Self Attended Stack Pointer Networks for Learning Long Term Dependencies

Salih Tuç
Hacettepe University
Department of Computer Engineering
Ankara, Turkey
salihtuc0@gmail.com

Burcu Can
University of Wolverhampton
Research Institute of
Information and Language Processing
Wolverhampton, UK
b.can@wlv.ac.uk

Abstract

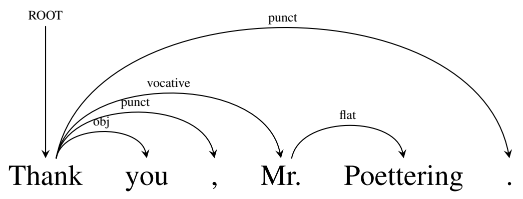
We propose a novel deep neural architecture for dependency parsing, which is built upon a Transformer Encoder (Vaswani et al., 2017) and a Stack Pointer Network (Ma et al., 2018). We first encode each sentence using a Transformer Network and then the dependency graph is generated by a Stack Pointer Network by selecting the head of each word in the sentence through a head selection process. We evaluate our model on Turkish and English treebanks. The results show that our trasformer-based model learns long term dependencies efficiently compared to sequential models such as recurrent neural networks. Our self attended stack pointer network improves UAS score around 6% upon the LSTM based stack pointer (Ma et al., 2018) for Turkish sentences with a length of more than 20 words.

1 Introduction

Dependency Parsing is the task of finding the grammatical structure of a sentence by identifying syntactic and semantic relationships between words. Dependency parsing has been utilized in many other NLP tasks such as machine translation (Carreras and Collins, 2009; Chen et al., 2017), relation extraction (Fundel-Clemens et al., 2007; Zhang et al., 2018), named entity recognition (Jie et al., 2017; Finkel and Manning, 2009), information extraction (Angeli et al., 2015; Peng et al., 2017), all of which involve natural language understanding to an extent. Each dependency relation is identified between a head word and a dependent word that modifies the head word in a sentence. Although such relations are considered syntactic, they are naturally built upon semantic relationships between words. For example, each dependent has a role in modifying its head word, which is a result of a semantic influence.

Within the context of dependency parsing, relations between heads and dependents are also labeled by specifying the type of the grammatical relation between words. In the Universal Dependencies (de Marneffe et al., 2014) tagset, there are 37 dependency relation types defined. In the latest Universal Dependencies (UD v2.0) tagset, relations are split into four main categories (Core Arguments, Non-core dependents, Nominal dependents and Other) and nine sub-categories (Nominals, Clauses, Modifier Words, Function Words, Coordination, MWE, Loose Special and Other).

One way to illustrate the grammatical structure obtained from dependency parsing is a dependency graph. An example dependency graph is given below:



Here, the relations are illustrated by the links from head words to dependent words along with their dependency labels. Every sentence has a global head word, which is the *ROOT* of the sentence.

There are two main difficulties in dependency parsing. One is the long term dependencies in especially long sentences that are difficult to be identified in a standard Recurrent Neural Network due to the loss of the information flow in long sequences. Another difficulty in parsing is the out-of-vocabulary (OOV) words. In this work, we try to tackle these two problems by using Transformer Networks (Vaswani et al., 2017) by introducing subword information for OOV words in especially morphologically rich languages such as Turkish. For that purpose, we integrate character-level word embeddings obtained from Convolutional Neural Networks (CNNs). The morphological complexity in such agglutinative languages makes the parsing task even harder because of the sparsity problem due to the number of suffixes that each word can take, which brings more problems in syntactic parsing. Dependencies in such languages were also defined between morphemic units (i.e. inflectional groups) rather than word tokens (Eryiğit et al., 2008), however this is not in the scope of this work.

In this work, we introduce a novel two-level deep neural architecture for graph-based dependency parsing. Graph-based dependency parsers build dependency trees among all possible trees, therefore the final dependency tree has the highest score globally. However, in transition-based dependency parsers, each linear selection in a sentence is made based on a local score which may lead to erroneous trees at the end of parsing. For this reason, we prefer graph-based dependency parsing in our approach to be able to do global selections while building dependency trees. In the first level of our deep neural architecture, we encode each sentence through a transformer network (Vaswani et al., 2017), which shows superior performance in long sequences compared to standard recurrent neural networks (RNNs). In the second level, we decode the dependencies between heads and dependents using a Stack Pointer Network (Ma et al., 2018), which is extended with an internal stack based on pointer networks (Vinyals et al., 2015). Since stack pointer networks benefit from the full sequence similar to self attention mechanism in transformer networks, they do not have left-to-right restriction as in transition based parsing. Hence, we combine the two networks to have a more accurate and efficient dependency parser.

