UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Liisa Rätsep

Generative Dependency Language Modeling Using Recurrent Neural Networks

Master's Thesis (30 ECTS)

Supervisor: Kairit Sirts, PhD

Tartu 2019

Generative Dependency Language Modeling Using Recurrent Neural Networks

Abstract: This thesis proposes an approach to incorporating syntactical data to the task of generative language modeling. We modify the logic of a transition-based dependency parser to generate new words to the buffer using the top items in the stack as input. We hypothesize that the approach provides benefits in modeling long-term dependencies. We implement our system along with a baseline language model and observe that our approach provides an improvement in perplexity scores and that this improvement is more significant in modeling sentences that contain longer dependencies. Additionally, the qualitative analysis of the generated sentences demonstrates that our model is able to generate more cohesive sentences.

Keywords: generative language modeling, recurrent neural networks, natural language processing, dependency parsing

CERCS: P176 Artificial intelligence

Süntaktilisi sõltuvusi kasutav generatiivne keele modelleerimine rekurrentsete tehisnärvivõrkudega

Lühikokkuvõte: Käesolev magistritöö esitleb meetodit süntaktilise infot kasutamiseks generatiivses keele modelleerimises, kus sõltuvusparseri loogikat laiendatakse, et jooksvalt parseri puhvrisse uusi sõnu genereerida. Selleks kasutatakse sisendina vastaval hetkel pinu tipus olevaid sõnu. Püstitame hüpoteesi, et antud lahendus annab eeliseid kaugete sõltuvuste modelleerimisel. Me implementeerime pakutud keelemudeli ja lähtemudeli ning näeme, et välja pakutud meetod annab märkimisväärselt parema *perplexity* skoori tulemuse ja seda eriti lausete puhul, mis sisaldavad kaugeid sõltuvusi. Lisaks näitab keelemudelite abil loodud lausete analüüs, et välja pakutud mudel suudab lähtemudeliga võrreldes luua terviklikumaid lauseid.

Võtmesõnad: generatiivne keele modelleerimine, rekurrentsed tehisnärvivõrgud, loomuliku keele töötlus, sõltuvusparsimine

CERCS: P176 Tehisintellekt

Contents

1 Introduction

A generative language model is a crucial part of natural language processing applications that contain text generation functionality, for example machine translation systems or conversational interfaces. As humans, we are able to understand and create structurally complex sentences that include long-term dependencies between words, but this may pose a challenge for language models. Before the recent emergence in the popularity of using neural networks in natural language processing tasks, statistical n-gram language models were very limited in modeling longer sequences and thus unaware of long-term dependencies.

In the past 20 years, there have been various research efforts to solve the problem of modeling long-term dependencies by using syntactically annotated corpora and modeling sentences as their dependency structures instead of word sequences [4, 30, 11]. The aforementioned corpora are primarily used for dependency parsing - the task of automatically inferring the syntactic dependency tree for a sentence.

With the increasing popularity of neural language models, the issue of modeling long-term dependencies has been addressed by using different recurrent neural network structures such as long short-term memory (LSTM) units [12] which are designed for modeling long-term dependencies. However, as Linzen et al. [18] have shown, even recurrent neural networks are not able to fully capture long-term dependencies and therefore it is still relevant to find effective ways to improve language modeling and one option is to take advantage of syntactically annotated corpora.

In this thesis we propose a novel approach for training language models on syntactically annotated data. The proposed approach is inspired by the logic of transition-based dependency parsing applications, the BIST parser [15] in particular, and extends the dependency parser to fit the task of language modeling. We compare our approach with other relevant research efforts and implement our proposed language model alongside a simple LSTM language model baseline to evaluate its potential benefits.

We are particularly interested to see whether we can notice any advantages in modeling long-term dependencies. We hope that thanks to the transition-based dependency parser logic which will be incorporated into our model, we are able to eliminate irrelevant words from our context of predicting upcoming words.

We find that our solution is able to outperform the baseline model and that this improvement is more significant when modeling sentences with long-term dependencies.

Additionally, when comparing the sentences generated by each model, we can also observe advantages over the baseline model.

