Universität Koblenz - Landau
Department für Informatik,
Institute für ComputerVisualistik
Prof. Dr. Karin Harbusch
Denis Memmesheimer

January 23, 2020

Master Thesis Kandhasamy Rajasekaran

Deep Learning techniques applied to Constituency parsing of German

1 Introduction

Constituency or Syntactic parsing is the process of determining the syntactic structure of a sentence by analysing 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 [1].

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

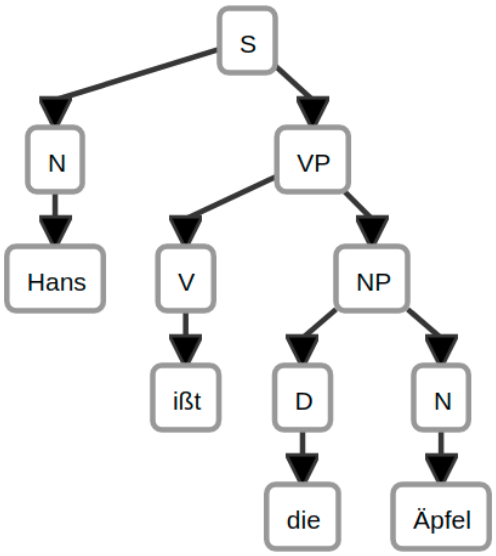

Figure 1: An example parser tree

In this tree, the abbrevations S, D, N, V, NP and VP refers 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, die and Äpfel. 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 as immediate parent of each word. The rest of 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. [2] in their 'Speech and Language processing' book stated that the parse tree is useful in many applications related to NLP. Infact 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 times, it is also an important intermediate representation for the semantic analysis of sentence. Thus it plays an important role in application such as relation extraction, semantic role labeling [3], question answering and paraphrase detection [4].

In this thesis, we will apply state of the art techniques to perform constituent parsing on German dataset released by University of Tübingen.

2 Related work

In the past, handwritten grammars 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 accomodate 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. In order to build statistical parsing system, the need for a versatile dataset is very critical.

Marcus et al.[5] prepared Penn Tree Bank dataset for 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 millions of words with POS tags [6]. These 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 lead to a empirical/data-driven approach to parsing than rule based approach. Some of the advantages of a treebank are reusability of lot of other subsystems such as POS taggers, parsers and a standard way to evaluate multiple parsing systems.

Probabilistic context free grammars (PCFG) 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. Chart based CKY algorithm is a bottom up parsing methodology which uses these probability scores to make decision about what combination of constituents are 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. PCFG parsing methods gives only 73% F1 score. Lexicalization of PCFG will create more rules pertaining to specific cases and the probability scores can be trained for each production rule. This indeed improves accuracy of parsing using Charniak method. But it increases production rules exponentially which results in sparseness. Kelin and Manning 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 [7]. It achieved 85.7% F1 score and does not lead to too many production rules but at the same time captures the essence of context. Untill now the rules of grammar was handwritten which were picked carefully by experts. Petrov and Klein came up with a unsupervised ML approach which use base categories and learns subcategorizing and splitting using treebank data. They used expectation maximization (EM) algorithm, like Forward-Backward for HMMs but constrained by tree structure.

On the other hand, 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 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. [[8], [1], [9]] 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 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. [10] implemented a minimum span neural network based constituency parser and achieved an accuracy of 91.79 F1 score on Penn Treebank dataset. They used an encoder-decoder architecture where in the encoder converts the input into a different reprsentation with context information and the decoder uses this augmented information to build the tree and get trained to become better. Bi-directional LSTM modules were used to encode the words in a sentence. This encoding gives out two vectors for each word containing the sentential context from 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 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 bottom up approach where in 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 whole tree beneath; whereas the top down parsing can refer to only local information. These results credit the ability of Bidirectional LSTM modules in capturing a lot of complex relations among words in a sentence. By using rich and expressive word vector representations, the encoding and decoding architectures are made simpler.

Kitaev et al. [11] also implemented a encoder-decoder based neural network architecture for constituency parsing and they achieved state of the art accuracy of 95.13 F1 score on Penn Treebank dataset. Their parser also outperforms all the previous parsers on 8 of the 9 languages in SPMRL dataset. In this, instead of LSTM modules, Attention modules [12] 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 columns and rows as words/components of sentences/inputs of each. At the same time, attention modules can also be applied to a sentence itself to express which part of sentence is dependent on which other words. These 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 the manner in which information is propagated between different locations in the sentence. A chart based decoder implemented by Stern et al. [10] is used along with the modifications from Gaddy et al [7]. They also found that positional information plays an very important role in parsing and inclusion of ELMo(Embeddings from Language Models) improved their accuracy by 3%.

3 Background Study

In this thesis, we develop a neural network based constituent parser for german dataset released by Tübingen. The required information to understand 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 connected with each other. That defines its ability to learn.

Feed-forward neural network [**Svozil1997**] 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 where in label data regarding classification is mandatory. These networks should contain a input layer which takes in data and a 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 the Figure 3

A model, data, objective or loss function, and optimizer are important components of any machine learning approach.

, to capabilities defines the architecture s of neural networks which vary mostly in how the neurons are connected. That defines the model

Each neuron in the network takes in multiple scalar inputs and gives out a scalar output. It uses parameters (weights and bias) to perform linear transformation of the input and most often applies non-linear function subsequntly.