We evaluate our model on Turkish which is a morphologically rich language and on English with a comparably poorer morphological structure. Although our model does not outperform other recent model, it shows competitive performance among other neural dependency parsers. However, our results show that our self attended stack pointer network improves UAS score around 6% upon the LSTM based stack pointer (Ma et al., 2018) for Turkish sentences with a length of more than 20 words.

The paper is organized as follows: Section 2 reviews the related work on both graph-based and transition-based dependency parsing, Section 3 explains the dependency parsing task briefly, Section 4 describes the proposed deep neural architecture based on Transformer Networks and Stack Pointer Networks, and finally Section 5 presents the experimental results of the proposed model for both English and Turkish.

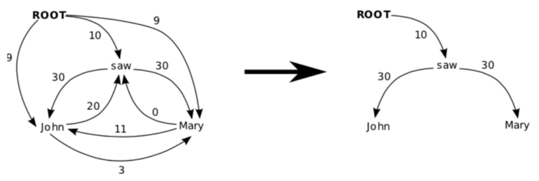


Figure 1: An example to graph-based dependency parsing with a maximum spanning tree.

2 Related Work

Dependency parsing is performed by two different approaches: graph-based and transition-based parsing. We review related work on both of these approaches.

Graph-based Dependency Parsing: Graph-based approaches are generally based on performing the entire parsing process as graph operations where the nodes in the graph represent the words in a sentence. For the sentence, *"John saw Mary"*, we can illustrate its parse tree with a weighted graph G with four vertices where each of them refers to a word including the $ROOT$. Edges store the dependency scores between the words. The main idea here is to find the maximum spanning tree of this graph G. The parse tree of the sentence is given in Figure 1. The dependencies are between $ROOT$ and saw, saw and $John$; and saw and $Mary$ where the first ones are the heads and the latter ones are the dependents.

When the parsing structure is represented as a graph, finding dependencies becomes easier to visualize, and moreover the task becomes finding the highest scored tree among all possible trees. Edge scores in the graphs represent the dependency measures between word couples.

Neural architectures have been used for graph-based dependency parsing extensively in the last decade. Li et al. (2018) introduce a seq2seq model using bi-directional LSTMs (BiLSTMs) (Hochreiter and Schmidhuber, 1997), where an attention mechanism is involved between the encoder and decoder LSTMs. Kiperwasser and Goldberg (2016) propose another model using BiLSTMs, where the right and left arcs in the dependency trees are identified through the BiLSTMs. Dozat and Manning (2016) proposes a parser that uses biaffine attention mechanism, which is extended based on the models of Kiperwasser and Goldberg (2016), Hashimoto et al. (2017), and Cheng et al. (2016). The biaffine parser (Dozat and Manning, 2016) provides a baseline for other two models introduced by Zhou and Zhao (2019) and Li et al. (2019), which forms trees in the form of Head-Driven Phase Structure Grammar (HPSG) and uses self-attention mechanism respectively. Ji et al. (2019) propose a Graph Neural Network (GNN) that is improved upon the biaffine model. Another LSTM-based model is introduced by Choe and Charniak (2016), where dependency parsing is considered as part of language modelling (LM) and each sentence is parsed with a LSTM-LM architecture which builds parse trees simultaneously with the language model.

The recent works generally focus on the encoder in seq2seq models because a better encoding of an input eliminates most of the cons of the sequence models. For example, Hewitt and Manning (2019) and Tai et al. (2015) aim to improve the LSTM-based encoders while Clark et al. (2018) introduce an attention-based approach to improve encoding, where they propose Cross-View Training (CVT).

In this work, we encode each sentence through a transformer network based on self-attention mechanism (Vaswani et al., 2017) and learn the head of each word using a stack pointer network as a decoder (Ma et al., 2018) in our deep neural architecture. Our main aim is to learn long term dependencies efficiently with a transformer network by removing the recurrent structures from encoder. Transformer networks (Vaswani et al., 2017) and stack pointer networks (Ma et al., 2018) have been used for dependency parsing before. However, this will be the first attempt to combine these two methods for the dependency parsing task.

Transition-based Dependency Parsing: In transition-based dependency parsing, local selections are made for each dependency relationship without considering the complete dependency tree. Therefore, globally motivated selections are normally not performed in transition-based parsing by contrast with graph-based dependency parsing. For this purpose, two stacks are employed to keep track of the actions made during transition-based parsing.

Similar to graph-based parsing, neural approaches have been used extensively for transition-based parsing. Chen and Manning (2014) introduce a feed forward neural network with various extensions by utilizing single-word, word-pair and three-word features. Weiss et al. (2015) improve upon the model by Chen and Manning (2014) with a deeper neural network and with a more structured training and inference using structured perceptron with beam-search decoding. Andor et al. (2016) use also feed forward neural networks similar to others and argue that feed forward neural networks outperform RNNs in case of a global normalization rather than local normalizations as in Chen and Manning (2014), which apply greedy parsing.