1.1 Structure of the Thesis

This section presents an overview of how this thesis is structured. The thesis contains the following chapters:

- Chapter 1 provides an overview of this thesis.
- Chapter 2 gives a theoretical overview of the concepts and methodologies that are relevant to our proposed solution.
- Chapter 3 describes our proposed approach of transition-based language modeling and the design of our model.
- Chapter 4 provides an overview of other relevant research on this topic and how it relates to our solution.
- Chapter 5 describes our implementation of the proposed solution as well as the baseline solution together with all preprocessing steps and parameters.
- Chapter 6 presents our results and contains their analysis.
- Chapter 7 summarizes the thesis with a conclusion section which also provides suggestions for future research on this topic.

2 Background

This section provides a technical overview of neural networks, language modeling and dependency parsing.

2.1 Neural Networks

Artificial neural networks are computational systems which mimic the biological brain and nervous system. The following overview is based on the tutorial by Yoav Goldberg [9].

Feedforward Neural Networks

Similarly to a biological brain that consists of neurons or nerve cells, a neural network is also made up of cells. These computational units can have multiple inputs, each with its own weight parameter, and the output of each cell can be fed into other cells as inputs. Inside the cell, each input is multiplied by its weight, then these values are summed and applied a non-linear activation function.

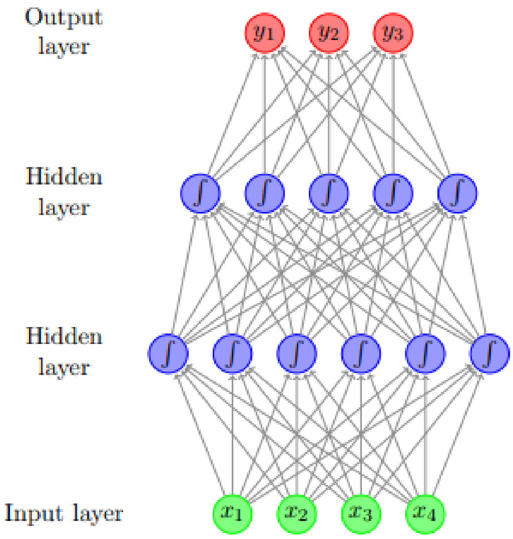


Figure 1. Scheme of a fully connected feed-forward neural network with two hidden layers. Each circle represents a cell and each arrow has been assigned a weight. The $\int$ symbol inside the cells refers to the sigmoid activation function $(1/(1 + e^{-x})$. The image is taken from Goldberg [9].

Feed-forward neural networks, as illustrated on Figure 1, consist of an input layer, an output layer and a number of hidden layers between them. Networks which have more than one hidden layer are considered deep neural networks. In case all cells in one layer are connected to all the cells in the next one, the layer is considered to be fully connected. In addition, a layer can have a bias term, which is an additional constant added before the activation function is applied. This can be achieved by adding an additional input cell to the layer, which has no inputs and has a value of 1.

The simplest feed-forward neural network is the perceptron:

$$NN_{Perceptron}(x) = xW + b$$

In this linear function, x is the input vector, W is the weight matrix and b is the bias term. This can be expanded by adding hidden layers to achieve a multi-layer perceptron (MLP):

$$\begin{aligned} NN_{MLP_n}(x) =& y \\ h^1 =& g^1(xW^1 + b^1) \\ & \dots \\ h^n =& g^2(h^{n-1}W^n + b^n) \\ y =& h^n W^{n+1} \end{aligned}$$

In this MLP, n is the number of layers, h is the output of the respective layer, g is the layer's activation function and y is the final output.

Training Neural Networks

The weights, biases and possibly other matrices of a neural network are collectively referred to as the parameters of the network. In order to achieve the desired output, these randomly initialized network parameters must be tuned. Other variables, such as layer dimensions, number of layers, etc. are collectively referred to as hyperparameters and these are set before the model is trained.

The first step of the training process is forward propagation, where numerous samples are passed through the network. The outputs are then compared to the expected results using a loss function which illustrates the differences between the two. The loss value is 0 when the output is correct and positive in all other cases.

The forward propagation is followed by backpropagation [31] where the errors are propagated backwards through the network. An optimization method is then used to tune the parameters to minimize the loss value. This process is repeated until the optimal model parameters are achieved.

Deep neural networks can experience the vanishing or exploding gradient problem. This refers to cases where the gradient is moving close to zero or increasing while backpropagating through the network. The deeper the network, the more relevant the issue becomes.