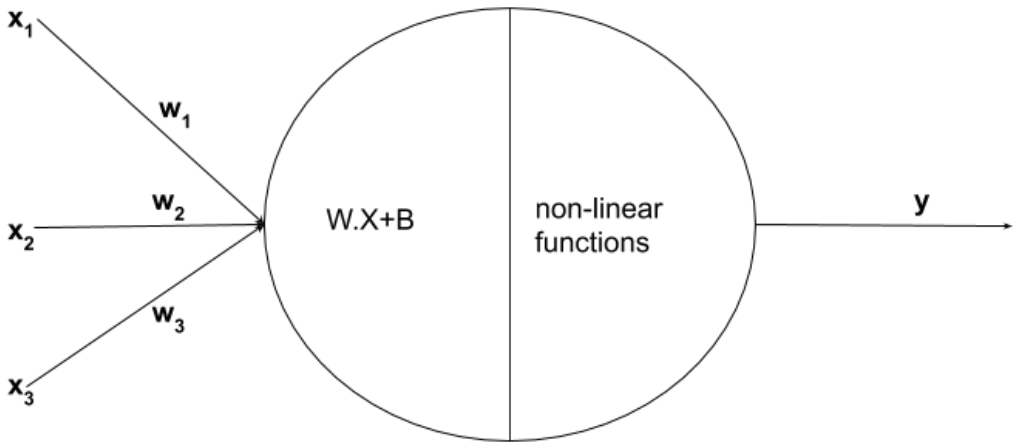

Figure 2: Single Neuron

to linearly transform the valuemultiplies each input by a weight and then sums them, adds the result with a bias, applies a non-linear function at the end, which gives out a scalar output.

3.1.1 Feed-forward network

There are different architectures of neural networks which vary mostly in how the neurons are connected to each other and how the weights are managed. Feed-forward neural networks [**Svozil1997**] can have multiple layers and each neuron in one layer is connected with every other neuron in the subsequent layer as given in the Figure 3

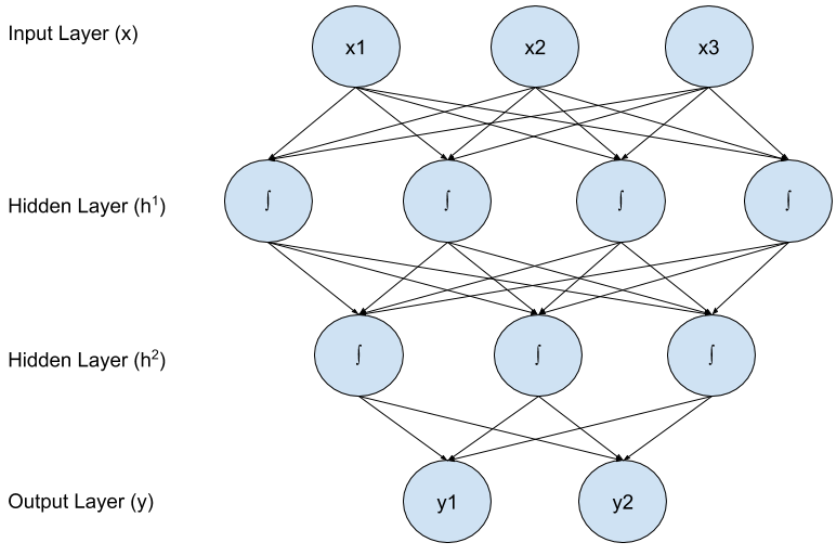

Figure 3: A Feed-Forward neural network.

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 a 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 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 second hidden layer as shown in equation 3.

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

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

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

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

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 Functions

The activation functions help the neural network models to approximate any nonlinear function. Different activation functions pose different advantages. Some popular activation functions are sigmoid, hyperbolic tangent and rectifiers [**Goldberg2016**]:

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

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

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.

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

3.2 Training a neural network

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 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 [**Bottou2012**]. A number of extensions exists, such as Nesterov Momentum [**Sutskever2013**] or AdaGrad [**Duchi2011**]. Some loss functions that exist are hinge loss (binary and multiclass), log loss and categorical cross-entropy loss [**Goldberg2016**].

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 $y = y^1, y^2, \ldots, y^n$ be representing the target multinomial distribution over the labels $1, 2, \ldots, n$ and let $\hat{y} = \hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n$ 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.

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

The overfitting in neural networks will cause the trained system to perform well only for trained data but not on the test data. This can be minimized by using regularization techniques such as L_2 regularization and dropout [**Hinton2012**]. 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 other half of neurons. This helps to prevent the network from relying on only specific weights.

3.3 Convolution Neural networks

Convolution neural networks - convolutions, diagram Advantages of convolutions Character convolution networks

3.4 Recurrent Neural networks

In case of text data, the input is sequential and of unknown length, where the ordering of words is important. Techniques such as continuous bag of words [**DBLP:journals/corr/abs-1301-3781**] 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) [**Bengio1997**] 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**]. 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 4. 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.

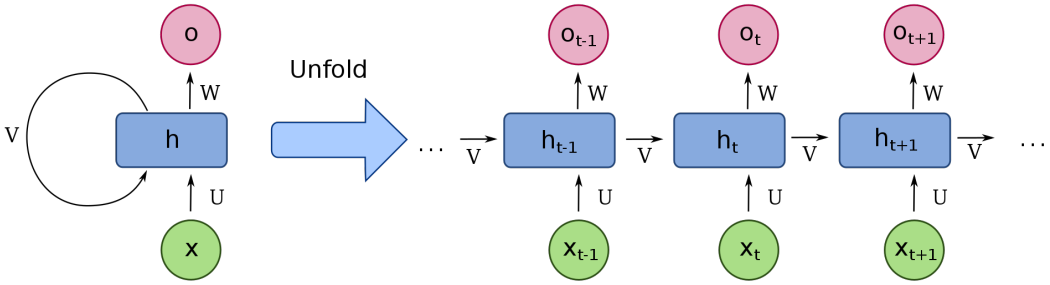

Figure 4: A basic example of RNN architecture [**WikipediaEN_RNN_unfold**].

