (Dos Santos & Gatti, 2014). Also, the word vectors can be computed using final states of bi-directional RNN over the characters in a word (Ling et al., 2015). Character embeddings compose syntactic relations as the character patterns within words are strongly related to their syntactic function (Goldberg, 2016). Some advantages of character embeddings are the models are smaller in size since only one vector for each character and also it is possible to compute a vector for any word irrespective of its presence in training dataset.

3.6.2 Embeddings from Language Models (ELMo)

In linguistics, a word sense is one of the meanings of a word. Words have different meanings depending on the context it is being used. The disadvantage of word embeddings is that it uses only one vector to represent all word senses of a word. Say e.g., consider the sentences below:

- We went to see the $play$ Romeo and Juliet
- The children went out to $play$ in the park

Here the word $play$ means different in each of the sentences. But with word embeddings, the same vector will be used to represent for two cases.

Peter et al. (Peters et al., 2018) created a deep contextualized word representation which models:

- Complex characteristics of the word use
- how these uses vary across many context

The representation is so powerful that it significantly improves six challenging NLP applications such as question answering, textual entailment, and sentiment analysis. The figure 10 shows the architecture of a model that produces ELMo based representations. This involves training for a longer period on a very large corpus.

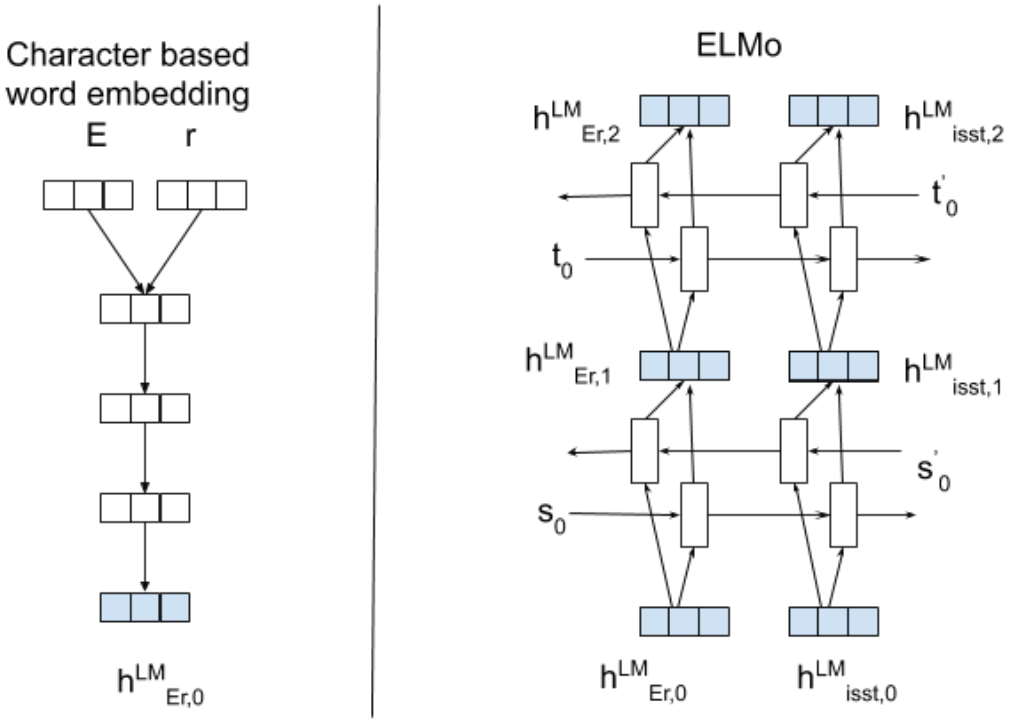

Figure 10: ELMo

Character embeddings with convolution filters or bidirectional RNNs are used to represent words as shown on the left side of the above figure. These representations are passed through a two-layered bidirectional RNNs and the outputs for a word w_i are $h^{LM}_{w_i,1}$ and $h^{LM}_{w_i,2}$. Since it involves bidirectional RNN, the vectors for word w_i will be dependent on the previous and next words. The training for producing ELMo embeddings is based on an unsupervised language modeling task. In this, for a

sentence, when w_i is passed as input, the system predicts the next word w_{i+1}^p given the previous words. In the next state, the actual word w_{i+1} is passed as input and now the system predicts the next word. In the case of ELMo, these predictions happen at the final layer which is at $h_{w_i,2}^{LM}$. This models the behavior of conditional probability given by $P(w_i|w_{i-1}, w_{i-2}, ...w_1)$.

The embeddings for words are obtained by considering all three layers. For each task, for each word, the system is trained to extract embeddings from all three layers using a weighted average of them. The following equation shows how to extract embeddings:

$$ELMo_k^{task} = \gamma^{task} \sum_{j=0}^{2} s_j^{task} h_{k,j}^{LM}$$

In this γ^{task} is the scalar that scales all vectors and s_j^{task} is the scalar weight for each layer representation with $s_j^{task} \epsilon [0, 1]$ and $\sum_{j=0}^{2} s_j^{task} = 1$.

The following table captures the difference between the outputs of GloVe and ELMo for the word 'play'. The nearest neighbor is calculated using Euclidean distance and for Glove they are playing, game, games, etc which is quite general. Whereas for ELMo, it is quite specific and very context-rich.

	Source	Nearest Neighbors
GloVe	play	playing, game, games, played, players, plays, player, Play, football, multiplayer
biLM	Chico Ruiz made a spectacular play on Alusik 's grounder {...}	Kieffer , the only junior in the group , was commended for his ability to hit in the clutch , as well as his all-round excellent play .
	Olivia De Havilland signed to do a Broadway play for Garson {...}	{...} they were actors who had been handed fat roles in a successful play , and had talent enough to fill the roles competently , with nice understatement .

Figure 11: ELMo analysis (May, 2019)

3.7 Attention Networks

Initially, machine translation systems were implemented using only sequence to sequence architectures which predominantly uses final states of RNN networks (Sutskever, Vinyals, & Le, 2014).

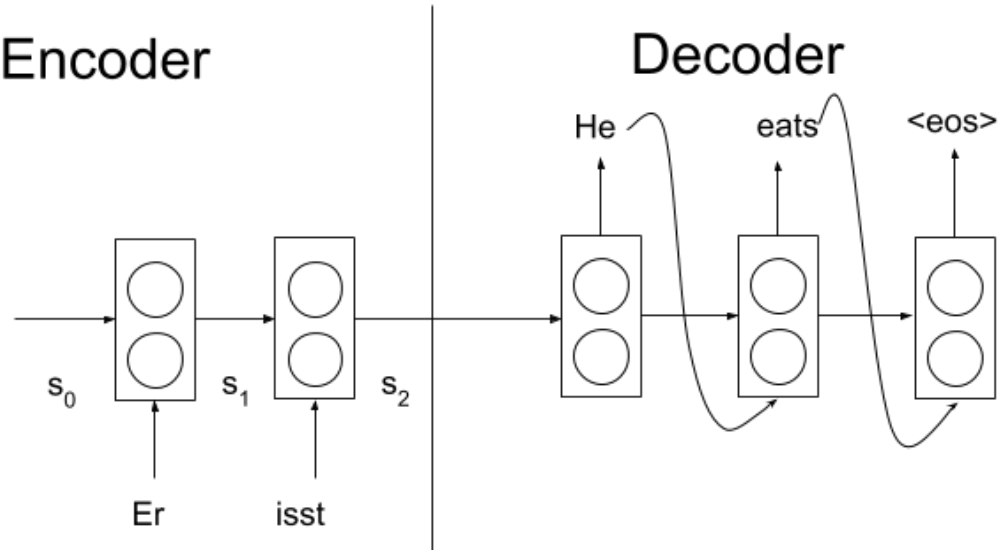

Figure 12: RNN network for machine translation (May, 2019)

In figure 12, the encoder of the machine translation system uses a RNN module that takes in a sentence 'Er isst' and outputs final hidden state s_2. The decoder part is nothing but a language modeling task conditioned on s_2. It uses RNN to predict the English sentence and outputs as 'He eats'.

For longer sentences, machine translation systems were giving poor results even with the use of LSTM or GRU models. These systems were attempting to convert sentences of any length to a fixed vector size. This has limitations to remember when the sentences get longer.

Attention models were introduced in machine translation systems and they achieved better results (Bahdanau, Cho, & Bengio, 2014). The figure 13 is an illustration of their work. In this method, a bidirectional RNN module is used to model the sentences. All the state outputs $a^1, a^2 \dots a^n$ are passed to an attention module to output $context^{<t>}$. This context is used along with the previous hidden state h_{t-1} and their predicted output e.g. 'He' to produce the next hidden state h_t and output e.g. 'eats'. In this, instead of converting all the words of a sentence to a single fixed output vector, depending upon the length of a sentence, their respective state output is used effectively.

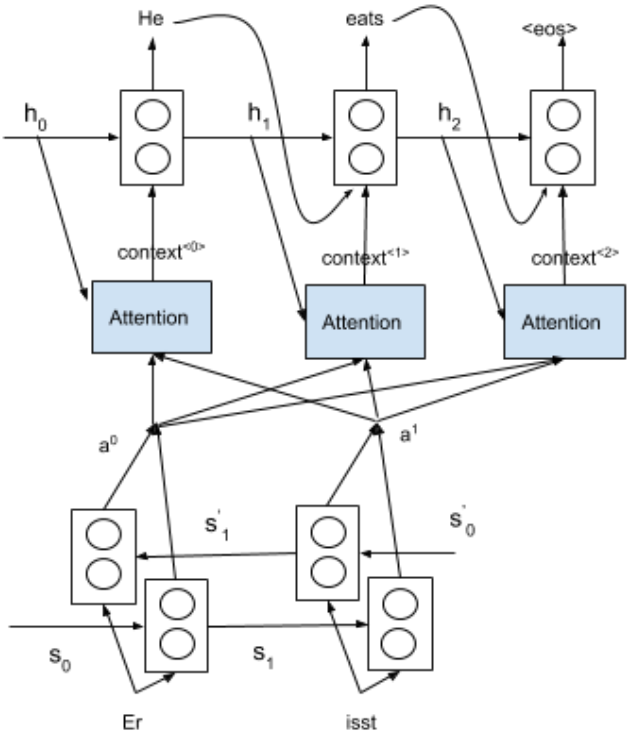

Figure 13: Attention network for machine translation

In this method, the attention module uses all the state outputs of the encoder to produce a context vector. This is achieved by computing weights which quantify how much each of the words is attended to or influences the translation at every time step t. The figure 14 reveals how the training of attention weights is achieved. It uses a simple feed-forward neural network whose input involves the decoder previous hidden state $h_{t-1}$and a respective encoder state output $a_{t'}$ to compute the attention weight $\alpha^{<t,t'>}$. This is then used to produce the next decode hidden state h_t.