In case the trained model is able to work well with the training data but not with new and unseen data, the model has been overfitted. To monitor the performance of the model, the model can be periodically tested with a validation or development dataset and in case the accuracy on that set stops improving, training can be stopped early. The final performance of the model is measured on a third test dataset. There are also other regularization methods such as using dropout - a technique where certain nodes within the network can be randomly ignored during training to reduce overfitting [33].

Recurrent Neural Networks

Recurrent neural networks (RNNs) are a class of neural networks dedicated to modeling sequential data. This is possible thanks to their hidden memory state which is an output of an RNN cell and which is used as an input to the same cell at the next time step as illustrated on Figure 2.

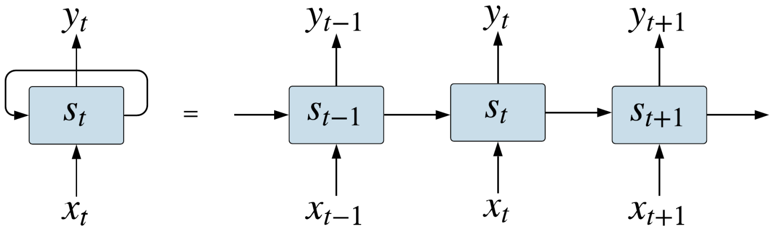


Figure 2. Scheme of a recurrent neural network with a compact representation on the left and an unfolded representation on the right. Each of the blue rectangles represents an RNN unit. At each time step t, the current hidden state is calculated using the input x_t and the previous hidden state x_{t-1} which in turn is passed to the next time step. The output at time step t is y_t.

Unlike feedforward neural networks which perform each calculation independently without any context of previous or upcoming inputs, the output of RNN cells at each time step is affected by the previous inputs via the previous hidden state. This makes RNNs the preferred method for solving tasks that analyze sequences, such as movement tracking and a variety of natural language processing tasks like language modeling, machine translation, etc.

In recurrent networks, the backpropagation is done through the previous time steps as if the unfolded graph is a single deep network. This process is called backpropagation through time [35]. This, however, makes RNNs very susceptible to the vanishing gradient problem which in turn makes RNNs inefficient for modeling long-term dependencies.

Long Short-Term Memory Networks

Long short-term memory (LSTM) [12] networks are a type of RNN designed to overcome the vanishing gradient problem for modeling data with long-term dependencies.

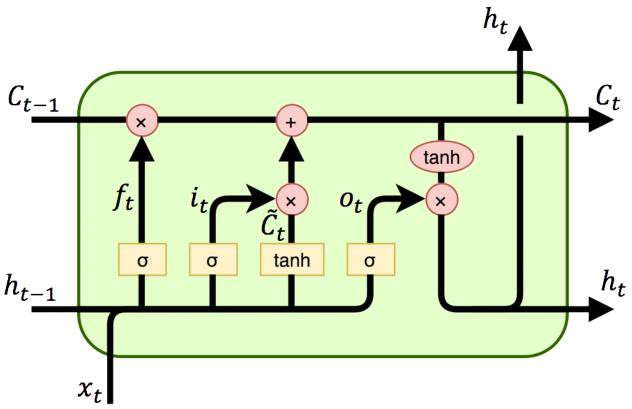


Figure 3. A scheme of an LSTM unit.[1]

An LSTM network is made up of LSTM units or cells as depicted on Figure 3. When a simple RNN unit functions as a single layer, an LSTM unit has four separate neural network layers with separate weights and biases for each. Each unit contains a forget gate, an input gate and an output gate. The cell state C_t is passed on from cell to cell and serves as the memory. The input at the current time step x_t and the output at the previous time step h_{t-1} are also passed to the cell as inputs.

The forget gate f_t is responsible for "forgetting" or "remembering" items in the cell state vector and uses a sigmoid function to do such filtering. The cell state is further

[1] The image is taken from `http://colah.github.io/posts/2015-08-Understanding-LSTMs`

modified by an input gate i_t which is responsible for modifying the cell state based on the input and previous output. The tanh function $2/(1+e^{-2x})-1$ provides new candidate values $\tilde{C}_t$ which are again filtered by a sigmoid layer and added to the cell state vector. The output gate o_t determines the output by taking the updated cell state and the input, and applying a tanh layer to the first and a sigmoid filter to latter.

Thanks to this structure with cell states and gates, the gradients can remain high even with long sequences and therefore LSTMs are particularly useful for many natural language processing tasks which deal with long sentences with long-term syntactic or other dependencies.

