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

January 5, 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:

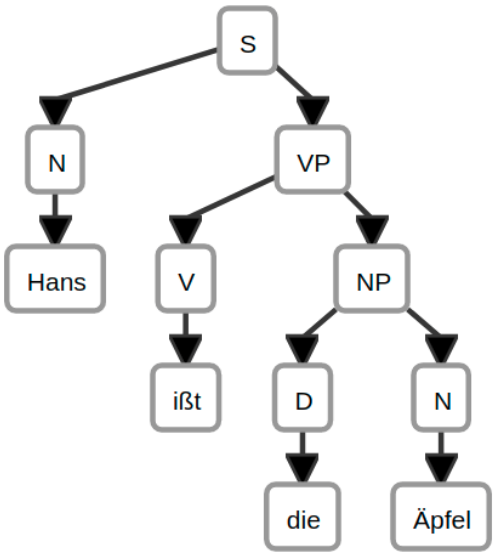

Figure 1: An example parser tree

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:

S –> N VP
VP –> V NP
NP –> D 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 percent 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 85.7percent 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 hand-written 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 percent 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(n2) is as good as using bottom up approach where in a global optimized tree is chosen with run time complexity O(n3). 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 percent.

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.

Papers to write about:

Parser showdown at the wallstreet corral: An empirical investigation of error types

Parsing three german treebanks: Lexicalized and Unlexicalized Baselines

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.

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.

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

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

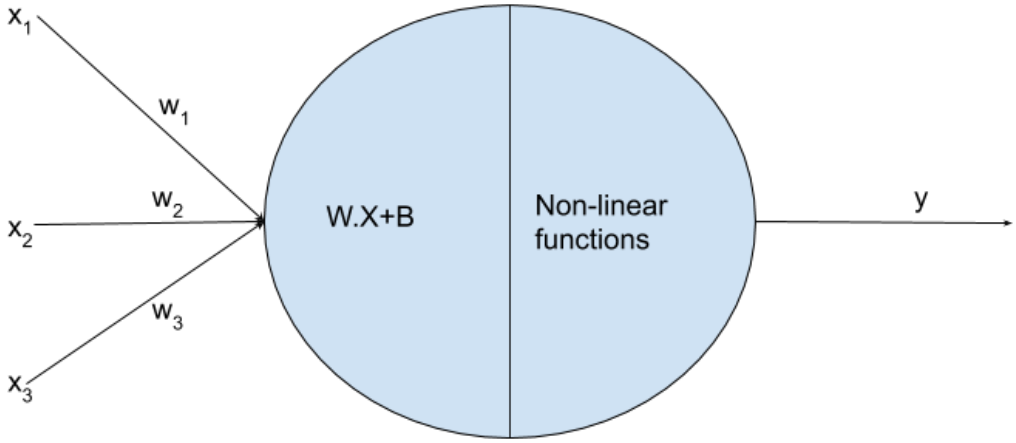

Figure 2: Single Neuron

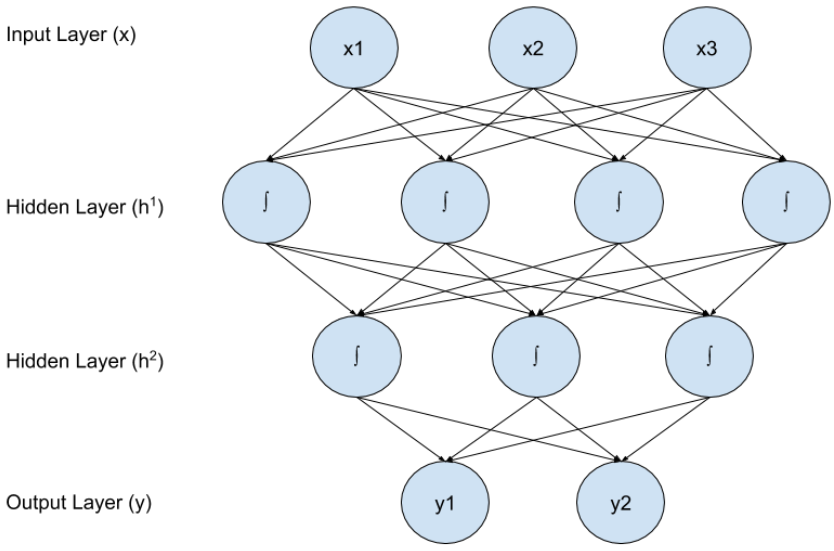

Figure 3: A Feed-Forward neural network.