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

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

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 [**Werbos1990**].

3.5 Advanced Recurrent Neural networks

3.5.1 LSTM or GRU

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. LSTM networks [**Hochreiter1997**] solve the vanishing gradient problem by introducing memory cells which 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 into 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.

3.6 Bidirectional

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 and 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.

The mentioned approaches take only Wikipedia sentence as an input. This can be extended further by feeding information which gives context about the input sentence. This extra information might enable the system to perform efficiently in classifying whether a news item is fake or not. Attention mechanism in neural networks is used to take two inputs and convert it into one common representation. This is done by configuring the level of attention that needs to be given to different parts of each input and combine them to form a single output. Most often the inputs will be matrices and the output will be a vector. Bidirectional RNNs can be used to convert both input and context information into matrices. The fixed output vector is then fed into a deep feed-forward neural network to predict a class. Attention mechanism with RNN gives state of the art results in many NLP applications such as Natural Language Inference [**Parikh2016**], Machine Translation [**Bahdanau2014**] and Document classification [**Yang2016**].

3.7 Data Representation

3.7.1 One Hot

3.7.2 Word embeddings, Char embeddings

3.7.3 ELMo

3.8 Attention

3.8.1 Self-attention

3.9 Parsing techniques

CKY Model

4 Approach

In this thesis work, deep learning techniques are used to develop constituent parser for german language. This section explains in detail about the architecture used and different components included in it. In this approach, the german dataset released by by Tübingen is used to train the model effectively.

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 network are effective in capturing constituency grammar of 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 utilised.
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 from the works of Kitaev et al. [11], which involves an encoder-decoder architecture as shown in figure 5:

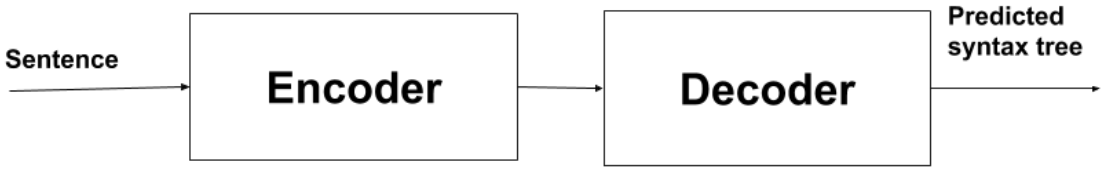

Figure 5: Broad Architecture

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 which outputs syntax tree. The building of syntax tree is seen as a continuous iteration of merging of certain subsequent words or merges that already happened in hierarchy. The decoder method involves steps to figure out what combination of merges are the best to build a good syntax tree.

The training of deep learning system involves a objective function to compare the prediction against golden output to allow fine tuning of different components of 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 from a state of art transformer neural network which predominantly uses self-attention modules, whereas the decoder module uses 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 are also used separately. The system does not use any other pre-processed information such as Parts Of Speech (POS) obtained by 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 which encompasses the sub-word patterns as well as the overall context of a sentence. This is achieved through a multi-layered, multi-head attention modules backed up with feed-forward networks as shown in the figure 6

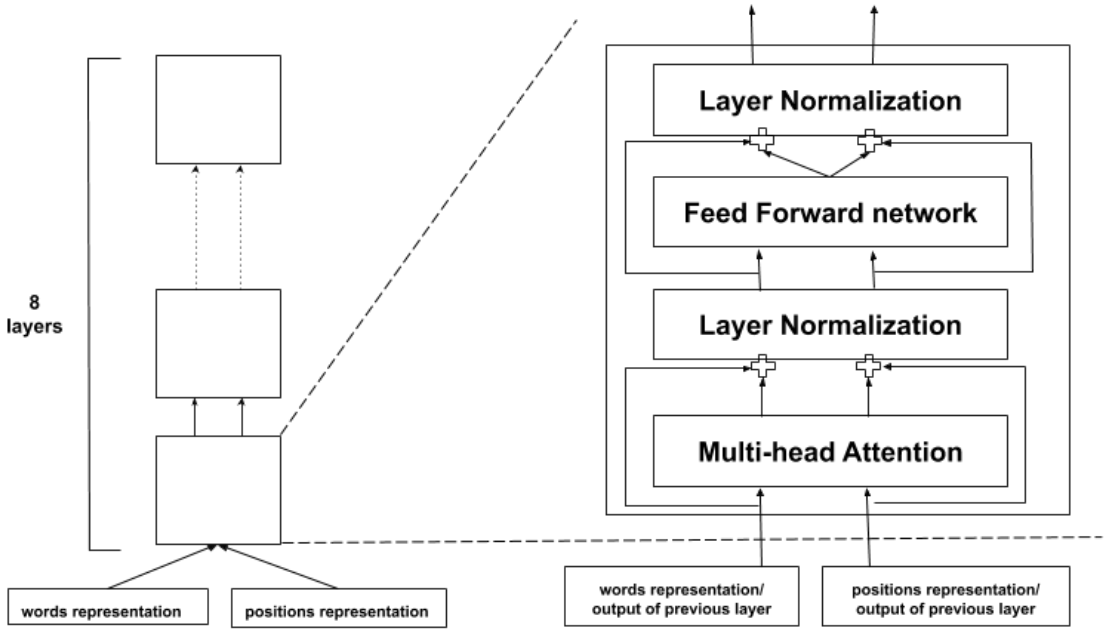

Figure 6: Encoder module

The input to this system is passed through 8 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. A Multi-head attention uses 8 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. [11] 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.