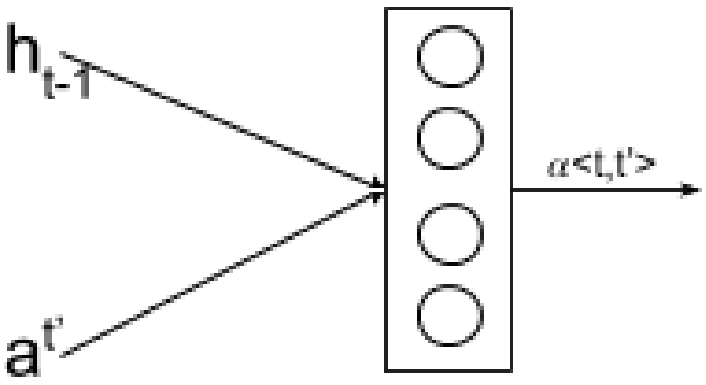

Figure 14: Attention module in machine translation

After computing all the attention weights for all state outputs of an encoder for a time step t, they are passed through softmax to scale their influence between 0 and 1. The resultant context vector is

computed by the weighted average of all state outputs of the encoder.

$$context^{<t>} = \sum_{t'} \alpha^{<t,t'>} a^{t'}$$

The attention weights also give a good idea of what words in a sentence influences what parts of translation output. This throws light in analyzing effectively how the system is performing for different inputs. In the figure 15, it is revealed that the words 'Er' and 'issue' influences 'He' and 'eats' separately and the translation is completed by outputting '<eos>' which is influenced by 'isst'.

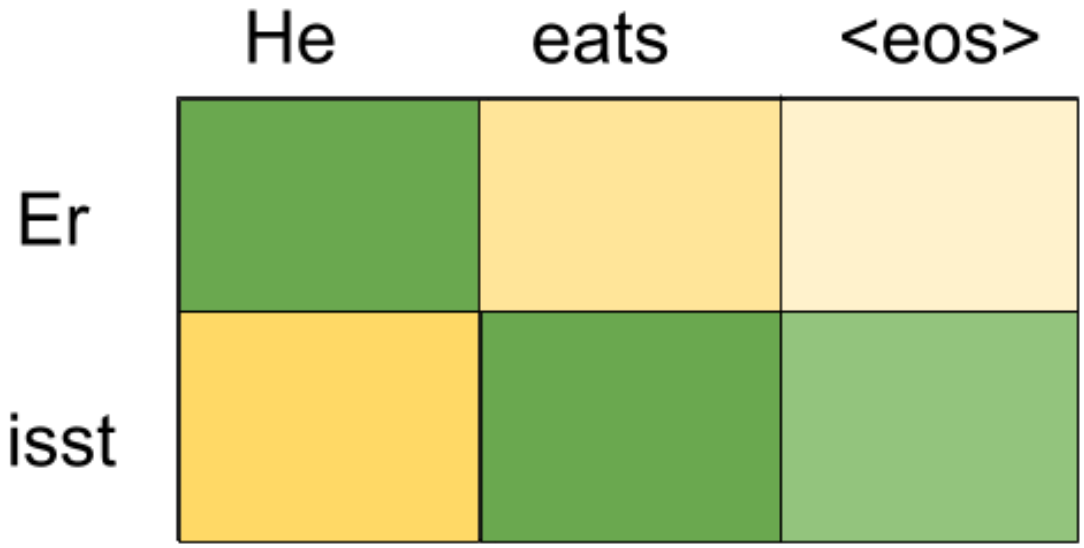

Figure 15: Attention matrix

Attention mechanism with RNN gives state of the art results in many NLP applications such as Natural Language Inference (Parikh, Täckström, Das, & Uszkoreit, 2016), Machine Translation (Bahdanau et al., 2014) and Document classification (Yang et al., 2016).

3.8 Self-attention Networks

Attention modules showed great capabilities in effectively using the encoder outputs to achieve decoder functionalities efficiently. It uses trainable attention weights that helps the system to adapt its focus on parts of encoder depending upon the intermediate outputs of decoder.

The Transformer model architecture (Vaswani et al., 2017) showed that attention modules capture the context of input effectively and when used to represent input, achieves better results in many NLP application systems such as machine translation, and English constituency parsing.

Instead of using bidirectional RNNs to encode input sentences they used attention modules on input sentences to compute influential weights against the same. This encodes the sentence very effectively by outputting vectors for each word which has the sentential context as shown in figure 16.

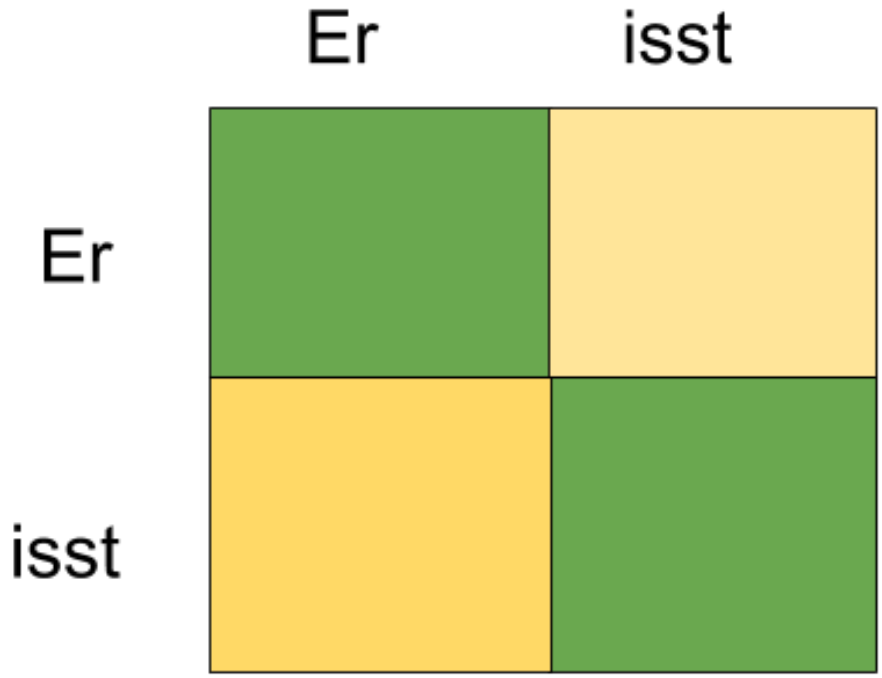

Figure 16: Self-attention matrix

The figure 17 shows how each word w_i is transformed into 3 vectors such as key (k_i), value (v_i) and query (q_i) with lesser dimensions than w_i. The attention of a word i to a word j in a sentence is then calculated as $p(i \rightarrow j) \propto exp(\frac{q_i^{(c)}.k_j^{(c)}}{sqrt(d_k)})$. The weighted average of all values to form a average vector $v_{i\_avg} = \sum_k p(i \rightarrow k)v_k$ represents the new value vector for word w_i which includes all the attention that it has on all the words in the sentence.

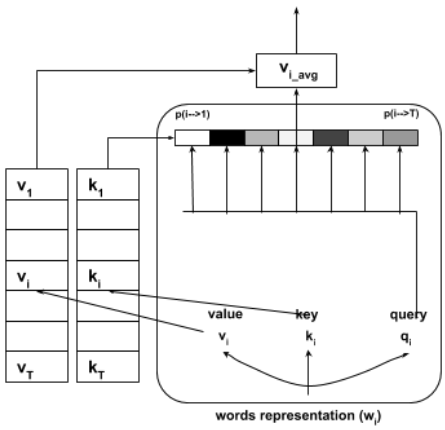

Figure 17: Self-attention module

3.9 CKY Parsing

Syntax structure can be modeled by Context-Free Grammars (CFG). Parsing attempts to extract the syntactic structure, in the form of a tree from a sentence, using the production rules specified in CFG. A naive top-down search will take one of the starting non-terminals and proceeds to explore the non-terminals, all the way up to terminals and try to match the words. It explores many options that never connect to the actual sentence. Whereas a naive bottom-up search will start with words and try to match the non-terminal, all the way up to the starting terminal. It explores many options that can never lead to a full parse. The number of unsuccessful searches depends on how much the grammar branches in each direction.

Cocke-Kasami-Younger (CKY) (Younger, 1967) is a bottom-up parsing and dynamic programming method that requires first normalizing the grammar.

Two key issues that CKY parsing are dealt with are computational complexity and ambiguity. The computational complexity is handled by using a chart that stores the previous recognition of production rules for word combination or sub-rule combinations and re-using it appropriately.

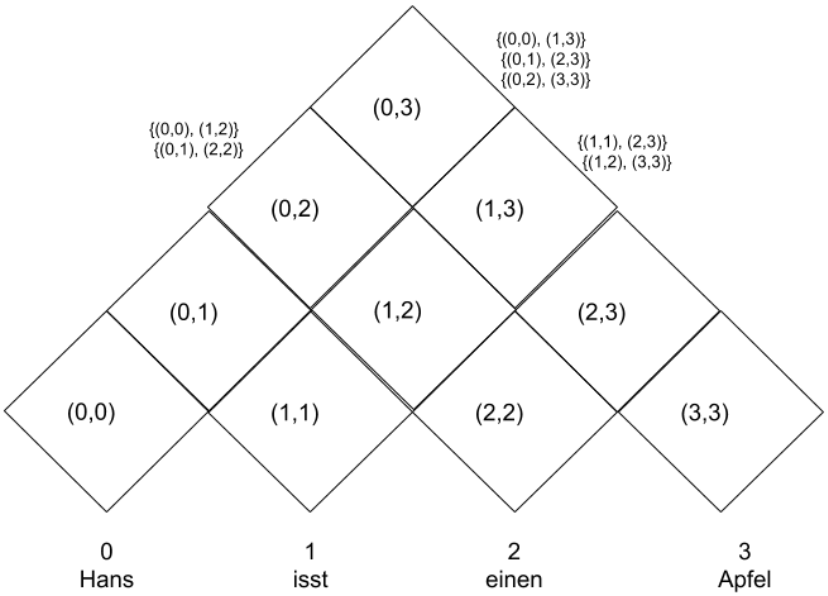

Figure 18: CKY chart

The ambiguity is dealt with by using Probabilistic Context-Free Grammar (PCFG). The probability values assigned to each production rule are used while parsing when the production rule replaces word combinations or sub-rule combinations and the importance of usage is computed. This importance is compared against other relevant production rules that could be applied and the best score is chosen.