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.

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

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

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

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

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.

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

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

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.

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

3.2 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 $y = y^1, y^2, \dots, y^n$ 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 $\hat{y}$.

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

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.

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

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 Neural networks for text data

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.

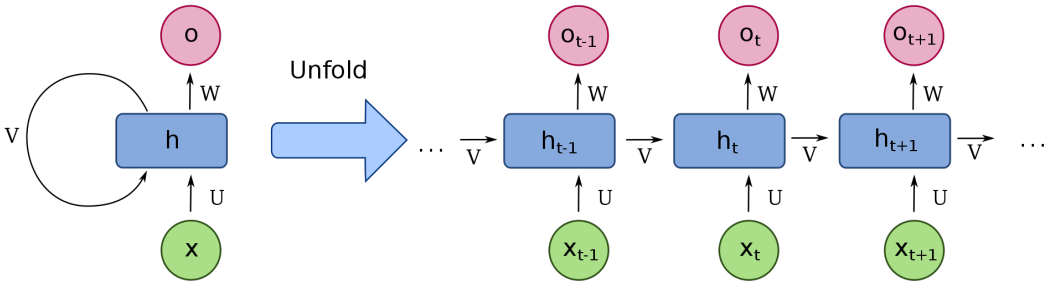

Figure 4: A basic example of RNN architecture [**WikipediaEN_RNN_unfold**].

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

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

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**]. 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.

3.4 Neural networks for parsing

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**].

3.5 Self Attention

4 Approach

In this thesis, we develop a neural network based constituent parser for german dataset released by Tübingen.

4.1 Research Questions

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

1. How well the self attention modules encode sentence which can be used to parse sentences of Tübingen dataset effectively.

2. An analysis on what features the techinique captures very well and what it does not?

4.2 Model

Model - general architecture Different kinds of context Represent input + context detailed model architecture (type of RNN, type of attention mechanism)

A general architecture of the proposed system is as shown in figure 7

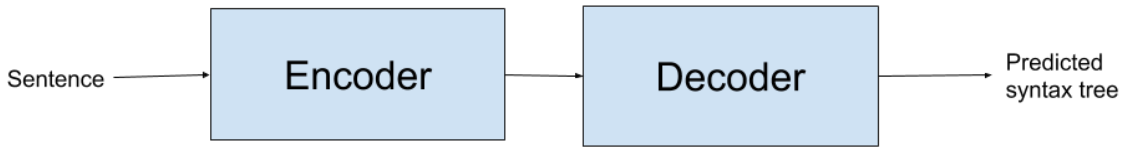

Figure 5: Broad Architecture

The reason behind using context input is that the RNN system would be better equipped with information to make a decision whether it is a true or fake news. And also that, the problem of fake news classification can be seen as a Natural Language Inference [**Parikh2016**] problem, which is about finding entailment or contradictory relationship between two sentences such as hypothesis and premise. Here, the context input is the premise and sentence input is the hypothesis. If there is an entailment between sentence input and context then it is true news otherwise false.