The output of Multi-head attention module is then passed to Layer Normalization by mixing with

residual component of input. The Normalization of intermediate result along with objective function will help the gradient based optimizer to arrive 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 7 shows the outline of what happens inside a single attention head:

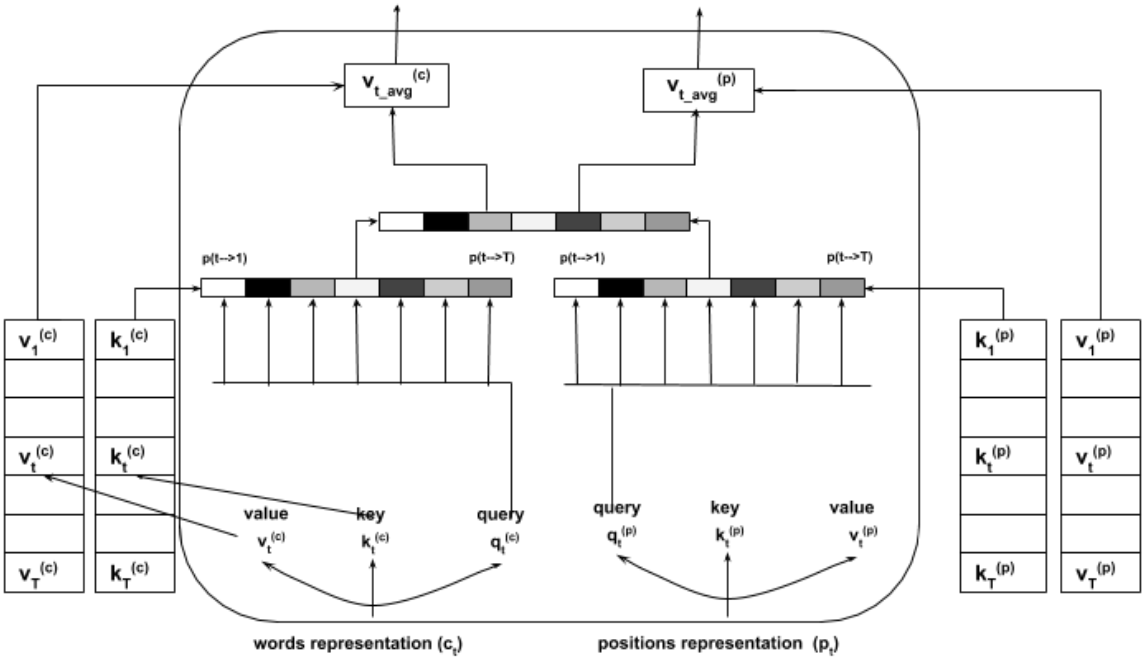

Figure 7: Single attention head

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 8

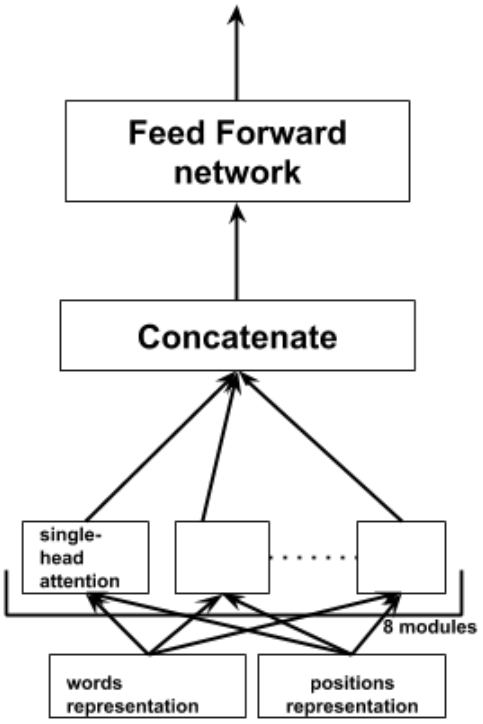

Figure 8: Mult-head attention module

The concatenated output fed through feed-forward to derive the patterns across multiple attentions in a sentence. These are then fed to Layer Normalization block as shown in the figure 6.

4.2.2 Decoder

As shown in the section, the chart parser is inspired from the works of Stern et al. and Gaddy et al. But have to explain here combination of each and every word is a CFG rule and the score determines the probability of using it