Bidirectional Recurrent Neural Networks

Bidirectional recurrent neural networks (BRNN) are a variant of RNNs which have two layers of RNN units that process the input data in different directions. With traditional RNNs, the output at each time step only depends on the current and previous inputs, however, no future context is considered. In a BRNN illustrated in Figure 4, the output at each time step uses two RNN units. One unit has already seen previous inputs by processing the input in a forward order and the other one has seen future inputs by processing the input in a reversed order. The outputs of these RNN units are afterwards

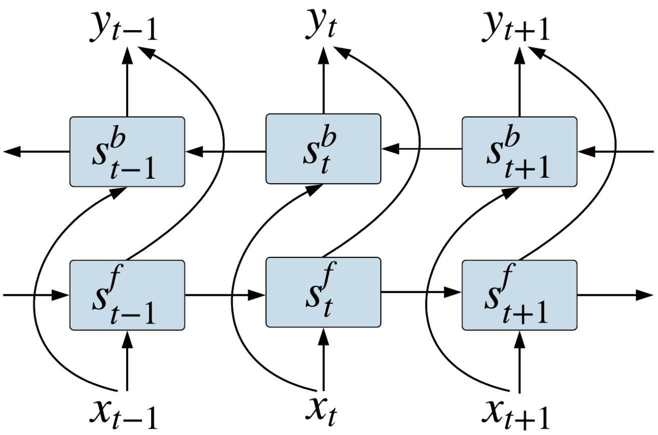


Figure 4. A bidirectional neural network unfolded where at each time step, the forward RNN unit is passed the current input x_t and the state of the previous RNN s^f_{t-1} and the backward RNN unit is passed the current input x_t and the state of the previous RNN s^b_{t+1} which has processed the next input x_{t+1}

concatenated into a single output. This allows the model to use the context of both past and future inputs and their potential dependencies. The bidirectional method is not limited to simple RNNs but can be applied to all types of RNN units, including LSTMs to form a bidirectional LSTM network.

BRNNs are therefore a popular choice for modeling data which includes dependencies in both directions and where future events are not only conditioned by the previous input. Natural language processing is once again a good example of a potential use case. However, it should be noted that BRNNs are not suitable for processing real-time data streams where the future inputs are not known at the time of processing.

2.2 Language Modeling

Language modeling is the task of assigning probabilities to tokens (words and punctuation marks) or character sequences, for example to sentences or to words occurring after a given sequence. The following overview on language modeling is based on Jurafsky and Martin [13].

There are two main types of language modeling implementations. Discriminative language modeling is used only for text evaluation purposes to see how well it matches the model. Examples of different applications where such models can be used include language detection or checking which domain the text matches, for example whether it is a legal, technical or medical text. The second type of language modeling is generative and this is used in applications that deal with text creation, for example machine translation, conversational interfaces or generating automatic image captions.

The most common evaluation metric for language models is perplexity:

$$PP = 2^{\frac{1}{M}\sum_{i=1}^{m}\log_2 p(x_i)}$$

In this formula, M is the total number of words in the test set, m is the number of sentences and $p(x_i)$ is the probability of the current sentence x_i. The lower the perplexity, the better the model is able to predict upcoming words as it is the inverse of the probability which has been normalized by the number of items in the test set, which in this case is the number of words. The intuition behind this value is to illustrate the number of potential options for the upcoming word.

In addition to perplexity, extrinsic evaluation methods can be used by integrating the language model to another application which uses language modeling and comparing the models by evaluating the metric by which that application is evaluated.

Statistical Language Modeling

Statistical language modeling is one of the most common approaches of language modeling. With this approach, the probability of the upcoming word x_t at the current time step t is determined by how many times it has appeared in the context of the words that preceded it:

$$P(x_t|x_{t-1},...,x_0) = \frac{C(x_t, x_{t-1},...,x_0)}{C(x_{t-1},...,x_0)}$$

As words can occur in different and unique contexts, counting the occurrences in the context of all the preceding words is not feasible. This can, however, be done using the Markov assumption according to which the approximate probability of the future state is only dependent on the current state and not the past [20]:

$$P(x_t|x_{t-1},...,x_0) \approx P(x_t|x_{t-1})$$

A language model that only uses the previous word to predict the next one is referred to as a unigram model, but this approach can be expanded to n-gram language models where the probability of the next word x_t is calculated using n previous words:

$$P(x_t|x_{t-1},...,x_0) \approx P(x_t|x_{t-1},...,x_{t-n})$$

Using n-gram models, the probability of an entire sequence with length i would be:

$$P(x_1, x_2, ..., x_i) \approx \prod_t P(x_t|x_{t-1},...,x_{t-n})$$

Such language models are able to perform relatively well on common sequences with length n or less, but are unable to take into consideration longer dependencies which exist in most languages. For example, in the following sentence, an n-gram model where $n = 3$ would not be able to predict the correct grammatical gender of the pronoun *her*:

Mary was walking home when the bus hit ***her****.*

In addition, statistical language models can run into the issue of data sparsity where all the potential word sequences do not exist in the training data and thus such sequences will have zero probability. This is because n-gram models are unable to generalize and take into account similarities between different words and therefore they disregard similar contexts. There are, however, various smoothing algorithms to alleviate the issue of data sparsity where the n-gram probabilities are modified to assign non-zero probabilities to unseen n-grams by reducing the probabilities of more frequent sequences.

Neural Language Modeling

Neural language modeling is another common approach to language modeling. These models are able to generalize to solve the issue of data sparsity. Instead of representing each word as a discrete entity, words are represented by continuous vector representations which are called embeddings. These embeddings can either be learned separately by a dedicated application or randomly initialized and tuned among other model parameters during training. Thanks to this, similar words often end up having similar embeddings. This helps the model to generalize and predict unseen sequences of words that have appeared in other contexts.

In addition, neural language models perform better in learning long-term dependencies thanks to their recurrent implementations such as LSTMs [34].

2.3 Dependency Parsing

Dependency parsing is the task of finding the syntactic dependency tree of a sentence as can be seen on Figure 5. The following overview is based on Jurafsky and Martin [13].

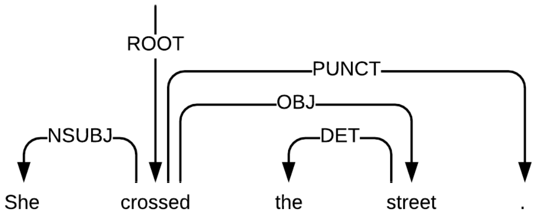


Figure 5. A projective dependency tree of a sentence, where arrows represent dependency arcs and labels on them the dependency relation types.

A dependency tree illustrates the syntactic structure of a sentence by describing the directed relations between the tokens. A dependency tree consists of a number of directed arcs and nodes which represent the tokens in the sentence and an additional root node. All arcs represent a dependent to head relationship between two nodes. There is one incoming arc per node from its head with the exception of root nodes which do not have any incoming arcs. Each of the arcs has a label that signifies the type of relation between the nodes.

Dependency trees can be either projective or non-projective. In projective trees, there is always a path from the head to all nodes between the head and the dependant. In this case the arcs in the tree never cross each other. However, in many languages, especially in ones with flexible word order, non-projective trees which do not have this constraint can occur. An example of such tree can be seen on Figure 6.

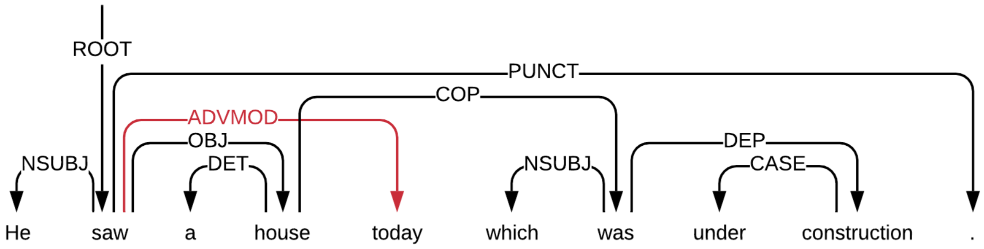


Figure 6. A non-projective dependency tree of a sentence.

Dependency parsers can be trained on treebanks which are corpora annotated with dependency information. The best-known source of such treebanks is the Universal Dependencies (UD) project [28] which has created a standard for treebank annotations that can be applied for all languages. For example, the dependency tree in Figure 5 can be represented using the revised version of the UD format, also known as the CoNLL-U format, as shown in Table 1.