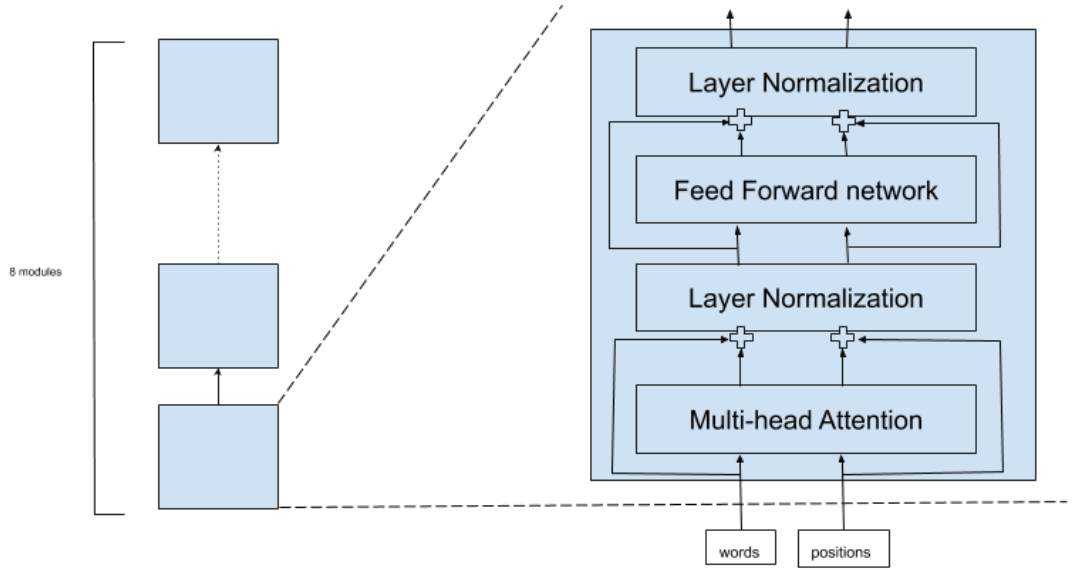

Figure 6: Broad Architecture

There are many ways by which context input can be modeled and inputted to the systems. Extracting the context as relevant to sentence input can be challenging and a suitable information retrieval method will be selected for it.

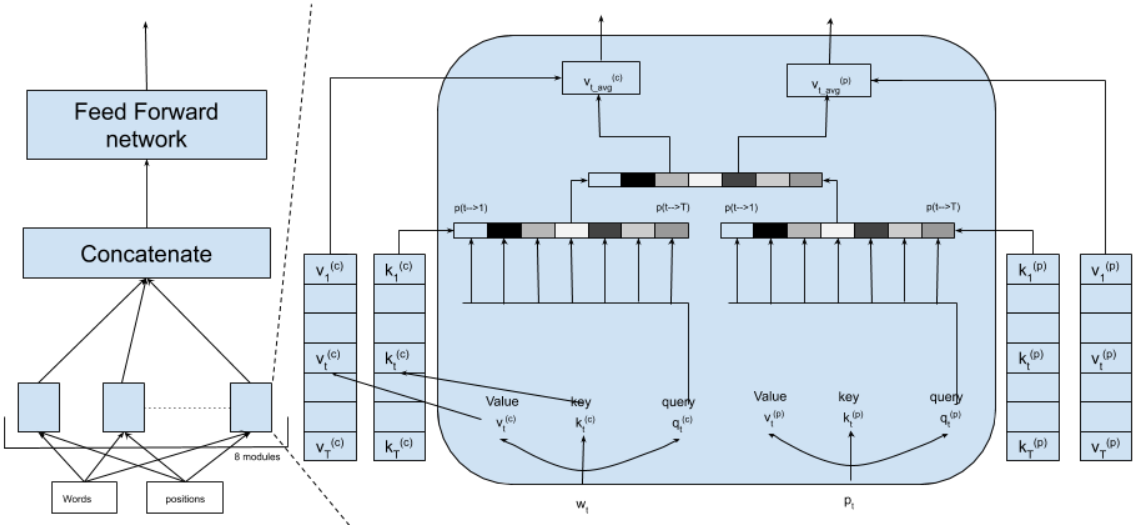

Figure 7: Broad Architecture

1. The context can be empty. The system would be trained to understand Wikipedia and use this information alone to make a decision.