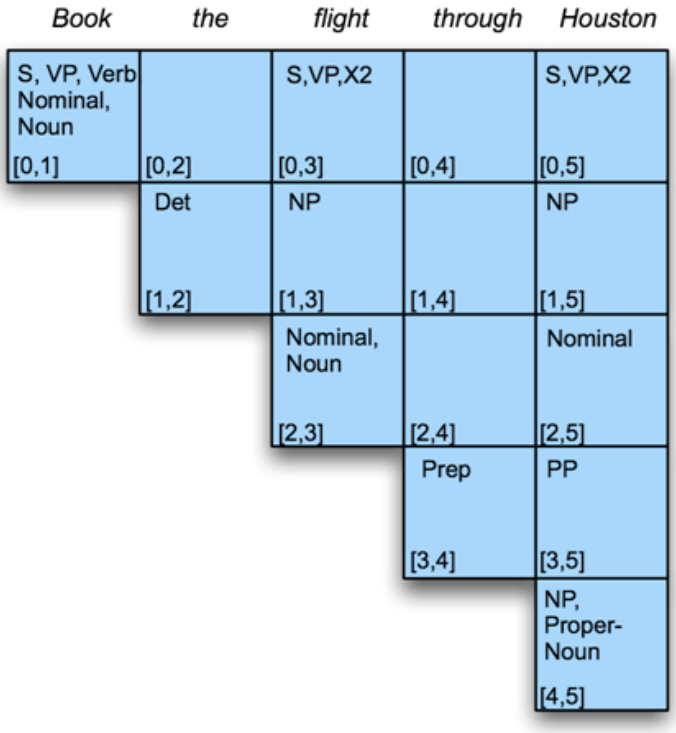

Figure 9: CKY parsing

4.2.3 Objective function

max - margin, scoring . how does it work with CKY parsing and decoding

4.3 Experiments

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

4.3.1 For research question1:

The first research question address how effectively 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 number of 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. [11]

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 fixed both the parameters and experiments will be conducted. The result metric will be computed and tabulated as show below:

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

Table 1: Parameters range table for Research question 1

Ideally all the variables should be changed and result metric 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 limiation on computation power and also the time, instead of choosing a classical Grid search which covers all the 25 combinations, 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, at the max, 5 experiments will be conducted which will be around the value 8.

4.3.2 For research question2:

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 analysed by trying out different values for number of identical layers and 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 a 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:

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

Similiar 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 no.of characters is selected by analysing 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 viceversa. 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 does not exist, a unique vector labelled as UNKNOWN_WORD will be used.
3. The third alternative is using both character and word embeddings matrix. The dimension of both the matrix will be fixed and the resultant vector from those matrices are added together to represent a word.

With character embeddings, the words which are not present in training dataset will still be expected to be assigned with a reasonable appropriate vector. The character embeddings is expected to learn sub-word patterns and use it when constructring unknown words. For each word, the vector is constructed on the fly, where as with word embeddings, the vectors are fixed. The word embeddings as stated by Mikolov et al. (???) is expected to learn similarities. But using UNKNOWN_WORD 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 utilised.

4.3.3 For research question3:

The third research question address the usage of Embeddings from Language Models (ELMo). And also what components or combination of them is helpful in building an efficient parser. ELMo are constructed using a multi-layer, character convolution based, bi-directional LSTM modules by training with a very large dataset atleast over a billion tokens. The representation are deep and have provided state of the art results in various application systems. As stated in the section (???), it outputs 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 concatenatation 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 least favourable result. This infact is much closer to using only character embeddings as specified in the previous research. But these are result of trainings 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. As specified in the section (???) it 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.

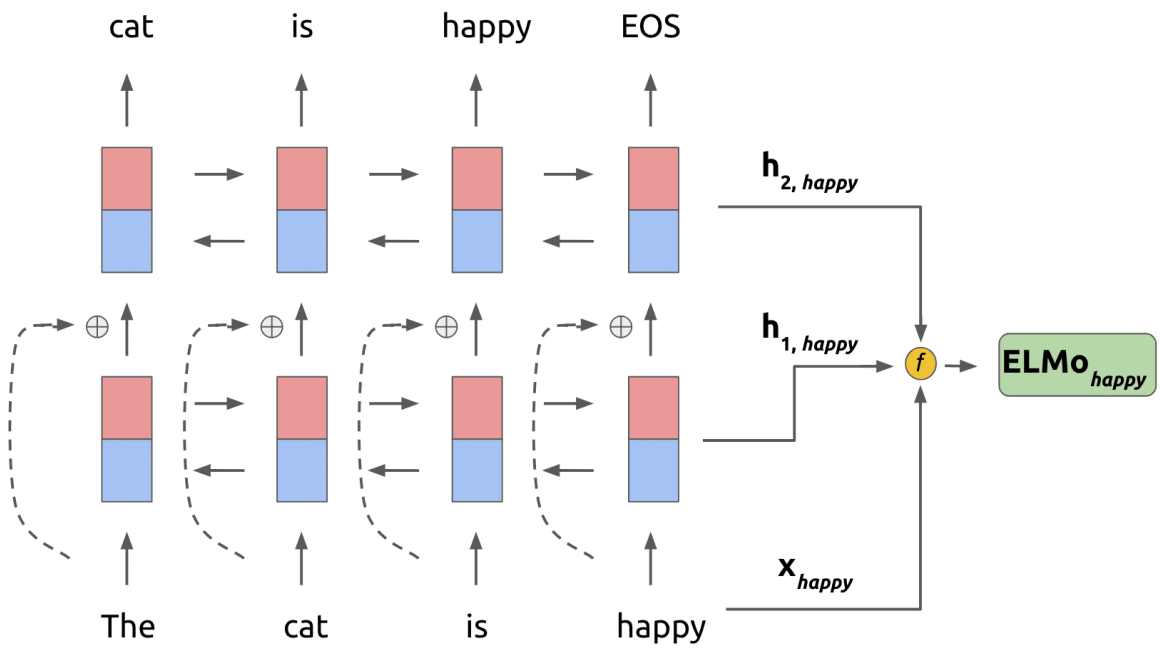

Figure 10: ELMo

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 sentence. It will be interesting to see how does this layer influences the parser as compared to the second layer.

4. Using all the combination of layers by training scalar weights which will be used as 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

Constituent parsing evaluation. The test data should be different from the training or validation data which is used by the system.