Mohammadshahi and Henderson (2019) utilize a transformer network, in which graph features are employed as input and output embeddings to learn graph relations, thereby their novel model, Graph2Graph transformer, is introduced.

Fernández-González and Gómez-Rodríguez (2019) propose a transition-based algorithm that is similar to the stack pointer model by Ma et al. (2018); however, left-to-right parsing is adopted on the contrary to Ma et al. (2018), where top-down parsing is performed. Hence, each parse tree is built in n actions for an n length sentence without requiring any additional data structure.

In addition to these models, there are some works such as the greedy parser of Ballesteros et al. (2016) and Kuncoro et al. (2016), and the high-performance parser by Qi and Manning (2017).

Nivre and McDonald (2008) indicate that graph-based and transition-based parsers can be also combined by integrating their features. And several works follow this idea (Goldberg and Elhadad, 2010; Spitkovsky et al., 2010; Ma et al., 2013; Ballesteros and Bohnet, 2014; Zhang and Clark, 2008).

3 The Formal Definition of Dependency Parsing

Dependency parsing is the task of inferring the grammatical structure of a sentence by identifying the relationships between words. Dependency is a head-dependent relation between words and each *dependent* is affected by its *head*. The dependencies in a dependency tree are always from the head to the dependents.

The parsing, no matter which approach is used, creates a dependency tree or a graph, as we mentioned above. There are some formal conditions of this graph:

- Graph should be connected.
  - Each word must have a head.

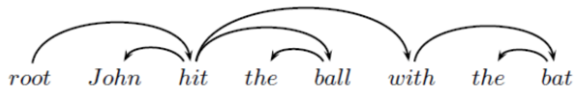


Figure 2: An example projective tree

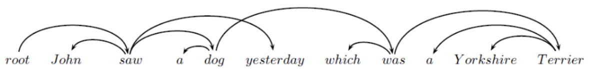


Figure 3: An example non-projective tree

- Graph must be acyclic.
  - If there are dependencies $w1 \rightarrow w2$ and $w2 \rightarrow w3$; there must not be a dependency such as $w3 \rightarrow w1$.
- Each of the vertices must have one incoming edge.
  - Each word must only have one head. A graph that includes $w1 \rightarrow w2$ and $w3 \rightarrow w2$ is not allowed in a dependency graph.

A dependency tree is projective if there are no crossing edges on the dependency graph. Figure 2 illustrates a projective tree and Figure 3 illustrates a non-projective dependency graph.

4 Dependency Parsing with Self Attended Stack Pointer Network

4.1 Overview

Self Attended Stack Pointer Network is extended on a standard Stack Pointer Network (STACKPTR) (Ma et al., 2018) along with a self attention mechanism. In STACKPTR, input word embeddings are processed via a BiLSTM-CNN encoder, where a BiLSTM is utilized to encode each word and a CNN is utilized to learn character-based encoding of each word. All words are stored in a stack structure and each encoded word on the top of the stack is decoded using an LSTM decoder to discover their heads by utilizing high-order information such as siblings and grandparents. Finally, each dependency relation is predicted through a Deep BiAffine Parser Dozat and Manning (2016) in a standard Pointer Network architecture.

Our model deviates from the STACKPTR model with a transformer network that encodes each word with a self-attention mechanism, which will allow to learn long-term dependencies since every word's relation to all words in a sentence can be effectively processed in a transformer network on the contrary to recurrent neural networks. In sequential recurrent structures such as RNNs or LSTMs, every word's encoding contains information about only previous words in a sentence and there is always a loss in the information flow through the long sequences in those structures.

In our transformer network, we adopt a multi-head attention and a feed-forward network. Once we encode a sequence with a transformer network, we decode the sequence to predict the head of each word in that sequence by using a stack pointer network.

4.2 Transformer Encoder

In RNNs, each state is informed by the previous states with a sequential information flow through the states. However, in longer sequences, information passed from earlier states loses its effect on the later states in RNNs by definition. Transformer networks are effective attention-based neural network architectures (Vaswani et al., 2017). The main idea is to replace the recurrent networks with a single transformer network which has the ability to compute the relationships between all words in a sequence with a self-attention mechanism without requiring any recurrent structure. Therefore, each word in a sequence will be informed by all other words in the sequence.