2. Top 10 Wikipedia articles relevant to the sentence will be passed as context input. The Wikipedia articles will be long and so will increase the size of context input. The system may face long distance dependency issues and a suitable hyper-parameter selection is critical. Since both the sentence and context input are text from Wikipedia, the system might find it easy to align one with respect to other and find patterns among them.

3. Nearly 100 triples of knowledge graph will be passed as context input. Resource Description Framework (RDF) triples of DBPedia can be used. Extracting such information from DBPedia will be relatively easier. The structure of context input will be simple and straightforward and the length of context input will be shorter than using Wikipedia articles.

4. Information such as parts of speech, named entity and dependency graph of a sentence will be passed as context input. These metadata will bring more clarity about the sentence structure, which will help the system to extract patterns efficiently.

A detailed architecture of an advanced RNN system is shown in the figure 8. The sentence input and context input will be encoded into a rich and convenient representation separately. Bidirectional RNNs can be used which will take a sequential input and convert it into a matrix. The number of rows of the matrix will be equal to number of tokens in a sentence and number of columns can be controlled. Each row in the matrix will represent a token with sentential context. Both the sentence and context representation are then combined to form an efficient common representation. The arbitrariness of both sentence and context representation will be controlled and converted into a representation of fixed length. Attention modules can be used which will figure out how much attention needs to be given to both the representations to produce a common representation. It usually takes two matrices

and outputs a fixed vector. But the attention mechanism can be a pure reduction process, where it takes only one matrix and converts into a fixed vector. This can be used when the context input is empty. The predictor takes in a fixed vector as input and predicts which one of the binary class label(true or fake) is most likely the final output. A feed-forward neural network will be a good choice to be used as a predictor.

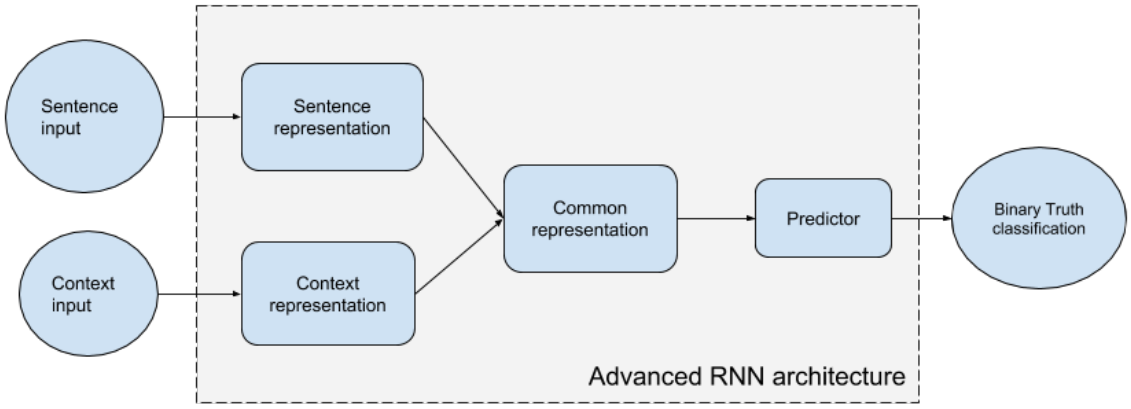

Figure 8: Detailed Architecture

4.3 Experiments

Experiments : 4 experiments Data wikipedia : sentence from wikipedia + negative examples through distortion Data fake news : kaggle - dataset or Ciampaglia dataset

1) Training - 90 percent data wikipedia, Testing - 10 data from wikipedia 2) 100 percent Data wikipedia, Data fake news 3) 90 percent Data fake news, 10 percent data fake news 4) Data from wikipedia and fake news, data fake news

Since we use Wikipedia as a proxy for authentic information, each sentence in the Wikipedia is considered as true news. There is hardly any false news present. For training to be successful to build a good fake news detector, it should be ensured that more or less equal amount of data should be present for both true and fake news. The construction of false news is tricky and different natural language processing (NLP) methods will be used to generate false news. Spacy[1], a python library for NLP tasks will be used for extracting parts of speech, named entity and dependency relations. Some techniques are

1. Swap the words randomly in each and every sentence extracted from Wikipedia. The RNN models used will remember the sequence information and so a randomly shuffled sentence will be considered as a different sentence by the system. However, the syntax and semantics of the sentence created will be poor. This method is primitive but it will serve as a good baseline.