5.1 Datasets

Tübingen dataset - write about history, coverage, versatility, who and what system have used it

5.2 Metrics

Inorder to measure the quality of a constituent parser, the predicted syntax tree should be compared with the respective golden tree. Comparison of two trees involves comparing all their subtrees for a match. With the extraction of subtrees from both gold and predicted tree, there will be two set 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 sub trees which are present in both golden and predicted tree. That is the parser system predicted correctly the constituent in a sentence.
- Tree Negative are set of sub trees which are not in both golden tree 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 sub trees which are there in predicted tree but not in golden tree. This means that the parser wrongly selected a range of entities as a constituent.
- False Negatives are set of sub trees which are there in 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:

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

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

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

Precision focuses completely on the output of system which is being measured. It is possible to develop a parser to list all possible combinations and regard them as constituent. This will give a 100% precision but a terrible constituent parser. Recall focuses completely based on the output of golden tree. It is possible to develop a parser which finds only one constituent. This will endup 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. 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.

The F1 score is computed for each and every 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 took one to two days 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 differentation, 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 analysed 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. [11]. There has been modifications made on Encoder, and ELMo modules to suit to the project needs. Their work did not focus on using ELMo modules for german dataset. Also, the code is upgraded to use latest version of dependencies, API support for extension. The code includes a CLI support for configuring all the parameters.

May et al. [**GerElmo**] worked on building a 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. (ELMo paper) and also the model serialization format and accessibility followed the same standards of AllenNLP format.

[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

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 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 tokens are added at the beginning and end. The works of Kitaev et al. [11] showed a considerable improvement in the quality of parser after adding these token.

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 number of epoch and number of batches, the training dataset is divided into respective batch size. By default, at regular intervals, for four times during one epoch, the performance of model is compared against development dataset and when the F1 score is better than the best scored in past, the model is replaced. The training of model can be stopped by specifiying number of epoch or when the model did not give better results consecutively for configured number of times. All these configurations can be changed as per convinence before 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 most of the cases. A docker interface is available for the 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 each and every research questions and also analyses the possible reasons for the outcomes.

7.1 Research Question1:

In this section, how well the self attention modules are effective in capturing constituency grammar of German language is analysed. 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 for four times in one epoch. Once when the development F1 score reached a saturation or started decreasing over a period of time then the program is stopped. Once when the training is completed, the model is made to predict the sentences in 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 combination of different values would lead to 100 and each training takes more than a day, training for all the combination will be time consuming. The methodology used is coarse-fine search to find the right combination. Initially the combination of values is wide spread 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 were 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 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 the table 7.1.1, it is surprising to know that one layer and one attention head captures essential pattern that it is able to 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.

Self-attention matrix reveals how much each of the words are 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 expose how does the influence is distributed.

The figure 11 shows self-attention matrix of a four word sentence 'Hilfe für kriegstraumatisier-te Frauen'. In this, the word 'Hilfe' has relatively higher influence on the whole sentence. Each value is color coded; the higher the value then the brighter it is.

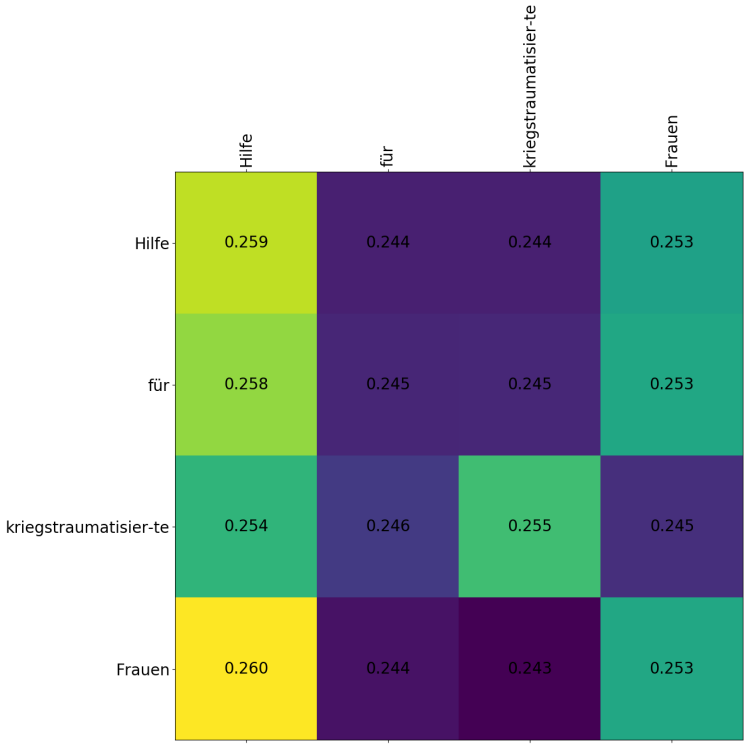

Figure 11: Self-attention head matrix

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 of input directly and independent 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 influence of 5 attention heads is computed or learned instead of the words. This will help the system to learn abstract patterns or higher level of understanding of context. The final layer improves the understanding of context along the same lines.

The figures 12 and 13 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 is able to capture different influences and segregrate 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 at 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 influences by predominantly by one word. Where as 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 the sentence has 20 words and the same pattern is observed. Each self attention model observers differnt influences and as the layers go up, the distinction of influences is distributed across all entities.