Learning long term dependencies in especially long sentences is still one of the challenges in dependency parsing. We employ transformer networks in order to tackle with the long term dependencies problem by eliminating the usage of recurrent neural networks while encoding each sentence during parsing. Hence, we use transformer network as an encoder to encode each word by feeding our transformer encoder with each word's pretrained word embeddings (Glove (Pennington et al., 2014) or Polyglot (Al-Rfou' et al., 2013) embeddings), part-of-speech (PoS) tag embeddings, character-level word embeddings obtained from CNN, and the positional encodings of each word.

Positional encoding (PE) is used to inject positional information for each encoded word, since there is not a sequential recurrent structure in a self attention mechanism. With the positional encoding, some relative or absolute positions of words in a sentence are utilized. The cos function is used for the odd indices and the sin function is used for even indices. The injection of the position information is performed with the sinus waves. The

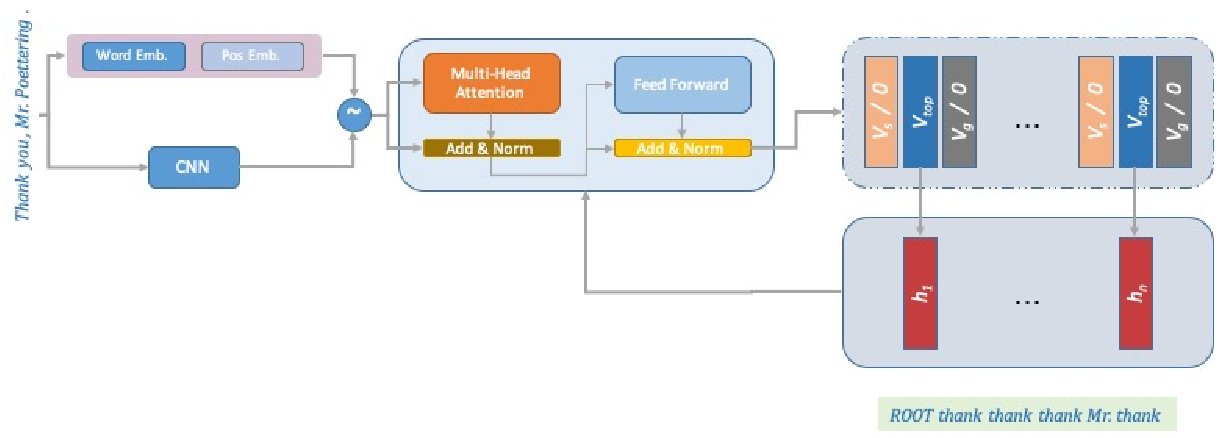


Figure 4: Overview of the Self-Attended Pointer Network Model. After concatenating word embeddings, POS tag embeddings, and char-embeddings obtained from CNN, the final embedding is fed into the self-attention encoder stack. Then, embedding of the word at the top of the stack, its sibling and grandparent vectors are summed-up in order to predict the dependency head.

sin function for the even indices is computed as follows:

$$PE(x, 2i) = \sin\left(\frac{x}{10000^{2i/d_{model}}}\right) \quad (1)$$

where d_{model} is the dimension of the word embeddings, $i \in [0, d_{model}/2)$, and x is the position of each word where $x \in [0, n]$ in the input sequence $s = (w_0, w_1 \dots w_n)$. The cos function for the odd indices is computed analogously.

The positional encoding is calculated for each embedding and they are summed. So the dimension d_{model} does not change. Concatenation is also possible theoretically. However, in the input and output embeddings, the position information is included in the first few indices in the embedding. Thus, when the d_{model} is large enough, there is no need to concatenate. The summation also meets the requirements.

The Encoder stack contains a Multi-Head Attention and a Feed-Forward Network. A Layer Normalization is applied after each of these two layers. There could be more than one encoder in the encoder stack. In this case, all of the outputs in one encoder is fed into the next encoder in the encoder stack. In our model, we performed several experiments with different number of encoder layers in the encoder stack to optimize the number of encoder layers for parsing.

Multi-Head Attention is evolved from Self-Attention Mechanism, which enables encoding all words using all of the words in the sentence. So it learns better relations between words compared to recurrent structures. The all-to-all encoding in self-attention mechanism is performed through query, key and value matrices. There are multiple sets of queries, keys and values that are learned in the model. Self-attention is calculated for each of these sets and a new embedding is produced. The new embeddings for each set are concatenated and multiplied with Z matrix which is a randomly-initialized matrix in order to compute the final embeddings. Z matrix is trained jointly and multiplied with the concatenated weight matrix in order to reduce the embeddings into a single final embedding for each set. In other words, the final embedding is learnt from different contexts at the same time. It is multi-head because it learns from the *head* of each set. The *head* of each set is calculated by using self-attention.

Finally, a Feed Forward Neural Network which is basically a neural network with two linear layers and ReLU activation function is used to process the embeddings obtained from multi-head attention. It is placed at the end of the encoder because with this feed-forward neural network, we can train the embeddings with a latent space of words.