2. Replace the random words picked in a sentence with a random word from the corpus. This method would retain some of the words as it is and changing only few words will bring a subtle change in sentence. This is similar to random swapping but the syntactic structure of the sentence will be preserved to some extent.

[1]https://www.spacy.io

3. Replace all or randomly picked words in a sentence based on their parts of speech. In this method, all the words in the corpus will be tagged with a parts of speech label. This information will be used and randomly a word will be picked from the corpus which has the same parts of speech as the word to be replaced. This method will preserve the syntactic structure of the sentence better than the previous methods.

4. Replace the words in each sentence with their opposites extracted from WordNet[2]. This method will not only preserve the syntax but also likely to have right semantics. The sentence created will more likely to have a meaning which is opposite of the original sentence.

5. Extract the dependency tree for each sentence and select an appropriate one from other sentences to replace it. The dependency relationship will contain information about which words are dependent on which other words. Replacing based on this similarity may create better examples.

Total 80 experiments. 4 ways to create a model. 5 ways to create negative examples. and 4 ways of experiments

4.4 Implementation

Implementation: Tensorflow, spacy

Recurrent neural networks will be used in this master thesis extensively since they perform well with sequential data. TensorFlow[3] is a python library for building intensive numerical computation applications and deploy them in multiple platforms such as CPU and GPU. TensorFlow has built-in libraries to develop different neural network architectures. Wikipedia consists of a lot of articles; each article consists of a lot of sentences. Each sentence will be fed as input to the neural network. ———— An advanced RNN architecture takes in either one or two inputs such as sentence in Wikipedia and context information. The context can be empty. The system processes the input and produces a binary output representing true or fake news. The outputs would contain a probability distribution of how likely it is true or fake news.

The inputs, in general, will be pre-processed to extract tokens, remove common words such as articles, prepositions and perform lemmatization which is converting the words to its root form. Low dimensional and dense word embeddings will be used to represent the inputs instead of one hot vectors. Already available word embeddings such as Word2vec [**Mikolov2013**] and GloVe[4] will be used to represent the inputs.

The dataset will be split and used for training, validation, and testing of the system. A general rule of thumb is to divide the dataset into 60-20-20 for training, validation, and testing respectively. But if the datasets are in order of millions then it is good enough to have approximately 10000 entries each for validation and testing. The validation dataset is used to try out different hyper-parameter configurations and select the optimal values for each. The hyper-parameters of the advanced neural network system that will be configured are number of layers, learning parameter, batch size, activation functions and epoch number(number of times the training dataset should be iterated).

[2]https://wordnet.princeton.edu/
[3]https://www.tensorflow.org/
[4]https://nlp.stanford.edu/projects/glove/

The proposed methods involve modeling the content of Wikipedia in different ways such that the system can understand it better. These procedures are completely automatic in building the training examples and none of the features will be manually crafted. The different configurations of the system along with unique methods in constructing the training data is expected to understand Wikipedia better and the fake news detection can be carried out in a platform agnostic manner.

5 Evaluation

The system will be evaluated as a fake news classifier. The test data should be different from the training or validation data which is used by the system.

5.1 Datasets

There are two comprehensive open datasets on fake news detection available.

1. One of it is published in Kaggle[5] which consists of structured data of 13,000 rows and 20 columns. Some of the important information in this dataset are title, text and spam score.
2. The LIAR dataset is published by Wang [**Wang2017**] which is mined from politifact.com and it consists of 12,836 short statements. In addition to that, it also contains information about context, labels with 6 classes and justification for the label classification.