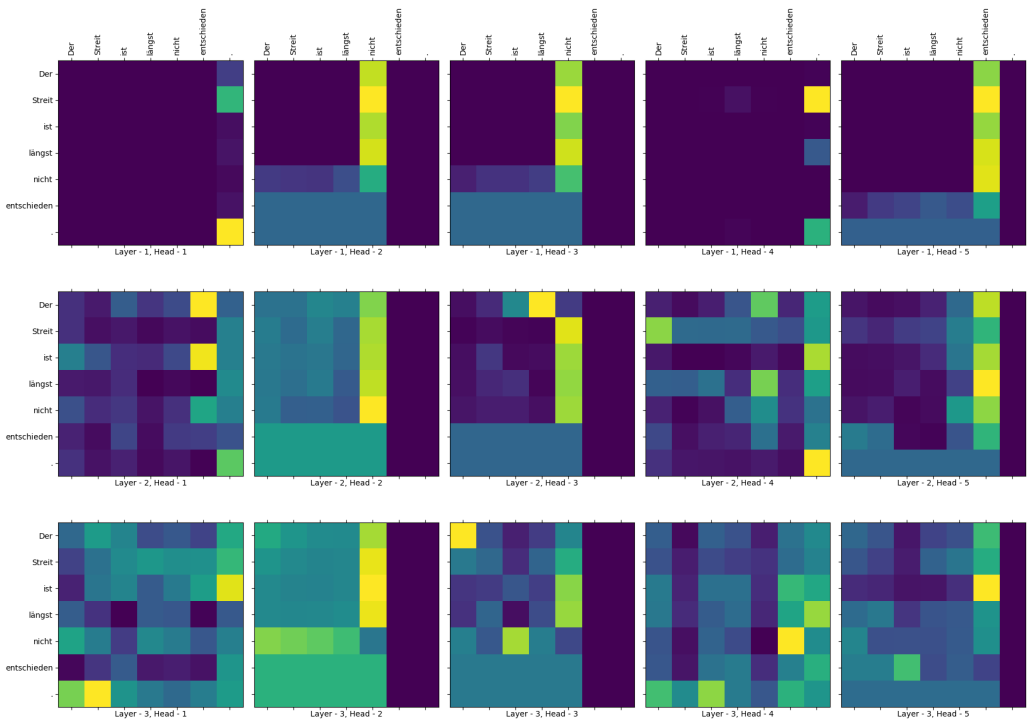

Figure 12: Self-attention matrix of small sentence

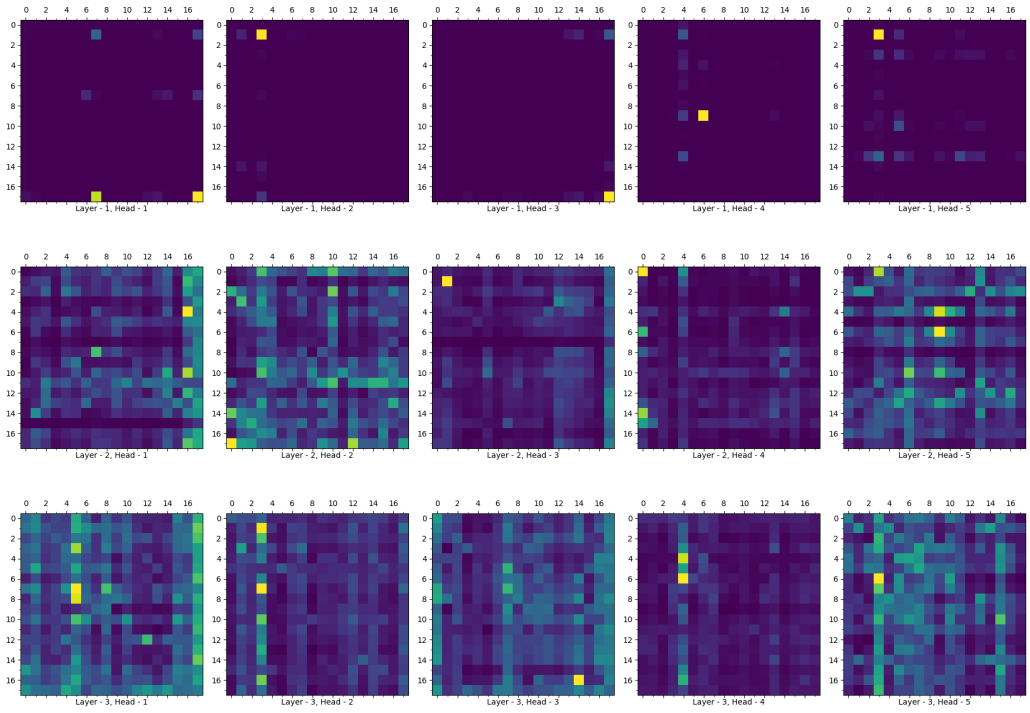

Figure 13: Self-attention matrix of long sentence

7.2 Research Question 2

In the previous section, the role of self attention modules are analysed. In this how effective a model can implemented for constituent parser using only the dataset which is the sentence and the labelled syntax trees. The representation of 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 summation of all the vectors that represent each character in the word. Secondly for each and word in a sentence is represented 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 7.2 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 a 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. This validates the consensus that character embeddings performs better in most NLP applications than using word embeddings. (Try to add citations???)
- Involving both character and word embeddings to represent input performs better than using them individually. A performance improvement of 0.42% F1 score on 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 test dataset does not appear in training dataset. In case of word embeddings all unknown words are represented using a unique <UNK> vector where as 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. Inorder to add credibility to it, it is made sure that words that appear only in test dataset but not in training dataset is 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 opportunity to the system to learn better.