The algorithm of CKY is explained briefly as follows:

1. Convert the given grammar into Chomsky Normal Form (CNF). This involves allowing only two kinds of right-hand sides in production rules such as either two non-terminals or one terminal. Any CFG can be written in CNF form

2. Create a chart with the number of layers equivalent to the number of words in a sentence as shown in figure 18. Each layer refers to the span of a sentence and so all possible combinations in a sentence will be explored

3. In the first layer, each word is dealt with. Replace the words with appropriate non-terminal(s) with the probability score

4. Look for unaries that can re-write non-terminals and if found then append those to appropriate cells in the chart. While adding recompute the new probability score by multiplying the appropriate production rule components (both left and right). This is continued until no possible replacement is available

5. Look for the duplication of each non-terminal in each cell. If found then the one with the highest score is chosen and the rest are removed

6. When a layer with larger span is involved, use the results computed from a lower layer which is referring to smaller span. Consider all the possible cell pair combinations and for each cell pair, consider all the combinations of non-terminals from each cell. Say for e.g. for a span (0,2), the cell pairs {(0,0), (1,2)} and {(0,1), (2,2)} will be taken into account as shown in the figure 18. Follow step 3 for the two non-terminals instead of words and also unary replacement steps. In each case compute the new probability score building from the previous cells

7. Keep on doing it until the span limit is covered. Take the non-terminal S, which is the starting terminal, with the highest score. If none found then the sentence cannot be parsed for a given grammar

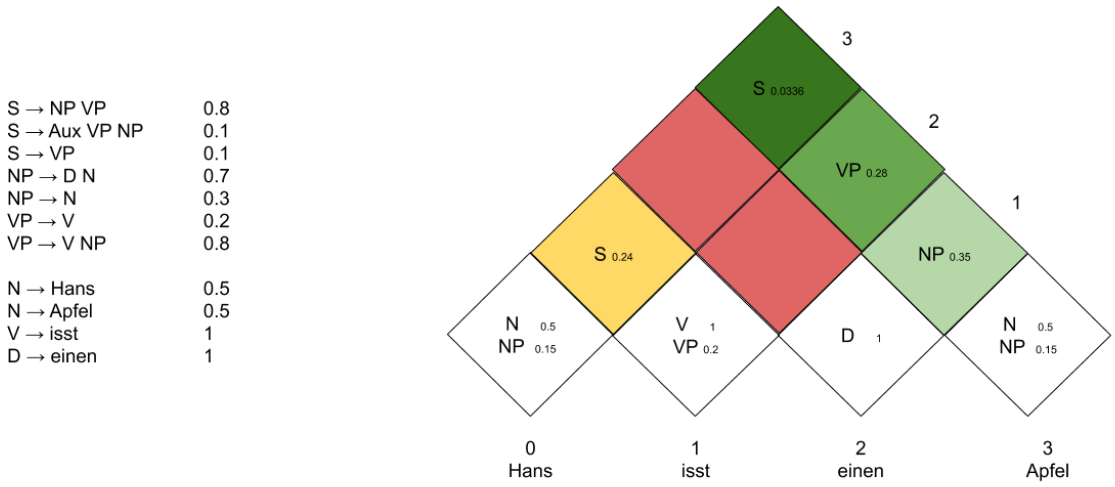

Figure 19: CKY parsing

In figure 19, the sentence 'Hans isst einen Apfel' is parsed against PCFG grammar on the left side using CKY parsing and the tree produced is shown in green color in the chart. 'Hans' can be replaced with non-terminal N as in production rule N –> Hans and unary replacement NP –> N. The probability score 0.15 is computed by multiplying 0.3 and 0.5 that are taken from production rule and non-terminal N respectively. Certain combinations are not at all possible and they are referred to as red color in the chart.

4 Approach

In this thesis work, deep learning techniques are used to develop constituent parser for the German language. This section explains in detail the architecture used and the different components included in it.

4.1 Research Questions

Through this work, the research questions that would be addressed are:

1. How well the self-attention based modules of neural networks are effective in capturing constituency grammar of the German language?
2. How efficient the model can be improved by using only the tree dataset? This means no other external data directly or indirectly will be utilized.
3. How efficient the model can be improved using Transfer learning which involves pre-trained deep learning systems using external data such as German Wikipedia?

4.2 Model

The models are highly inspired by the works of Kitaev et al. (2019), which involves an encoder-decoder architecture as shown in figure 20:

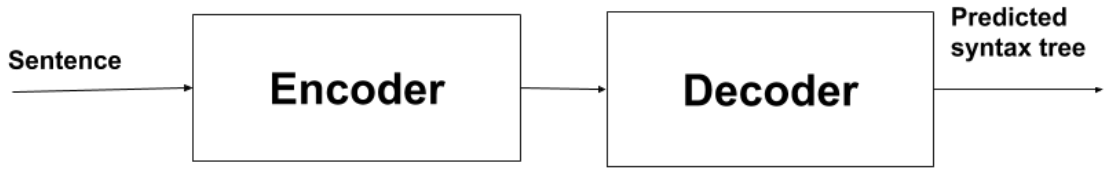

Figure 20: Broad Architecture

The input to this system is a german sentence and the output is a predicted syntax tree of that particular sentence. The encoder takes in input sentence and does processing to convert it into a convenient, rich and versatile representation. This representation embodies information for each word with sub-word patterns as well as the context of the whole sentence. The encoded output is then fed into a decoder that outputs a syntax tree. The building of syntax tree is seen as a continuous iteration of the merging of certain subsequent words or merges that already happened in a hierarchy. The decoder method involves steps to figure out what combination of merges is the best to build a good syntax tree.

The training of a deep learning system involves a loss function to compare the prediction against golden output to allow fine-tuning of different components of the model using backward propagation of gradients of error. In this context, the predicted best possible syntax tree is compared against the golden tree and the parameters involved in the encoder section is updated accordingly to yield the best syntax tree next time.

There are many ways by which the encoder and decoder components can be modeled. But in this approach, the encoder and decoder components are fixed and only the representation of sentences is varied to a certain degree. The encoder module is inspired by a state of art transformer neural network which predominantly uses self-attention modules, whereas the decoder module uses the CKY

approach to build the syntax tree. Sentences are represented using character embeddings and different word embeddings. In addition to words, position information of words is also used separately. The system does not use any other pre-processed information such as Parts Of Speech (POS) obtained by an external tagger system as in old parsing methods.

4.2.1 Encoder

The purpose of the encoder, in this context, is to represent a sentence into a rich format that encompasses the sub-word patterns as well as the overall context of a sentence. This is achieved through multi-layered, multi-head attention modules backed up with feed-forward networks as shown in the figure 21

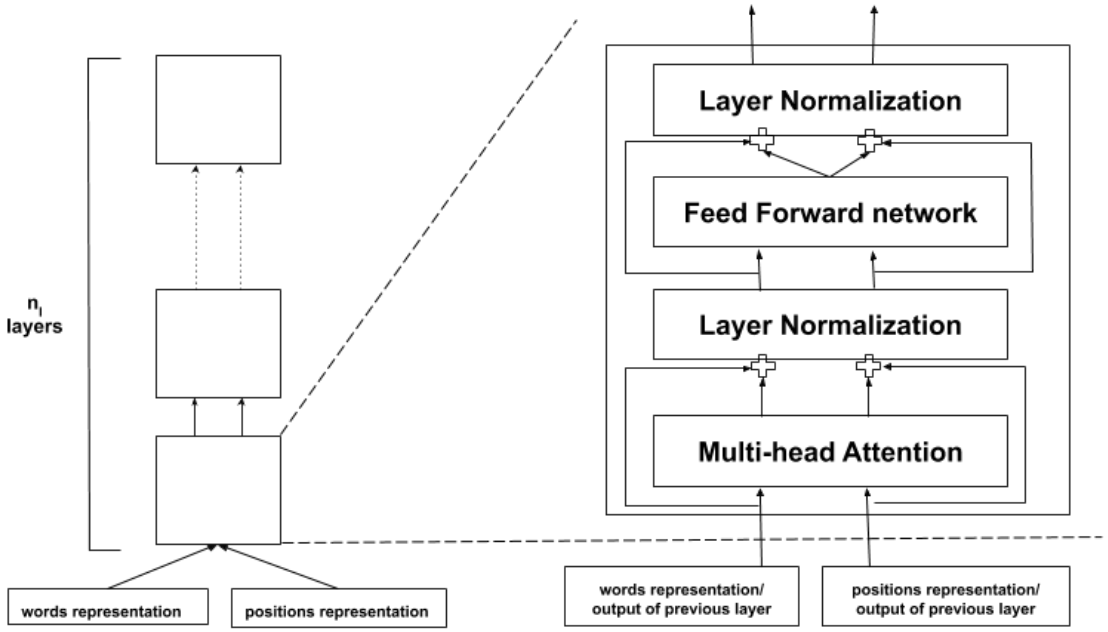

Figure 21: Encoder module

The input to this system is passed through n_l identical layers made up of multi-head self-attention mechanism modules sequentially. The output of the first layer is passed as input to the second layer. At each layer, the representation of a sentence is expected to improve gradually. Multi-head attention uses n_h single heads, whereas each one allows every word in a sentence to gather information from one word in the same sentence. This brings out a matrix of information regarding what words attend to or obtain information from what other words in a sentence. This is applied not only to words but also to positions separately. Kitaev et al. (2019) shown in their work that separating words and positions for building constituent parser improved the results by 0.5 % F1 score. And also that, position attention contributes more than content attention by 18 % F1 score. Hence the position attention is used separately. In their setup, value 8 for both n_l and n_h parameters gave better results.