Layer Normalization (Ba et al., 2016) is applied to normalize the weights and retain some form of information from the previous layers, which is performed for both Multi-Head Attention and Feed Forward Neural Network.

Final output embeddings contain contextual information about the input sentence and the words in the sequence. So, the output of the Transformer Encoder is a -theoretically- more comprehensive representation of contextual information compared to the input word embeddings and also compared to the the output of a BiLSTM encoder

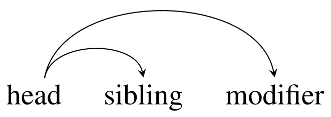


Figure 5: Sibling structure

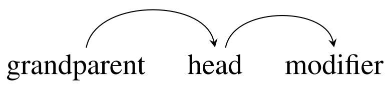


Figure 6: Grandchild structure

of a STACKPTR.

4.3 Stack Pointer Network

Stack Pointer Network (STACKPTR) (Ma et al., 2018) is a transition-based structure but it still performs a global optimization over the potential dependency parse trees of a sentence. STACKPTR is based on a pointer network (PTR-NET) (Vinyals et al., 2015) but differently, a STACKPTR has a stack to store the order of head words in trees. In each step, an arc is built from a child to the head word at the top of the stack based on the attention scores obtained from a pointer network.

We use a Stack Pointer Network for decoding the sequence to infer the dependencies, where each word is encoded with a Transformer Network as mentioned in the previous section.

The transformer encoder outputs a hidden state vector s_i for the ith word in the sequence. The hidden state vector is summed with higher-order information similar to that of Ma et al. (2018). There are two types of higher-order information in the model: Sibling (two words that have the same parent) and grandparent/grandchild (parent of the word's parent and the child of the word's child). Figure 5 and Figure 6 shows an illustration of these high-order structures.

So, the input vector for the decoder is the sum of the state vector of the word on the top of the stack, its sibling and its grandparent:

$$\beta_i = s_h + s_s + s_g \tag{2}$$

In the decoder part, an LSTM gathers all of the contextual and higher-order information about the word at the top of stack. Normally, in the pointer networks, at each time step t, the decoder receives the input from the last step and outputs decoder hidden state h_t. Therefore, an attention score is obtained as follows:

$$e_i^t = score(h_t, s_i) \tag{3}$$

where e^t is the output of the scoring function, s_i is the encoder hidden state and h_t is the decoder hidden state at time step t. After calculating the score for each possible output in the Biaffine attention mechanism, the final prediction is performed as follows with a softmax function to convert it into a probability distribution:

$$a^t = softmax(e^t) \tag{4}$$

where a^t is the output probability vector for each possible child word and e^t is the output vector of the scoring function.

In our model, scoring function is adopted from Deep Biaffine attention mechanism (Dozat and Manning, 2016):

$$e_i^t = h_t^T W s_i + U^t h_t + V^t s_i + b \tag{5}$$

where W is the weight matrix, U and V are the weight vectors and b is the bias.

Additionally, before the scoring function, an MLP is applied to the output of decoder, as proposed by Dozat and Manning (2016) to reduce the dimensionality.

As for the dependency labels, we also use another MLP to reduce the dimensionlity and then apply deep biaffine to score the possible labels for the word at the top of the stack.

4.4 Learning

We use cross-entropy loss for training the model similar to STACKPTR. The probability of a parse tree y for a given sentence x under the parameter set θ is $P_\theta(y|x)$ and estimated as follows:

$$\begin{aligned} P_\theta(y|x) &= \prod_{i=1}^{k} P_\theta(p_i|p_{<i}, x) && (6) \\ &= \prod_{i=1}^{k}\prod_{j=1}^{l_i} P_\theta(c_{i,j}|c_{i,<j}, p_{<i}, x) && (7) \end{aligned}$$

$p_{<i}$ denotes the preceding paths that have already been generated, $c_{i,j}$ represents the j^{th} word in the path p_i and $c_{i,<j}$ denotes all the proceeding words on the path p_i. Here, a path consists of a sequence of words from the root to the leaf.

The model learns the arcs and labels in the dependency tree simultaneously.

5 Experiments & Results

5.1 Datasets

We ran experiments on both Turkish and English. We used Penn Treebank (PTB) (Marcus et al., 1993) for English and IMST dataset (Sulubacak et al., 2016) in Universal Dependencies for Turkish.

As for the word embeddings, we used pre-trained Glove embeddings (Pennington et al., 2014) on Wikipedia and pre-trained Polyglot embeddings (Al-Rfou' et al., 2013) on Wikipedia for both Turkish and English.

5.2 Evaluation Metrics

For the evaluation, we used two different evaluation metrics: UAS and LAS, which are the standard metrics for dependency parsing.