7.3 Research Question 3

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

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 previous. The following table shows the performance of constituent parsing when these layer inputs are used separately and combined together:

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 analysed as follows:

- The first layer which does not embody the context gives a least performance. The second layer which embodies the syntactic relationship of words helps the model to achieve 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 lesser F1 score by 0.7%. This gives a clarity that constituency parser which predicts the syntactic structure of a sentence performs better with a layer output which embodies syntactic relations than other layered outputs.
- It is interesting that combination of all layers performed better than using only 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 scalar matrix. This scalar matrix is fine tuned by the training of model to achieve the maximum objective score.

The figure 14 reveals the influence of layers when all of them used. As expected nearly 80% of second layer output is used. Usage of only third layer gives a better F1 score than using only the first layer as given in the table 7.3 . But what is surprising is that it uses only 0.02% of 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.

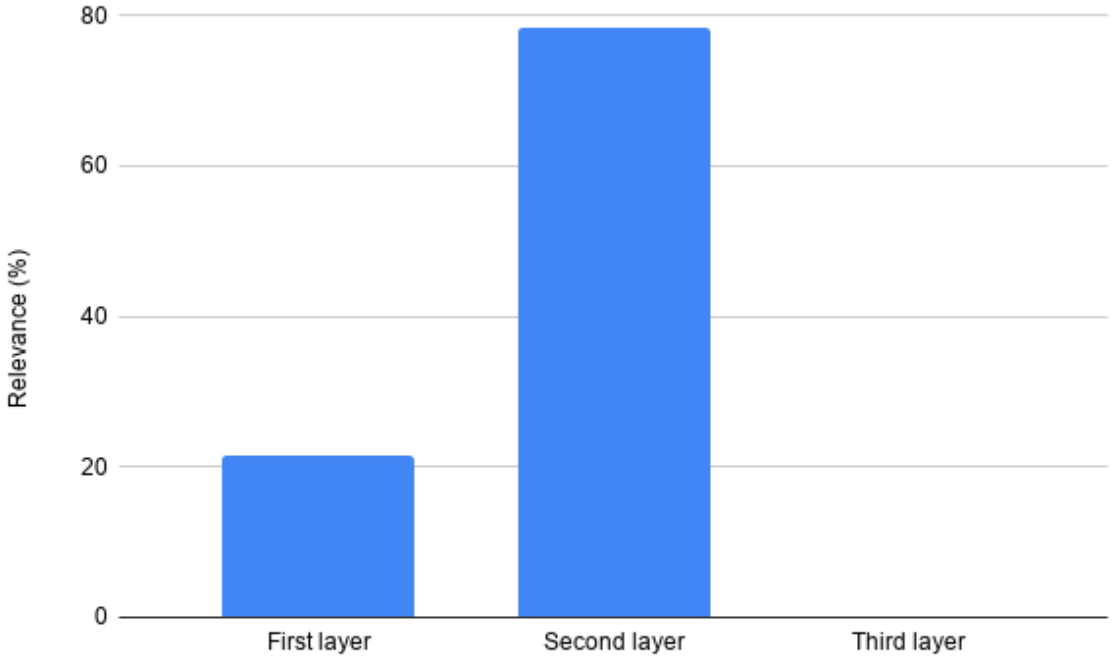

Figure 14: Importance of ELMo layers in Constituent parsing

8 Conclusion and Future work

- Self attention - no. of identical layers and no. of attention heads
- char encoding
- char encoding + word embeddings
- elmo embeddings
- Apply BERT

References

[1] Richard Socher et al. *Parsing with Compositional Vector Grammars*. Tech. rep., pp. 455–465.

[2] Daniel Jurafsky and James H Martin. "Speech and language processing: An introduction to speech recognition". In: *Computational Linguistics and Natural Language Processing.2nd Edn., Prentice Hall, ISBN* 10.0131873210 (2008), pp. 794–800.

[3] Daniel Gildea and Martha Palmer. "The Necessity of Parsing for Predicate Argument Recognition". In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. ACL '02. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 239–246. DOI: 10.3115/1073083.1073124. URL: https://doi.org/10.3115/1073083.1073124.

[4] Chris Callison-Burch. "Syntactic constraints on paraphrases extracted from parallel corpora". In: October (2010), p. 196. DOI: 10.3115/1613715.1613743.

[5] Mitchell P Marcus. "J93-2004.pdf". In: (1993).

[6] Ann Taylor, Mitchell Marcus, and Beatrice Santorini. "The Penn Treebank: An Overview". In: (2003), pp. 5–22. DOI: 10.1007/978-94-010-0201-1_1.

[7] David Gaddy, Mitchell Stern, and Dan Klein. "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)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2018, pp. 999–1010. DOI: 10.18653/v1/N18-1091. URL: http://aclweb.org/anthology/N18-1091.

[8] Richard Socher. "Parsing Natural scenes and natural language with recursive neural networks". In: 2011. ISBN: 9781450306195. DOI: 10.1007/s10107-018-1337-6. arXiv: arXiv:1207.6324.

[9] Richard Socher, Christopher D Manning, and Andrew Y Ng. *Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks*. Tech. rep.

[10] Mitchell Stern, Jacob Andreas, and Dan Klein. "A Minimal Span-Based Neural Constituency Parser". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2017, pp. 818–827. DOI: 10.18653/v1/P17-1076. URL: http://aclweb.org/anthology/P17-1076.

[11] Nikita Kitaev and Dan Klein. "Constituency Parsing with a Self-Attentive Encoder". In: Association for Computational Linguistics (ACL), 2019, pp. 2676–2686. DOI: 10.18653/v1/p18-1249.

[12] Ashish Vaswani et al. "Attention Is All You Need". In: (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.