The output of Multi-head attention module is then passed to Layer Normalization by mixing with a residual component of the input. The normalization of intermediate results along with objective function will help the gradient-based optimizer to arrive at the best values quickly. A feed-forward network at this juncture helps to look for patterns among the multiple attention representation of words in a sentence and also to map the intermediate result dimension space to be compatible with inputs and outputs of each layer.

The dimensionality of representation, inputs, and outputs of each layer are the same. A total of 1024 neurons are used to represent and it is split half and used by word and position related functions respectively.

The figure 22 shows the outline of what happens inside a single attention head:

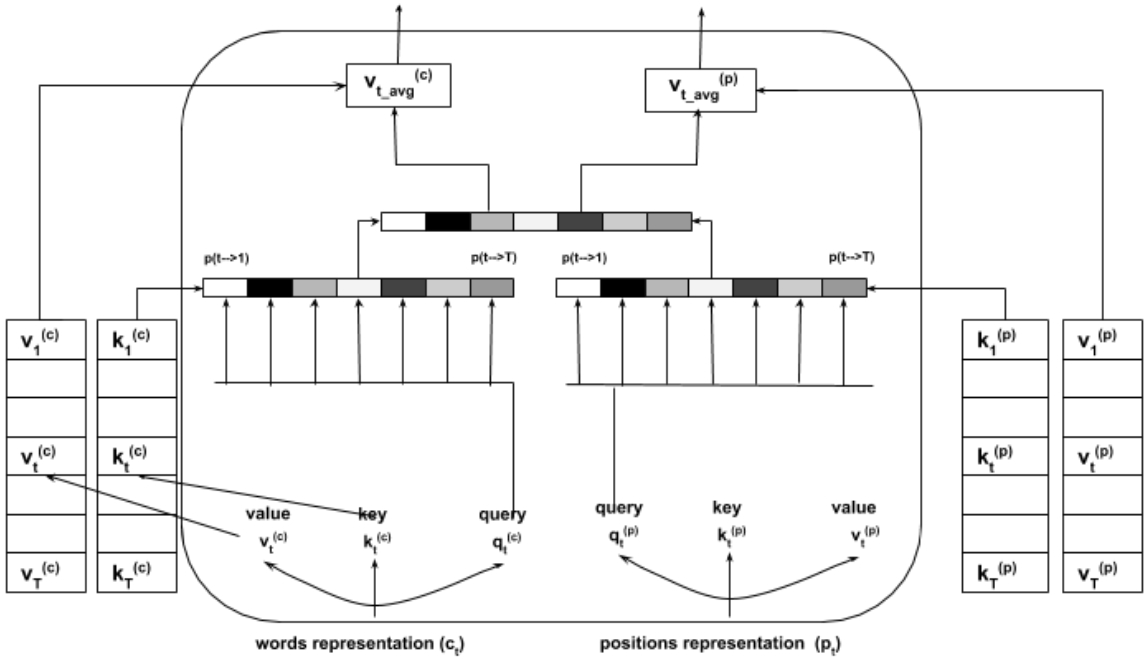

Figure 22: Single attention head

Each word or position representation is transformed into 3 vectors such as key (k_t), value (v_t) and query (q_t) with lesser dimensions (d_k). This is done by using separate trainable parameters for word and position representation as well as for the above three entities. For example, the parameters for key transformation for both word and position representation are W_K^c and W_K^p respectively.

The attention of a word i to a word j in a sentence is then calculated as $p(i \rightarrow j) \propto exp(\frac{q_i^{(c)}.k_j^{(c)}}{sqrt(d_k)})$. In words, it is essentially a normalized dot product of respective query and key vectors. The weighted average of all values to form a average vector $\hat{v}_i = \sum_k p(i \rightarrow k)v_k$ represents the new value vector for word i which includes all the attention that it has on all the words in the sentence.

The following set of equations reveal overall functionality of single attention head in mathematical terms. The query, key and value vectors are computed by respective weight parameters against the word representation. The output of this is a vector for each and every word. A matrix containing all the vectors concatenated represents the overall sentence and they are shown as Q_p, K_p, and V_p respectively. The softmax function models the probability distribution of the dot product of query and key matrix. The dot product with value matrix computes the weighted average and it is transformed

by $W_O^{(c)}$ matrix to align with other output dimensions.

$$q_t^{(c)} = W_Q^{(c)} c_t, k_t^{(c)} = W_K^{(c)} c_t, v_t^{(c)} = W_V^{(c)} c_t$$
$$SingleHead(C) = \left[softmax\left(\frac{Q_c K_c^T}{\sqrt{d_k}} \right) V_c \right] W_O^{(c)}$$
$$where\ Q_c = CW_Q^{(c)}; K_c = CW_K^{(c)}; V_c = CW_V^{(c)}$$

Similarly for position representation, the following equations convey single head functionality. Different set of trainable parameters are used to get the final output.

$$SingleHead(P) = \left[softmax\left(\frac{Q_p K_p^T}{\sqrt{d_k}} \right) V_p \right] W_O^{(p)}$$
$$where\ Q_p = PW_Q^{(p)}; K_p = PW_K^{(p)}; V_p = PW_V^{(p)}$$

The output of single attention head is concatenated as shown in the figure 23

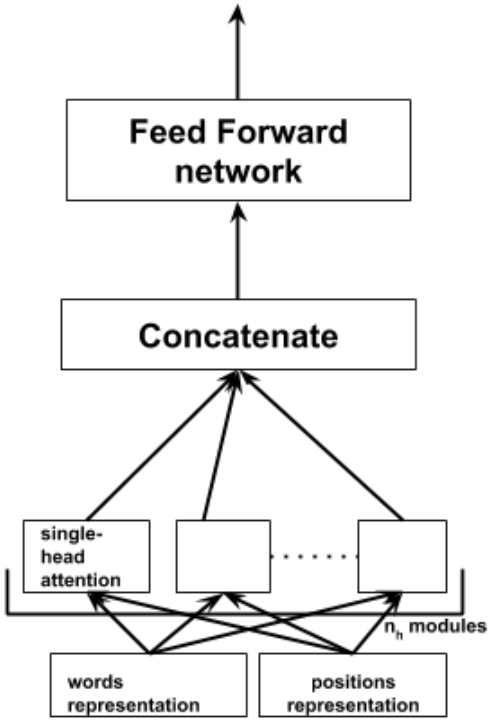

Figure 23: Mult-head attention module

The concatenated output fed through feed-forward to derive the patterns across multiple attentions in a sentence. These are then fed to the Layer Normalization block as shown in figure 21.

4.2.2 Decoder

The decoder of this parser is inspired by the works of Stern et al. (Stern et al., 2017) and Gaddy et al. (Gaddy et al., 2018). The output of encoder is a matrix wherein each row vector of size 1024 neurons represents a word in a sentence. The quantification of how good a combination i and j is considered as

a constituent with label l is computed by scoring modules. The combination of two entities i and j is represented by a vector $v = [\overrightarrow{y}_j - \overrightarrow{y}_i; \overleftarrow{y}_{j+1} - \overleftarrow{y}_{i+1}]$. The vector of size 1024 for a word or entity is divided into two vectors of equal size to be used as a representative for the left or right part of a constituent. The computed vector v for entities i and j is transformed non-linearly with normalization into a space of size equal to the number of labels. The label information is obtained by going through all the training samples and used to predict the label as well as the scores associated with it.

$$s(i, j, l) = M_2 relu(LayerNorm(M_1 v + c_1)) + c_2$$

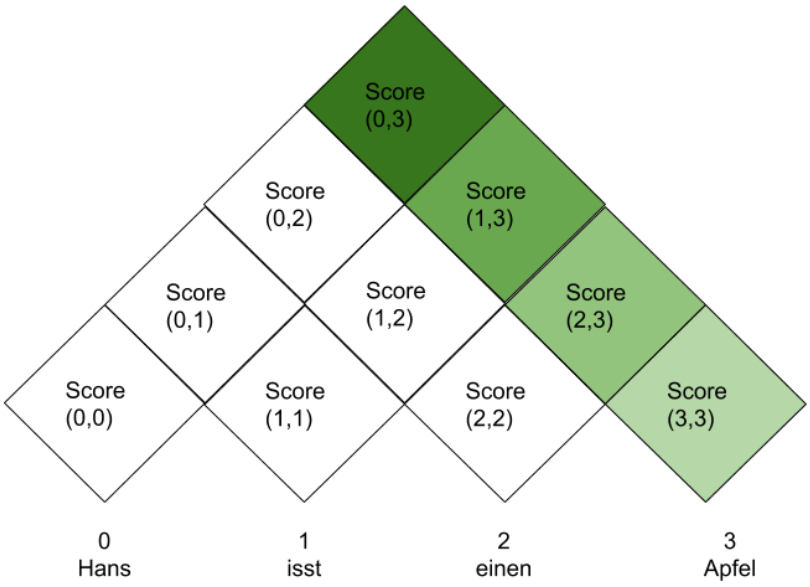

Figure 24: CKY parsing with scores

The decoder uses CKY parsing to compute scores for all the combinations and also computes the best overall tree $\hat{T}$ as the sum of all best subtrees. There are no defined PCFG grammar and its components associated with it. Every combination is a production rule, every label in a training set is a non-terminal, every word is a terminal and all non-terminals are starting terminals. The score for a combination is considered as probability value of a production rule. The best score for a combination $s_{best}(i, j)$ is obtained by the sum of its label score and best split that can divide the combination recursively. For consecutive entities, $s_{best}(i, i + 1)$ is based only on the label score.

$$s(T) = \sum_{(i,j,l)\epsilon T} s(i, j, l)$$

$$\hat{T} = \underset{T}{\operatorname{argmax}}\ s(T)$$

$$s_{best}(i, i + 1) = \max_{l}\ s(i, i + 1, l)$$

$$s_{best}(i, j) = \max_{l}\ s(i, j, l) + \max_{k}\ [s_{best}(i, k) + s_{best}(k, j)]$$

4.2.3 Training

A max-margin based objective function is used to ensure that training of constituent parser results in the right parser tree.

$$max\left(0, \underset{T \neq T^*}{max}\left[s(T) + \Delta(T, T^*)\right] - s(T^*)\right)$$

Here T^* is the golden tree and s(T) refers to all the possible trees except the golden tree. Δ used is a Hamming loss computed for golden and predicted tree. In this system, for every comparison of the subtree with label failing between golden and predicted tree, a value 1 is summed.