UAS is a metric that is used to calculate the accuracy of predicting words' heads. In other words, it is the ratio of the number of correctly predicted heads to the total number of words in the dataset:

$$\frac{UAS = \#ofcorrectheads}{\#ofwords} \tag{8}$$

LAS is another metric for dependency parsing that measures the correctness of both heads and labels. In other words, it is the ratio of correctly predicted heads **and** labels to the total number of words in the dataset:

$$\frac{LAS = \#ofcorrecthead, labelpair}{\#ofwords} \tag{9}$$

5.3 Hyperparameters

In our experiments, we use similar configurations with the baseline models: StackPtr model by Ma et al. (2018) and Self-Attention mechanism by Vaswani et al. (2017). Differently from the baseline models, for the self-attended encoder stack; we used 6 layers because this configuration performs better with the Polyglot embeddings for both English and Turkish as seen in Table 1 and Table 2 for English and Turkish respectively.

5.4 Results

The results obtained from IMST dataset (Sulubacak et al., 2016) in Turkish is given in Table 3, along with the results of other related work. OUr results compared to other related work show competitive performance for Turkish language. Our model gives an UAS score of 74.43% and LAS score of 64.26% with Glove embeddings, whereas

Layer	UAS
1	86.24
2	88.56
4	92.40
6	**94.23**
8	93.13

Table 1: Accuracy for different number of encoder layers for PTB Dataset (Marcus et al., 1993)

Layer	UAS
1	69.89
2	71.48
4	74.51
6	**76.81**
8	75.32

Table 2: Accuracy for different number of encoder layers for Turkish IMST Dataset (Sulubacak et al., 2016)

an UAS score of 76.81% and LAS score of 67.95% are obtained with Polyglot embeddings. Therefore, using Polyglot embeddings gives far better results in Turkish. This could be due to the size of the train set used for the Polyglot embeddings.

The results obtained from Penn Treebank dataset (Marcus et al., 1993) in English is given in Table 4. Our results again show competitive performance compared to other related work for English. Similar to the Turkish results, our model performs better with Polyglot embeddings. While Glove gives 93.43% UAS and 91.98% LAS, Polyglot gives 94.23% UAS and 92.67% LAS.

5.5 Error Analysis

5.5.1 Sentence Length

The main aim in this study is to utilize Transformer Networks to resolve the long-term dependencies problem in dependency parsing. We analyzed the accuracy of our model in both short and longer sentences to see the impact of the Transformer Networks in our model compared to sequential StackPtr model that is based on LSTMs.

Table 7 gives the results for different lengths of sentences to show the impact of using Transformer Networks in long term depedencies. We compare our model with the original StackPtr (Ma et al., 2018) model, which is based on LSTMs. As the results show, our model performs far better for sentences with more than 20 words compared to the standard StackPtr model, with an improvement

Model	UAS	LAS
Our Model w/ Glove	74.43	64.26
Our Model w/ Polyglot	76.81	67.95
Nguyen and Verspoor (2018)	70.53	62.55
Kondratyuk and Straka (2019)	74.56	67.44
McDonald et al. (2006)	74.70	63.20
Dozat and Manning (2016)	77.46	68.02
Ma et al. (2018)	79.56	68.93
Ballesteros et al. (2015)	79.30	69.28

Table 3: Results for Turkish IMST Dataset (Sulubacak et al., 2016)

Model	UAS	LAS
Our Model w/ Glove	93.43	91.98
Our Model w/ Polyglot	94.23	92.67
Ballesteros et al. (2015)	91.63	89.44
Chen and Manning (2014)	91.8	89.6
Kiperwasser and Goldberg (2016)	93.1	91.0
Ballesteros et al. (2016)	93.56	91.42
Weiss et al. (2015)	94.26	92.41
Andor et al. (2016)	94.61	92.79
Ma and Hovy (2017)	94.88	92.98
Dozat and Manning (2016)	95.74	94.08
Ma et al. (2018)	95.87	94.19

Table 4: Results for English PTB Dataset (Marcus et al., 1993)

of UAS score with around 7%.

For less than 20 words, our model's accuracy is lower compared to longer sentences. It shows that our self-attention based model is not able to learn shorter sentences better than the BiLSTM based StackPtr model. However, we observed that decreasing the number of layers in our encoder stack gives a higher accuracy for shorter sentences. However, it decreases the overall accuracy for the entire dataset.