The mentioned datasets might be too diverse to compare the performance of the system at the beginning. It might require training from the whole of Wikipedia. Research work from Ciampaglia et al. [**Ciampaglia2015**] used many simple datasets to evaluate network analysis techniques on a knowledge graph to build fake news classfier. The datasets are US politicians and their party affiliation, directors and their movies, US president and their spouses, US states and their capitals and world countries and their capitals. It also involves a dataset from Google Relation Extraction Corpus (GREC) which is about education degrees and institutional affiliations of people. These datasets will help to compare the performance of fake news classifier in specific domains of knowledge against their achieved results.

In addition to that, the system can be evaluated based on how well it is able to understand Wikipedia. We use the presence and absence of a information in Wikipedia as a proxy for labeling that information as true or fake news. So, it is good to evaluate our system based on how well it is able to classify whether a given information is present in Wikipedia or not. This is done by inputting either random sentences or the sentences taken from Wikipedia and check whether it is present or not. If the system performs very well in this test and fails to perform well with the mentioned standard datasets for fake news classification then it might help to gain insights on how good the coverage of news in Wikipedia is.

5.2 Metrics

The baseline system involves only RNN and the improvements that will be added are RNN with LSTM blocks, bidirectional RNNs, and inclusion of context input along with sentence input. The following factors are used to compare the baseline with the improved system

[5]https://www.kaggle.com/mrisdal/fake-news

1. How good the system is able to classify the non-Wikipedia news / fake news?
2. How fast the model is build up?

The quality of the system can be measured by metrics such as Precision, Recall, F1 measure, and Accuracy. The F1 measure is preferred since it includes both false positive and false negatives. In this master thesis, the focus is mainly on improving the accuracy of the system and the speed of the system can be improved by using good hardware.

6 Organizational matters

Duration of work:	01-Aug-2019 – 31-Jan-2019
Candidate:	Kandhasamy Rajasekaran
E-Mail:	kandhasamy@uni-koblenz.de
Student number:	216100855
Primary supervisor:	Prof. Dr. Karin Harbusch
Supervisor:	Prof. Dr. Karin Harbusch
Secondary supervisor:	Denis Memmesheimer

7 Time schedule

- Introduction and Literature: 01-May-2018 – 30-June-2018
- Initial phase: 01-Aug-2018 – 15-Oct-2018
  - Prototyping: 01-Aug-2018 – 30-Aug-2018
  - ML pipeline implementation: 01-Sep-2018 – 15-Sep-2018
  - Baseline implementation: 15-Sep-2018 – 30-Sep-2018
  - Testing and refining: 01-Oct-2018 – 15-Oct-2018
- Development phase: 16-Oct-2018 – 30-Dec-2018
  - RNN with different configuration: 16-Oct-2018 – 30-Oct-2018
  - Comprehend benchmark results: 16-Oct-2018 – 30-Oct-2018
  - Analyse and figure out improvements: 22-Oct-2018 – 30-Oct-2018
  - Improvisation using NLP techniques: 01-Nov-2018 – 30-Nov-2018
  - Attention mechanism implementation: 01-Dec-2018 – 30-Dec-2018
  - Comprehend benchmark results: 16-Dec-2018 – 30-Dec-2018
- Final phase: 01-Jan-2019 – 01-Feb-2019
  - Comprehend Benchamark results: 01-Jan-2019 – 07-Jan-2019
  - Revision: 08-Jan-2019 – 22-Jan-2019
  - Thesis report: 01-Jan-2019 – 30-Jan-2019

A meeting with Lukas Schmelzeisen will happen approximately once in two weeks to discuss about the progress made and set the targets and milestones for subsequent weeks. fgdfg

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.

8 Signatures

_____

Kandhasamy Rajasekaran

_____ _____

Denis Memmesheimer Prof. Dr. Karin Harbusch

9 Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here.

This paper was not previously presented to another examination board and has not been published.

Koblenz, on January 5, 2020

_____

Kandhasamy Rajasekaran