With this training approach, the score of a golden tree will be maintained higher than any other tree by a margin of Δ.

$$s(T^*) \geq s(T) + \Delta(T, T^*)$$

Here, the parsing ability of the system is not only learned but also the PCFG grammar of the German language. While trying to maintain the score for a golden tree to be high, the scores for combinations are fine-tuned which is analogous to the probability values of that production rule.

4.3 Experiments

The experiments are different for each pursuit of research questions. But the findings of one research are used in the subsequent researches.

4.3.1 Research Question 1

The first research question address how effectively the self-attention module can be utilized to maximize the optimal building of constituent parser. The following points cover the idea of experiments that will be conducted for this scenario:

1. The first parameter that controls the self-attention module is the number of self-attention heads (n_h). This helps the system to let the words to gather information from multiple locations in the same sentence.

2. The next parameter which influences the output is the number of identical layers (n_l) that are stacked above each other. It helps the system to draw generalized or abstract or deep patterns from the primitive low-level patterns already computed.

3. The internal dimension parameter for key, value and query attributes. This is used in conjunction with several self-attention heads. In this approach, this is assigned with a fixed value 64 and the first parameter is changed. This particular value is taken from the research carried out by Kitaev et al. (2019)

The assumption is that the more the value the better the result will be. But after some point, the rate of returns will be diminishing. A range of values are assigned to both the parameters and experiments will be conducted. The result will be computed and tabulated as shown below:

Ideally, all the variables should be changed and the result should be computed and tabulated. The pair of (n_l, n_h) values for which the resultant metric value is high, will be chosen as an ideal combination. But given the limitation on computation power and also the time, instead of choosing a classical Grid search which covers all the 100 combinations, a random set of pairs will be chosen. In addition to that, Kitaev et al. from their research, chose value to be 8 after hyper tuning for both n_l and n_h respectively. Given these conditions, fewer experiments will be conducted to find out the ideal pair.

n_l/n_h	1	3	5	8	10	
1						
3						
5						
8						
10						

Table 1: Parameters range table for Research question 1

4.3.2 Research Question 2

The second research question is intended to explore building the best possible constituent parser using only training data. In the previous case, the self-attention model is analyzed by trying out different values for the number of identical layers and the number of attention heads.

The input representation is expressed using an embedding matrix. Each component of input is represented using a row or column vector. Upon passing an index, the respective vector is retrieved. For example, in the case of words, each word is given a unique index i and represented by a row vector w_i with d dimension as follows:

$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1d} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2d} \\ \dots & \dots & \dots & \dots & \dots \\ w_{n1} & w_{n2} & w_{n3} & \dots & w_{nd} \end{bmatrix}$$

Similar is the case for position representation as well. In this experiment, the representation of input is trained along with other parameters. So, with every batch, it is expected to learn effectively. The various representation of input to the system with only training data is explored as follows:

1. Character embeddings matrix represents each character used in the training set with a vector. To be on the safer side, a maximum range for the number of characters is selected by analyzing the training dataset. The representation for each word will be constructed by using a bi-directional LSTM based RNN network. For each character in the word, the RNN is unrolled in sequence and the two last output vectors will be obtained by traversing left to right and vice-versa. Those two final vectors are merged to form the final vector to represent the word.

2. Instead of characters, an embedding matrix is constructed where each vector represents a word directly. For all the words that do not exist, a unique vector labeled as <UNK> will be used.

3. The third alternative is using both the character and word embeddings matrix. The dimension of both the matrix will be fixed and the resultant vectors from those matrices are added together to represent a word.

With character embeddings, the words which are not present in the training dataset will still be expected to be assigned with a reasonable appropriate vector. The character embeddings are expected to learn sub-word patterns and use them when constructing unknown words. For each word, the vector is constructed on the fly, whereas with word embeddings, the vectors are fixed. The word embeddings as stated by Mikolov et al. (Mikolov, Yih, & Zweig, 2013) is expected to learn similarities. But using <UNK> vector will not be as effective as the former strategy. By combining both character and word embeddings, the benefits of both are expected to be utilized.

4.3.3 Research Question 3

The third research question addresses the usage of Embeddings from Language Models (ELMo). In addition to that, what components or combination of them helps build an efficient parser. ELMo embeddings are constructed using a multi-layer, character convolution-based, bi-directional LSTM modules by training with a very large dataset at least over a billion tokens. The representations are deep and have provided state of the art results in various application systems. ELMo consist of 3 layers and the following section shows the experiments to be conducted with ELMo:

1. The first layer of ELMo does not involve any context and it is constructed by the concatenation of character embeddings with a feed-forward network. This will be used as a base model for these experiments and it is expected to give the least favorable result. This is much closer to using only character embeddings as specified in the previous research. But these are the result of training from external, rich resources such as Wikipedia. The expectation of a relatively better result is still valid.

2. The data of second layer of ELMo is the output of first bi-directional language model module. It also embodies the syntactic relations of the different words in a sentence. As constituent parsing also is a syntactic relation extraction, this layer output will be very useful.

3. The data of third layer of ELMo is the output of second bi-directional language model module. It embodies the deep, contextual relationship of words in a sentence. It will be interesting to see how does this layer influences the parser as compared to the second layer.

4. Using all the combinations of layers by training scalar weights which will be used as a weighted average of outputs of all the layers. This will be specific to the constituent parsing task and at the same time expected to use the benefits of all three layers.

All the above experiments will reveal how best ELMo can be used in building an effective constituent parser.

5 Evaluation

This section covers how the system will be evaluated as a constituent parser. In this master thesis, the focus is mainly on improving the quality of the system and the speed of the system can be improved by using good hardware. Dataset and the metric are the two most important factors play a major role in achieving the evaluation.

5.1 Dataset

The TüBa-D/Z treebank (Telljohann & Hinrichs, 2004) released by the University of Tübingen is used to train constituent parser. The dataset consists of 104,787 sentences that are divided randomly into training, dev, and test dataset of size 84,787, 10,000, and 10,000 respectively. The dev dataset is used to fine-tune the hyper-parameters of the model after getting trained with the training dataset. The test dataset is used to measure the quality of constituent parser at the end.

5.2 Metrics

To measure the quality of a constituent parser, the predicted syntax tree should be compared with the respective golden tree. A comparison of two trees involves comparing all their subtrees for a match. With the extraction of subtrees from both gold and predicted trees, there will be two sets of trees to be compared against.

The following table shows the confusion matrix for a constituent parser:

		Predicted Tree	
		Negative	**Positive**
Golden Tree	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

Table 2: Confusion matrix

The terms described in the above table are generic. But it involves some differences in the context of tree comparison, which is explained as follows:

- True Positives are set of subtrees which are present in both golden and predicted tree. That is the parser system predicted correctly the constituent in a sentence.
- True Negatives are set of subtrees which are not in both golden trees and predicted tree. In an information retrieval system, it makes sense, since the system can correctly leave the unrelated documents. But in this case, it will always be zero.
- False Positives are set of subtrees that are there in the predicted tree but not in the golden tree. This means that the parser wrongly selected a range of entities as a constituent.
- False Negatives are set of subtrees that are there in the golden tree but not in predicted. This means that parser failed to select a range of entities as a constituent which it should have.

The quality of the system can be measured by metrics such as Precision, Recall, and F1 measure. Each one can be computed using the formula as follows:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall}$$

Precision focuses completely on the output of the system which is being measured. It is possible to develop a parser that finds only one right constituent. This will give a 100% precision but a terrible constituent parser. At the same time, recall focuses completely based on the output of the golden tree. It is possible to develop a parser to list all possible combinations and regard them as constituents. This will end up giving a 100% recall.

F1 measure takes in a balanced approach by involving both precision and recall. It is preferred since it includes both false positive and false negatives. The F1 score is computed for each predicted output tree and an average is computed for all the trees in a batch.

6 Implementation

This section covers the implementation details for building a constituent parser. All the programs required to build the parser using neural networks are written in python (3.7.4) [1]. The model built is relatively complex and involves a lot of parameters. The training of these parameters approximately took one day using GeForce GTX 1050 Ti GPU. And so, it is important to have a good GPU configuration to run and train the model effectively.

6.1 Dependency Management

The main dependencies of the program are listed below:

- CUDA [2], a parallel computing platform and programming model developed by NVIDIA for general computing on GPU
- Pytorch (1.3.1) [3], an open-source machine learning framework, written in python, has immense support for tensor computation with GPU acceleration, automatic differentiation, and a huge set of library function to ease development experience
- Cython (0.29.14) [4], optimized static compiler, helps python code to call back and forth from and to C or C++ code natively. The decoder part of this project is written in Cython to improve parallelism and speed. All the possible combination of constituent formation has to be analyzed for forming the best-predicted tree using CKY method.
- AllenNLP (0.9.0) [5], an open-source NLP research library, built on PyTorch. ELMo module, an important component of this project, is taken from AllenNLP libraries.

6.2 Source Code

The source code[6] is highly inspired and derived from the published GitHub code [7] of Kitaev et al. (2019). There have been modifications made on Encoder, and ELMo modules to suit the project needs. Their work did not focus on using ELMo modules for the German dataset. Also, the code is upgraded to use the latest version of dependencies, API support for an extension. The code includes a CLI support for configuring all the parameters.

May et al. (2019) worked on building an ELMo based word representation for German language using German Wikipedia. The source and also the pre-trained data are shared in a GitHub project [8]. Their implementation followed exactly the works of E.Peters et al. (Peters et al., 2018) and also the model serialization format and accessibility followed the same standards of AllenNLP format.

For parallelism to work smoothly, the arbitrary sentences and words should be of same length to have same size matrix. Instead of fixing the size for all the sentences, a quick pass is made on one particular batch and the maximum number of words in a sentence and also the maximum number of characters in a word is obtained. Using this information the matrices are constructed and parallelism is achieved. In this process, to let the system explicitly know the start and end of sentence or word, unique

[1] https://www.python.org/downloads/release/python-374/
[2] https://developer.nvidia.com/cuda-zone
[3] https://pytorch.org/
[4] https://cython.org/
[5] https://allennlp.org/
[6] https://github.com/Kandy16/self-attentive-parser
[7] https://github.com/nikitakit/self-attentive-parser
[8] https://github.com/t-systems-on-site-services-gmbh/german-elmo-model

tokens are added at the beginning and end. The works of Kitaev et al. (2019) showed a considerable improvement in the quality of parser after adding these tokens.