5.5.2 The Impact of Punctuation

We also analyzed the impact of using punctuation in the datasets during training. Analysis of Spitkovsky et al. (Spitkovsky et al., 2011) shows that the usage of lexicalized and punctuated sentences gives better results in dependency parsing. So, we ran our model with both punctuated and not-punctuated versions of both datasets in Turkish and English. Table 5 shows that punctuation affects the learning of the model for both languages

Dataset	**w/ Punctuation**	**w/o Punctuation**
PTB	94.23 (92.67)	93.47 (91.94)
IMST	76.81 (67.95)	71.96 (62.41)

Table 5: Accuracy (UAS (LAS)) with and without punctuation on IMST (Sulubacak et al., 2016) and PTB (Marcus et al., 1993) Datasets

Input Embeddings	UAS
Glove	63.24
Polyglot	65.76
Polyglot + PoS	70.48
Polyglot + CNN	73.81
Polyglot + PoS + CNN	76.81

Table 6: The impact of using word embeddings (Glove or Polyglot), PoS tag embeddings and character-based word embeddings for the Turkish IMST Dataset (Sulubacak et al., 2016)

and the results are comparably higher when the punctuation is also used in the datasets. The impact of using punctuation is even more for Turkish language and both UAS and LAS are around %5 higher compared to training on datasets without punctuation.

5.5.3 The Impact of Using Embeddings

We analyzed the effect of using various embeddings in the Transformer encoder. As mentioned before, we utilize word embeddings, PoS tag embeddings and char embeddings obtained from CNN in our model. Table 6 shows the impact of the embeddings on the accuracy of the model. As the results show, character-level encoding plays a crucial role in our model because it helps to mitigate the OOV problem during training. We obtained the highest scores when Polyglot word embeddings, PoS tag embeddings and character-based word embeddings are incorporated in training.

6 Conclusion & Future Work

Our experiments show that using Self-Attention mechanism increases parsing accuracy especially in longer sentences in Turkish. However, our parser requires more data to learn better for also shorter sentences. The results also show that using character level word embeddings along with word embeddings and PoS tag embeddings gives the highest accuracy for our model.

We obtained the highest scores when we include

Number of words in sentence	UAS - StackPtr	UAS - Self-Attended StackPtr
less than 10 words	93.23	86.47
between 10 and 20 words	88.96	81.63
more than 20 words	56.49	62.33

Table 7: Accuracies for different lengths of sentences in IMST Dataset in Turkish (Sulubacak et al., 2016)

6 layers in our encoder stack by using Polyglot embeddings. Our results also show that including punctuation in the dataset improves the accuracy substantially.

We leave integrating morpheme-level information in especially morphologically rich languages such as Turkish as future work.

References

Rami Al-Rfou', Bryan Perozzi, and Steven Skiena. 2013. Polyglot: Distributed word representations for multilingual NLP. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 183–192, Sofia, Bulgaria. Association for Computational Linguistics.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany. Association for Computational Linguistics.

Gabor Angeli, Melvin Jose Johnson Premkumar, and Christopher D. Manning. 2015. Leveraging linguistic structure for open domain information extraction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 344–354, Beijing, China. Association for Computational Linguistics.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Miguel Ballesteros and Bernd Bohnet. 2014. Automatic feature selection for agenda-based dependency parsing. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 794–805, Dublin, Ireland. Dublin City University and Association for Computational Linguistics.

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with lstms.

Miguel Ballesteros, Yoav Goldberg, Chris Dyer, and Noah A. Smith. 2016. Training with exploration improves a greedy stack-lstm parser.

Xavier Carreras and Michael Collins. 2009. Non-projective parsing for statistical machine translation. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 200–209, Singapore. Association for Computational Linguistics.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 740–750, Doha, Qatar. Association for Computational Linguistics.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. 2017. Improved neural machine translation with a syntax-aware encoder and decoder. pages 1936–1945.

Hao Cheng, Hao Fang, Xiaodong He, Jianfeng Gao, and Li Deng. 2016. Bi-directional attention with agreement for dependency parsing. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2204–2214, Austin, Texas. Association for Computational Linguistics.

Do Kook Choe and Eugene Charniak. 2016. Parsing as language modeling. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2331–2336, Austin, Texas. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc V. Le. 2018. Semi-supervised sequence modeling with cross-view training.

Timothy Dozat and Christopher D. Manning. 2016. Deep biaffine attention for neural dependency parsing.

Gülşen Eryiğit, Joakim Nivre, and Kemal Oflazer. 2008. Dependency parsing of turkish. *Computational Linguistics*, 34(3):357–389.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2019. Left-to-right dependency parsing with pointer networks. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 710–716, Minneapolis, Minnesota. Association for Computational Linguistics.

Jenny Rose Finkel and Christopher D. Manning. 2009. Joint parsing and named entity recognition. In *Proceedings of Human Language Technologies: The*

2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pages 326–334, Boulder, Colorado. Association for Computational Linguistics.