6.3 Deployment

The model is stored in PyTorch specific serialization format. The model is trained on training dataset through batches. All these inputs are parameterized and depending on the choice of the number of epoch and number of batches, the training dataset is divided into respective batch sizes. By default, at regular intervals, four times during one epoch, the performance of the model is compared against the development dataset and when the F1 score is better than the best scored in past, the model is replaced. The training of the model can be stopped by specifying the number of epoch or when the model did not give better results consecutively for the configured number of times. All these configurations can be changed as per convenience before the start.

A docker[9] container image is a lightweight, standalone, executable package of software that includes everything needed to run an application. This makes the distribution of dependencies for a system much efficient and can avoid virtual hardware in most of the cases. A docker interface is created for both source code, and model, which can be executed on any machine as long as the CUDA is compatible.

[9]https://www.docker.com/

7 Analysis

This section reveals the results achieved for every research question and also analyses the possible reasons for the outcomes.

7.1 Research Question 1

In this section, how well the self-attention modules are effective in capturing constituency grammar of the German language is analyzed. The values for number of layers (n_l) and number of attention heads (n_h) are changed and for each combination, the model is trained and checked against development dataset four times in one epoch. Once when the development F1 score reached saturation or started decreasing over some time then the program is stopped. Once when the training is completed, the model is made to predict the sentences in the test dataset and compared against its actual output tree.

7.1.1 Optimal values for n_l and n_h

Since the number of possible combinations of different values would lead to 100 and each training takes more than a day, training for all the combinations will be time-consuming. The methodology used is a coarse-fine search to find the right combination. Initially, the combination of values is widespread covering the whole spectrum and the models are trained. Depending upon the results, the window of search is restricted but the granularity of search is increased. The following table shows the coarse search of parameter values and their results with the trained model:

n_l	n_h	Dev F1 (%)	Test F1 (%)
1	1	90.05	89.98
3	3	**92.47**	**92.44**
5	5	**93.83**	**92.77**
8	8	87.95	87.82
10	10	86.27	86.30

Table 3: Performance of models with different n_l and n_h values

A total of five searches and each one covering a new value for n_l between 1 and 10 and they are equally spaced. From the table above, it is evident that the performance of parser was improving until 5 and then it started declining. From the results, when the values are increased further, it is reasonable to assume that the performance of parser will decline even more.

From the table, it is clear that n_l values as 3 and 5 gave good results. A fine search is carried for values 3 to 5 and the results are published in the following table:

Instead of doing a fine search, all the values are tried including even values for n_h. The value 4 is also tried for n_l which is not tabulated. From the table, it is evident that the value 3 for n_l and 5 for n_h gives a maximum test F1 score of 93.68%. This combination is tested multiple times and the results are quite consistent. This combination will be used for further researches.

n_l	n_h	Dev F1 (%)	Test F1 (%)
3	1	91.80	91.78
	3	92.47	92.44
	5	**93.60**	**93.68**
	8	91.98	92.00
	10	92.84	92.86
5	1	92.75	92.78
	3	92.37	92.37
	5	92.83	92.77
	8	92.48	92.50
	10	92.27	92.24

Table 4: Performance of models with n_l values as 3 and 5

7.1.2 Self-attention module - how well it captures context?

From table 7.1.1, it is surprising to know that one layer and one attention head captures the essential pattern that it can achieve almost 90% test accuracy. This shows that self-attention heads are much effective in building a constituent parser. It captures the context of sentence much effectively which is used by high-level layers including the scoring module.

The self-attention matrix reveals how much each of the words is influenced by other words in the same sentence. The larger the value, the greater the influence that word has. The values need not sum up to 1, but for an understanding purpose, this relatively exposes how does the influence is distributed.

The figure 25 shows a self-attention matrix of a sentence with four words 'Hilfe für kriegstraumatisierte Frauen'. In this, the word 'Hilfe' has a relatively higher influence on the whole sentence. Each value is color-coded; the higher the value then the brighter it is.

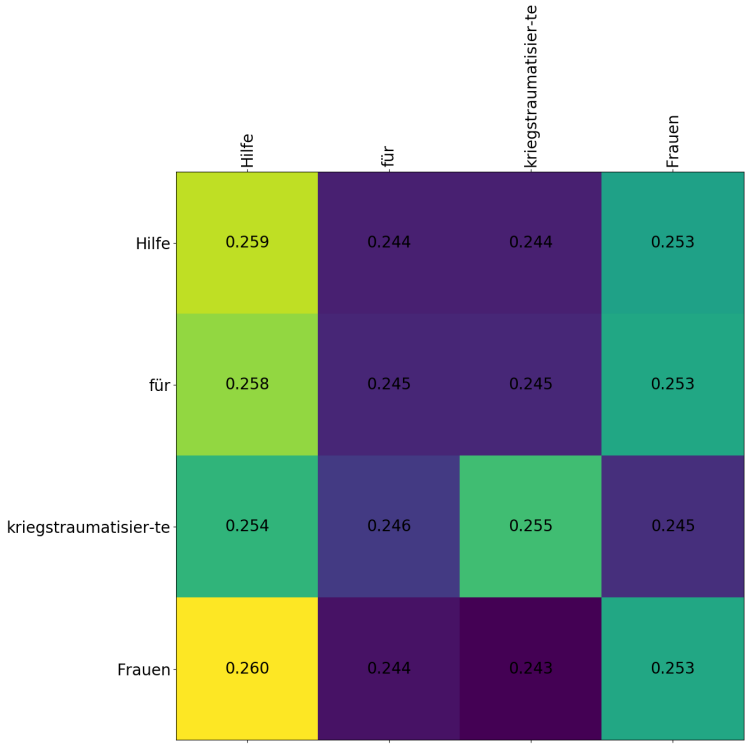

Figure 25: Self-attention head matrix

From the above experiments, n_l and n_h values as 3 and 5 gave best results. A total of 15 attention heads is being used in the model. Out of which, 5 attention heads act on input directly and independently of each other. The output of it is combined and passed to 5 attention heads and so on. So, in the second layer of attention heads, the influence of influences of the first 5 attention heads is computed or learned instead of the words. This will help the system to learn abstract patterns or a higher level of understanding of context. The final layer improves the understanding of context along the same lines.

The figures 26 and 27 show the self-attention matrix of 15 heads arranged in 3 layers. From the analysis of it with multiple sentences, the following are the findings:

- Every self-attention module in the same layer captures different influences. There are hardly any strict similarities. There is a slight variation from other modules. This means that the system can capture different influences and segregate them efficiently through training. This representation will be quite efficient as it is very clear for higher layers to act upon
- The influence is strong on fewer entities in the first layer. In the sentence 'Der Streit ist längst nicht entschieden', it is evident that the first layer contains distinct influences, the words 'der', 'Streit', 'ist', and 'längst' gets influenced predominantly by one word. Whereas in the subsequent layers, the influence is shared among other entities to a certain degree. This shows the abstraction or higher-level understanding of the influences of entities
- The behaviors are the same for longer sentences. In the figure 27, the sentence has 20 words and the same pattern is observed. Each self-attention model observes different influences and as the layers go up, the distinction of influences is distributed across all entities

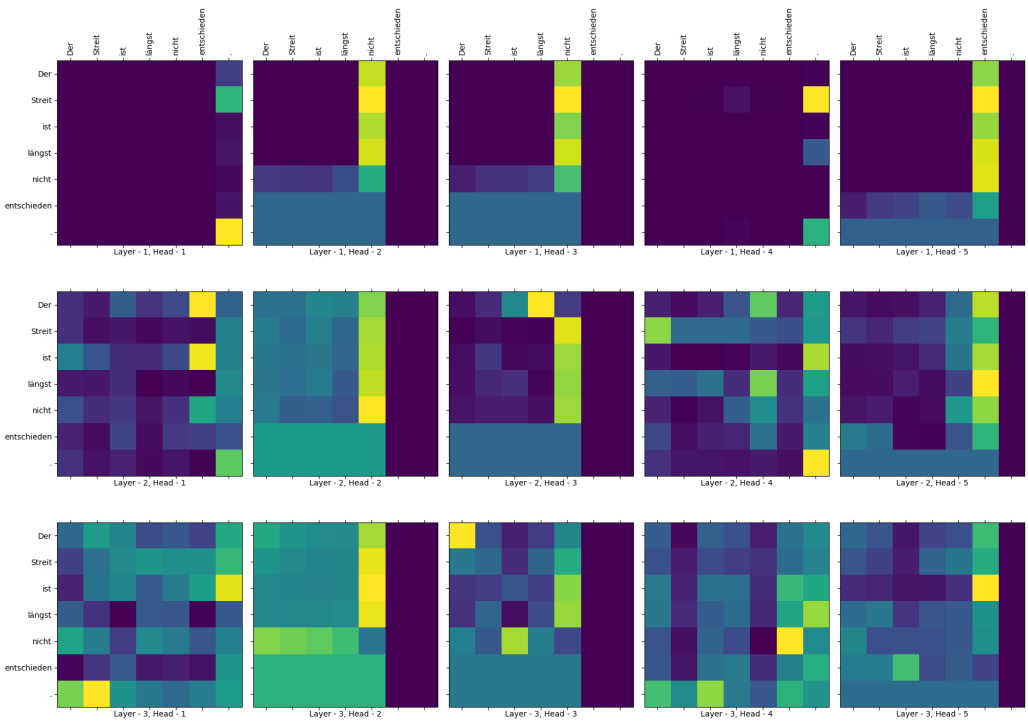

Figure 26: Self-attention matrix of small sentence

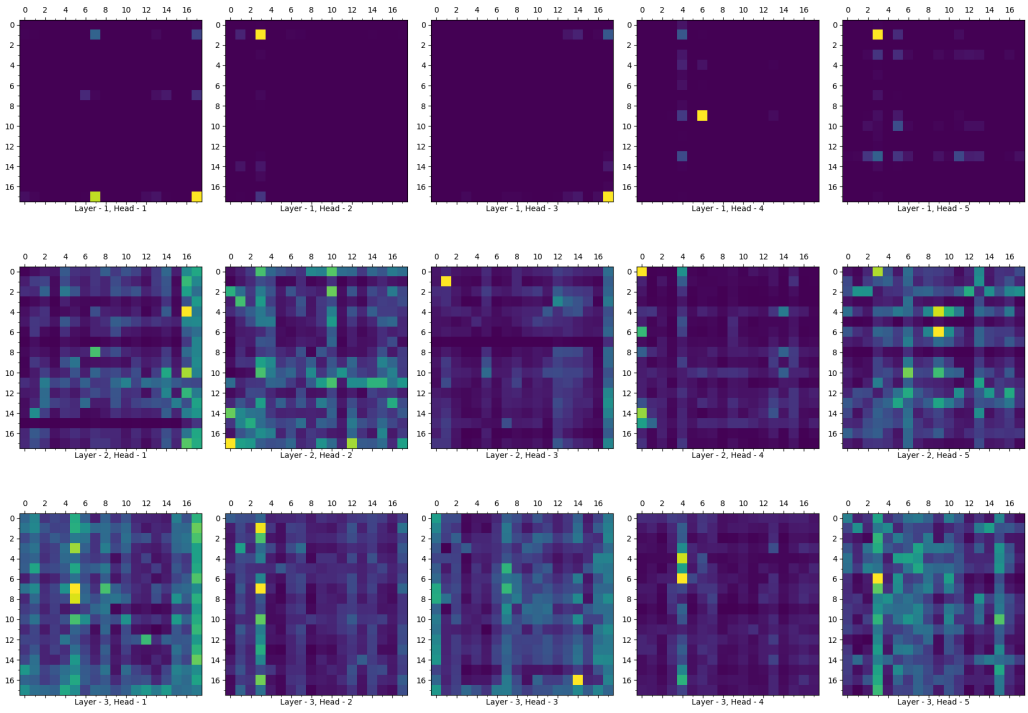

Figure 27: Self-attention matrix of long sentence

7.2 Research Question 2

In the previous section, the role of self-attention modules is analyzed. In this how effective a model can be implemented for constituent parser using only the dataset which is the sentence and the labeled syntax trees. The representation of the sentence is subjected to changes and the model is trained separately.

At first, the input is represented using character embeddings whereas each character is referred by a vector and the word is the summation of all the vectors that represent each character in the word. Secondly, each word in a sentence is represented by a unique vector. In the third experiment, both character embeddings and word embeddings are used together. They are added up to refer to a word in the sentence. The table below reveals the model development and test F1 score as follows:

Embeddings based models	Dev F1 (%)	Test F1 (%)
Character embeddings	**93.60**	**93.68**
Word embeddings	91.22	91.16
Character + Word embeddings	**94.17**	**94.10**

Table 5: Performance of different emdeddings based models

From the table, it is quite evident that embeddings play an important role in improving the accuracy and some important understanding are as follows:

- Character embeddings are better than word embeddings. By using the former, the F1 score is improved by 2.5% on the test dataset.
- Involving both character and word embeddings to represent input performs better than using them individually. A performance improvement of 0.42% F1 score on the test dataset is achieved. Although it is very marginal, this level of improvement beyond 90% is important
- It is observed that 4.8% of words in the test dataset do not appear in the training dataset. In case of word embeddings all unknown words are represented using a unique <UNK> vector whereas in character embeddings all unknown words can be represented with different vectors since all characters are covered in training dataset itself
- Also character embeddings are capable of learning sub-word patterns since the granularity of representation is much stronger than word embeddings. To add credibility to it, it is made sure that words that appear only in the test dataset but not in the training dataset are replaced with <UNK> token and so even in character embeddings there will be only one unique vector to represent all unknown words. Even with this arrangement, the character embeddings performed better in the same way than word embeddings. This adds validity that character embeddings representation offers an opportunity for the system to learn better.

7.3 Research Question 3

So far the best possible model for a constituent parser that could be developed using only the training dataset is tried out. In this section, with the use of a pre-trained model using external data, how efficient a constituent parser can be built is analyzed.

For this, ELMo modules which are built from German Wikipedia with more than 1 billion tokens are used to represent the input instead of character or word embeddings. This ELMo uses 2 layers of bi-directional LSTM on character convoluted input. The input layer which is the first layer is composed of a function of character embeddings network. The second layer is the first bi-directional LSTM network output and the third layer is built on top of the previous. The following table shows the performance of constituent parsing when these layer inputs are used separately and combined:

ELMo based models	Dev F1 (%)	Test F1 (%)
First layer	92.79	92.86
Second layer	**95.43**	**95.49**
Third layer	94.69	94.79
Combination of all layers	**95.76**	**95.87**

Table 6: Performance of different ELMo based models

From the table, it is clear that layered outputs influence the model much effectively and is analyzed as follows:

- The first layer which does not embody the context gives the least performance. The second layer which embodies the syntactic relationship of words helps the model to achieve a 95.49% F1 score. This is 1.39% more than the previous best which is given by using character and word embeddings. The third layer which embodies the deep contextual relationship of words gives a lesser F1 score by 0.7%. This gives clarity that constituency parser which predicts the syntactic structure of a sentence performs better with a layer output that embodies syntactic relations than other layered outputs.

- It is interesting that the combination of all layers performed better than using only the syntactic layer by 0.38%. In this, a scalar matrix influences what proportion of layered output has to be used. This is achieved by computing a weighted average of layered output influenced by the scalar matrix. This scalar matrix is fine-tuned by the training of the model to achieve the maximum objective score.

The figure 28 reveals the influence of layers when all of them used. As expected nearly 80% of second layer output is used. Usage of only the third layer gives a better F1 score than using only the first layer. But what is surprising is that it uses only 0.02% of the third layer which means it has almost no influence. What can be assumed from this is that all the relation required is given by the second layer itself and so the third layer information is redundant in this case.

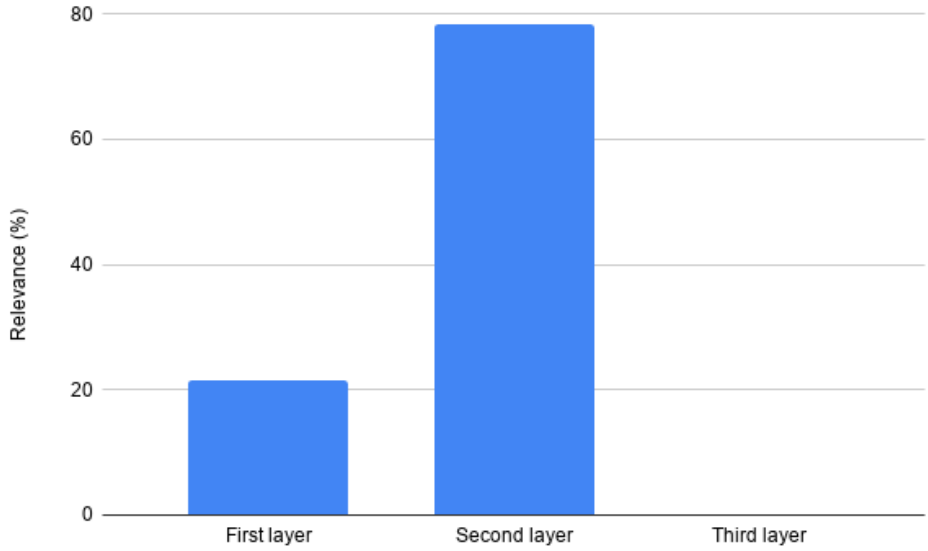

Figure 28: Importance of ELMo layers in Constituent parsing

7.4 Primitive Tree analysis

The model that gave the best F1 score (95.87%) is taken and their predicted syntax trees are compared against golden trees. All the noun and verb labels in the golden test dataset are compared against predicted labels and the accuracy is listed below:

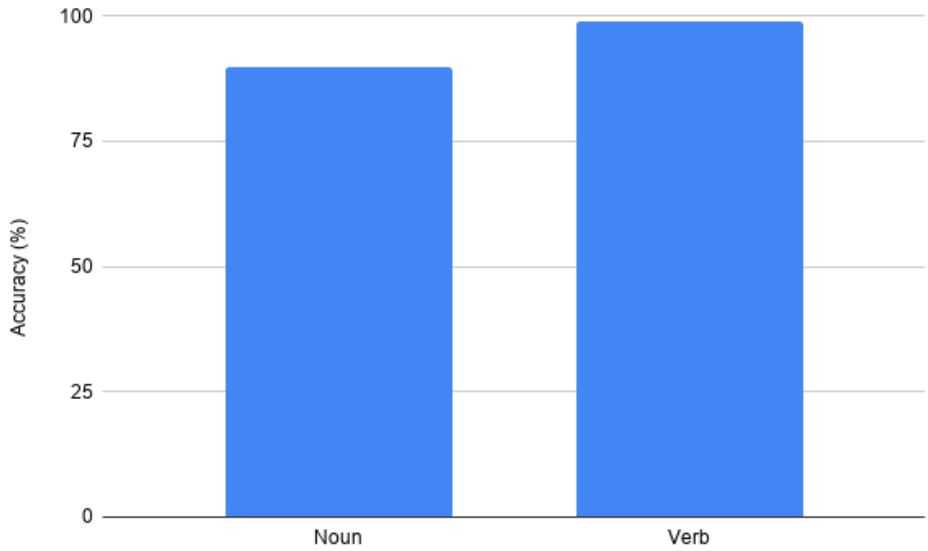

Figure 29: Noun and Verb prediction accuracy

The model achieved an accuracy of 89.8% and 99.0% for noun and verb labels respectively.

8 Conclusion and Future work

During this thesis work, considerable attempts are put to build a constituent parser using neural networks. Over the past, Recurrent neural networks have been used in building a parser and also many NLP applications. In this, self-attention neural network modules are used intensively to understand sentences effectively. This understanding is used by a CKY decoder to predict the syntactic structure in a sentence.

With multi-layered self-attention networks, constituent parsing achieves a 93.68% F1 score. Each of the self-attention heads learns distinct features and the heads in the higher layer learn abstract patterns. This is improved by using character and word embeddings as a representation of the input. An F1 score of 94.10% was the best achieved by constituent parser using only the dataset provided.