Katrin Fundel-Clemens, Robert Küffner, and Ralf Zimmer. 2007. Relex - relation extraction using dependency parse trees. *Bioinformatics (Oxford, England)*, 23:365–71.

Yoav Goldberg and Michael Elhadad. 2010. An efficient algorithm for easy-first non-directional dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 742–750, Los Angeles, California. Association for Computational Linguistics.

Kazuma Hashimoto, Caiming Xiong, Yoshimasa Tsuruoka, and Richard Socher. 2017. A joint many-task model: Growing a neural network for multiple NLP tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1923–1933, Copenhagen, Denmark. Association for Computational Linguistics.

John Hewitt and Christopher D. Manning. 2019. A structural probe for finding syntax in word representations. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4129–4138, Minneapolis, Minnesota. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9:1735–80.

Tao Ji, Yuanbin Wu, and Man Lan. 2019. Graph-based dependency parsing with graph neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2475–2485, Florence, Italy. Association for Computational Linguistics.

Zhanming Jie, Aldrian Obaja Muis, and Wei Lu. 2017. Efficient dependency-guided named entity recognition. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 3457–3465. AAAI Press.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Dan Kondratyuk and Milan Straka. 2019. 75 languages, 1 model: Parsing universal dependencies universally.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an ensemble of greedy dependency parsers into one mst parser.

Ying Li, Zhenghua Li, Min Zhang, Rui Wang, Sheng Li, and Luo Si. 2019. Self-attentive biaffine dependency parsing. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, pages 5067–5073. AAAI Press.

Zuchao Li, Jiaxun Cai, Shexia He, and Hai Zhao. 2018. Seq2seq dependency parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3203–3214, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Ji Ma, Jingbo Zhu, Tong Xiao, and Nan Yang. 2013. Easy-first POS tagging and dependency parsing with beam search. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 110–114, Sofia, Bulgaria. Association for Computational Linguistics.

Xuezhe Ma and Eduard Hovy. 2017. Neural probabilistic model for non-projective MST parsing. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 59–69, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414, Melbourne, Australia. Association for Computational Linguistics.

Mitchell Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of english: The penn treebank.

Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. 2014. Universal Stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, pages 4585–4592, Reykjavik, Iceland. European Languages Resources Association (ELRA).

Ryan McDonald, Kevin Lerman, and Fernando Pereira. 2006. Multilingual dependency analysis with a two-stage discriminative parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 216–220, New York City. Association for Computational Linguistics.

Alireza Mohammadshahi and James Henderson. 2019. Graph-to-graph transformer for transition-based dependency parsing.

Dat Quoc Nguyen and Karin Verspoor. 2018. An improved neural network model for joint. *Proceedings of the*.

Joakim Nivre and Ryan McDonald. 2008. Integrating graph-based and transition-based dependency parsers. In *Proceedings of ACL-08: HLT*, pages 950–958, Columbus, Ohio. Association for Computational Linguistics.

Nanyun Peng, Hoifung Poon, Chris Quirk, Kristina Toutanova, and Wen-tau Yih. 2017. Cross-sentence n-ary relation extraction with graph lstms. *Transactions of the Association for Computational Linguistics*, 5.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Peng Qi and Christopher D. Manning. 2017. Arc-swift: A novel transition system for dependency parsing.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2010. From baby steps to leapfrog: How "less is more" in unsupervised dependency parsing. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 751–759, Los Angeles, California. Association for Computational Linguistics.

Valentin I. Spitkovsky, Hiyan Alshawi, and Daniel Jurafsky. 2011. Punctuation: Making a point in unsupervised dependency parsing. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 19–28, Portland, Oregon, USA. Association for Computational Linguistics.

Umut Sulubacak, Memduh Gokirmak, Francis Tyers, Çağrı Çöltekin, Joakim Nivre, and Gülşen Eryiğit. 2016. Universal dependencies for Turkish. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 3444–3454, Osaka, Japan. The COLING 2016 Organizing Committee.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. 2015. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, Beijing, China. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. 2015. Structured training for neural network transition-based parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 323–333, Beijing, China. Association for Computational Linguistics.

Yue Zhang and Stephen Clark. 2008. A tale of two parsers: Investigating and combining graph-based and transition-based dependency parsing. In *Proceedings of the 2008 Conference on Empirical Methods in Natural Language Processing*, pages 562–571, Honolulu, Hawaii. Association for Computational Linguistics.

Yuhao Zhang, Peng Qi, and Christopher D. Manning. 2018. Graph convolution over pruned dependency trees improves relation extraction. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2205–2215, Brussels, Belgium. Association for Computational Linguistics.

Junru Zhou and Hai Zhao. 2019. Head-driven phrase structure grammar parsing on Penn treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2396–2408, Florence, Italy. Association for Computational Linguistics.