With the help of external datasets such as German Wikipedia, pre-trained ELMo models are used as representative of input. The second layer of the ELMo model has the greatest influence and combining all the layers with a weighted average achieves a 95.87% F1 score. So a self-attention module with ELMo based vectors as input representative achieves the maximum performance in building a constituent parser.

The research work did not scope in the analysis of results along the lines of computational linguistics. With the help of experts, linguistic analysis has to be carried out which will give clarity on what parts of grammar identification needs to be improved. This information will help in building some of the hypotheses on how to improve and experiments can be carried out. It will also give an understanding of what self-attention module is good or bad at capturing.

Devlin et al. made efforts to build Bidirectional Encoder Representations from Transformers (BERT) (Devlin, Chang, Lee, & Toutanova, 2018) models which are the state of art techniques to represent text. This work achieved the best results for 11 NLP tasks such as MultiNLI with 86.7%, and SQuAD v1.1 to 93.2% F1 score. Pre-existing BERT based representations can be used for the input used in constituent parsing and see whether performance is improved or not.

References

Alphex34. (2015). Convolutional neural network architecture. By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45679374. Retrieved from https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. arXiv: 1409.0473 [cs.CL]

Bengio, Y. [Y]. (1997). Convolutional Networks for Images, Speech, and Time-Series Parsing View project Oracle Performance for Visual Captioning View project. Retrieved from https://www.researchgate.net/publication/2453996

Bengio, Y. [Yoshua], Ducharme, R., Vincent, P., & Janvin, C. (2003, March). A neural probabilistic language model. *J. Mach. Learn. Res. 3*(null), 1137–1155.

Booth, T. L. (1969, October). Probabilistic representation of formal languages. In *10th annual symposium on switching and automata theory (swat 1969)* (pp. 74–81). doi:10.1109/SWAT.1969.17

Bottou, L. (2012). Stochastic Gradient Descent Tricks. In G. B. Montavon Grégoire and Orr & M. Klaus-Robert (Eds.), *Neural networks: tricks of the trade: second edition* (pp. 421–436). Berlin, Heidelberg: Springer Berlin Heidelberg. doi:10.1007/978-3-642-35289-8_25

Brants, S., Dipper, S., Hansen, S., Lezius, W., & Smith, G. B. (2002). The tiger treebank.

Callison-Burch, C. (2010). Syntactic constraints on paraphrases extracted from parallel corpora. (October), 196. doi:10.3115/1613715.1613743

Charniak, E. (1997, December). Statistical techniques for natural language parsing. *AI Magazine*, *18*(4), 33. doi:10.1609/aimag.v18i4.1320

Charniak, E. (2000). A maximum-entropy-inspired parser. In *Proceedings of the 1st north american chapter of the association for computational linguistics conference* (pp. 132–139). NAACL 2000. Seattle, Washington: Association for Computational Linguistics.

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. arXiv: 1406.1078 [cs.CL]

Deloche, F. (2017a). Lstm architecture. CC BY-SA 4.0. Retrieved from https://en.wikipedia.org/wiki/Recurrent_neural_network#/media/File:Long_Short-Term_Memory.svg

Deloche, F. (2017b). Recurrent neural network architecture. CC BY-SA 4.0. Retrieved from https://commons.wikimedia.org/wiki/File:Typical_cnn.png

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: pre-training of deep bidirectional transformers for language understanding. arXiv: 1810.04805 [cs.CL]

Dos Santos, C. & Gatti, M. (2014). Deep convolutional neural networks for sentiment analysis of short texts. In *Proceedings of coling 2014, the 25th international conference on computational linguistics: technical papers* (pp. 69–78).

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, *12*, 2121–2159. doi:10.1109/CDC.2012.6426698. arXiv: arXiv:1103.4296v1

Elman, J. L. (1990). Finding Structure in Time. *COGNITIVE SCIENCE*, *14*(1), 179–21. doi:10.1207/s15516709cog1402_1

Gaddy, D., Stern, M., & Klein, D. (2018). What's Going On in Neural Constituency Parsers? An Analysis. In *Proceedings of the 2018 conference of the north american chapter of the association for computational linguistics: human language technologies, volume 1 (long papers)* (pp. 999–1010). Stroudsburg, PA, USA: Association for Computational Linguistics. doi:10.18653/v1/N18-1091

Gildea, D. & Palmer, M. (2002). The necessity of parsing for predicate argument recognition. In *Proceedings of the 40th annual meeting on association for computational linguistics* (pp. 239–246). ACL '02. Philadelphia, Pennsylvania: Association for Computational Linguistics. doi:10.3115/1073083.1073124

Goldberg, Y. (2016). A Primer on Neural Network Models for Natural Language Processing. *Journal of Artificial Intelligence Research*, *57*, 345–420.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. doi:arXiv:1207.0580. arXiv: 1207.0580

Hochreiter, S. & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. doi:10.1162/neco.1997.9.8.1735

Jurafsky, D. & Martin, J. H. (2008). Speech and language processing: An introduction to speech recognition. *Computational Linguistics and Natural Language Processing.2nd Edn., Prentice Hall, ISBN*, *10*(0131873210), 794–800.

Kitaev, N. & Klein, D. (2019, June). Constituency Parsing with a Self-Attentive Encoder. (pp. 2676–2686). Association for Computational Linguistics (ACL). doi:10.18653/v1/p18-1249

Klein, D. & Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st annual meeting on association for computational linguistics - volume 1* (pp. 423–430). ACL '03. Sapporo, Japan: Association for Computational Linguistics. doi:10.3115/1075096.1075150

LeCun, Y. & Bengio, Y. [Y.]. (1995). Convolutional networks for images, speech, and time-series. In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks*. MIT Press.

Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., & Amir, S. (2015, September). Finding function in form: compositional character models for open vocabulary word representation. In *Proceedings of the 2015 conference on empirical methods in natural language processing* (pp. 1520–1530). Lisbon, Portugal: Association for Computational Linguistics. doi:10.18653/v1/D15-1176

Marcus, M. P. (1993). J93-2004.pdf.

May, P. (2019). German ELMo Model. Retrieved from https://github.com/t-systems-on-site-services-gmbh/german-elmo-model

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013a). Efficient estimation of word representations in vector space. arXiv: 1301.3781 [cs.CL]

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013b). Efficient estimation of word representations in vector space. *CoRR*, *abs/1301.3781*. arXiv: 1301.3781. Retrieved from http://arxiv.org/abs/1301.3781

Mikolov, T., Yih, W.-t., & Zweig, G. (2013). Linguistic regularities in continuous space word representations. In *Hlt-naacl* (pp. 746–751).

Parikh, A. P., Täckström, O., Das, D., & Uszkoreit, J. (2016). A Decomposable Attention Model for Natural Language Inference. doi:10.18653/v1/N16-1062. arXiv: 1606.01933

Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (emnlp)* (pp. 1532–1543).

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. arXiv: 1802.05365 [cs.CL]

Petrov, S., Barrett, L., Thibaux, R., & Klein, D. (2006). Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st international conference on computational linguistics and the 44th annual meeting of the association for computational linguistics* (pp. 433–440). ACL-44. Sydney, Australia: Association for Computational Linguistics. doi:10.3115/1220175.1220230

Richard Socher. (2011). Parsing Natural scenes and natural language with recursive neural networks. doi:10.1007/s10107-018-1337-6. arXiv: arXiv:1207.6324

Schuster, M. & Paliwal, K. K. (1997, November). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681. doi:10.1109/78.650093

Schuster, M. & Paliwal, K. (1997, November). Bidirectional recurrent neural networks. *Trans. Sig. Proc. 45*(11), 2673–2681. doi:10.1109/78.650093

Schwenk, H., Déchelotte, D., & Gauvain, J.-L. (2006, January). Continuous space language models for statistical machine translation. doi:10.3115/1273073.1273166

Socher, R., Bauer, J., Manning, C. D., & Ng, A. Y. (2013). *Parsing with Compositional Vector Grammars*.

Socher, R., Manning, C. D., & Ng, A. Y. (2010). *Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks*.

Stern, M., Andreas, J., & Klein, D. (2017). A Minimal Span-Based Neural Constituency Parser. In *Proceedings of the 55th annual meeting of the association for computational linguistics (volume 1: long papers)* (pp. 818–827). Stroudsburg, PA, USA: Association for Computational Linguistics. doi:10.18653/v1/P17-1076

Sutskever, I., Martens, J., Dahl, G., & Hinton, G. (2013). On the importance of initialization and momentum in deep learning. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, (2010), 8609–8613. doi:10.1109/ICASSP.2013.6639346. arXiv: arXiv:1301.3605v3

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104–3112).

Svozil, D., Kvasnieka, V., & Pospichal, J. (1997). Chemometrics and intelligent laboratory systems Introduction to multi-layer feed-forward neural networks. *Chemometrics and Intelligent Laboratory Systems*, *39*, 43–62.

Taylor, A., Marcus, M., & Santorini, B. (2003). The Penn Treebank: An Overview, 5–22. doi:10.1007/978-94-010-0201-1_1

Telljohann, H. & Hinrichs, E. (2004, March). Stylebook for the tübingen treebank of written german (tüba-d/z).

Telljohann, H., Hinrichs, E., Kübler, S., Kübler, R., & Tübingen, U. (2004). The tüba-d/z treebank: annotating german with a context-free backbone. In *In proceedings of the fourth international conference on language resources and evaluation (lrec 2004* (pp. 2229–2235).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2017, June). Attention Is All You Need. arXiv: 1706.03762. Retrieved from http://arxiv.org/abs/1706.03762

Werbos, P. J. (1990). Backpropagation Through Time: What It Does and How to Do It. *Proceedings of the IEEE*. doi:10.1109/5.58337

Yang, Z., Yang, D., Dyer, C., He, X., Smola, A., & Hovy, E. (2016). Hierarchical Attention Networks for Document Classification. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1480–1489. doi:10.18653/v1/N16-1174. arXiv: 1606.02393

Younger, D. H. (1967). Recognition and parsing of context-free languages in time n3. *Information and Control*, *10*(2), 189–208. doi:https://doi.org/10.1016/S0019-9958(67)80007-X