ID	FORM	LEMMA	UPOS	XPOS	FEATS	HEAD	DEPREL	DEPS	MISC
1	She	she	PRON	PRP	Case=Nom|Gend...	2	nsubj	_	_
2	crossed	cross	VERB	VBD	Mood=Ind|Tense...	0	root	_	_
3	the	the	DET	DT	Definite=Def|Pro...	4	det	_	_
4	street	street	NOUN	NN	Number=Sing	2	obj	_	_
5	.	.	PUNCT	.	_	2	punct	_	_

Table 1. An annotated sentence in the CoNLL-U format.

The CoNLL-U format has ten fields for each token - its index number (ID), the token itself (FORM), its lemma (LEMMA), its universal part-of-speech tag (UPOS), a language-specific part-of-speech tag (XPOS), a list of morphological features (FEATS), the ID of its head (HEAD), the dependency relation label (DEPREL), enhanced dependency information (DEPS) and other information (MISC). There are 37 standard dependency relation labels which have been derived from the relations described by de Marneffe et al. [7]. These include *nmod* for nominal modifiers, *aux* for auxiliaries, *det* for determiners, etc. Thanks to the standardized CoNLL-U format, many parsers are trained to take advantage of part-of-speech tags or morphological feature information during parsing [15, 29].

There are two main evaluation metrics for evaluating dependency parsers. The labeled attachment score (LAS) is the accuracy of correct arcs with the correct relation type label. The unlabeled attachment score (UAS) is the accuracy of correct arcs without considering the dependency relation type.

Transition-Based Parsing

Transition-based parsing [6] is a simple yet efficient method of dependency parsing. It is an extension of shift-reduce parsing which was developed in 1972 [1] and used on programming languages. In transition-based parsing, the sentence is processed with various predefined operations that result in different configurations. A configuration c consists of a stack σ, buffer β and relations T.

The most common transition-based approaches include the arc-standard [27], arc-eager [26] and arc-hybrid [17] methods. We describe the arc-hybrid approach of transition-based parsing which has three possible operations to parse all items in the buffer. These operations can be defined as:

$$\begin{gathered}\text{SH } (\sigma, x_i|\beta, T) \Rightarrow (\sigma|x_i, \beta, T) \\ \text{LA } (\sigma|x_i, x_j|\beta, T) \Rightarrow (\sigma, x_j|\beta, T \cup \{(x_j \rightarrow x_i)\}) \\ \text{RA } (\sigma|x_i|x_j, \beta, T) \Rightarrow (\sigma|x_i, \beta, T \cup \{(x_i \rightarrow x_j)\})\end{gathered}$$

Shift (SH) will move the first item of the buffer to the stack. Left-Arc (LA) will assign the first word in the buffer as the head of the top word in the stack and remove the latter from the stack. Right-Arc (RA) will assign the second word in the stack as the head of the top word and remove the latter.

The initial transition-based parser configuration with the arc-hybrid approach can be defined as:

$$c = ([], [x_0, ..., x_n], [])$$

where x_0 refers to the root and x_n to the last word of the sentence. The parser actions are performed until the terminal condition where the buffer is empty and the stack only contains the root node:

$$c = ([x_0], [], T)$$

Stack	Buffer	Operation	Relation
	root She crossed the street .	SH	
root	She crossed the street .	SH	
root She	crossed the street .	LA	She ← crossed
root	crossed the street .	SH	
root crossed	the street .	SH	
root crossed the	street .	LA	the ← street
root crossed	street .	SH	
root crossed street	.	RA	crossed → street
root crossed	.	SH	
root crossed .		RA	crossed → .
root crossed		RA	root → crossed
root		-	

Table 2. A sentence parsed using the arc-hybrid approach.

The transition-based parsing approach is, however, is unable to produce non-projective parse trees. This can impose a constraint in languages with flexible word order.

A step-by-step example of arc-hybrid operations applied on a sentence can be seen in table 2.

Graph-Based Parsing

Another common approach to dependency parsing is graph-based parsing. With this approach, the goal is to find the highest-scoring parse tree as if it were a directed graph. Unlike transition-based parsing, graph-based parsing is able to produce non-projective trees.

The optimal graph is selected using graph theory methods (for example the maximum spanning tree algorithm) and processed further to make sure a valid parse tree is produced (for example by removing cycles).

BIST Parser

BIST parser [15] is a dependency parser implemented in Python which uses the DyNet [24] neural network toolkit. The parser uses bidirectional LSTMs (BLSTMs) and has both a transition-based as well as a graph-based implementation, however, the latter will not be regarded in this overview.

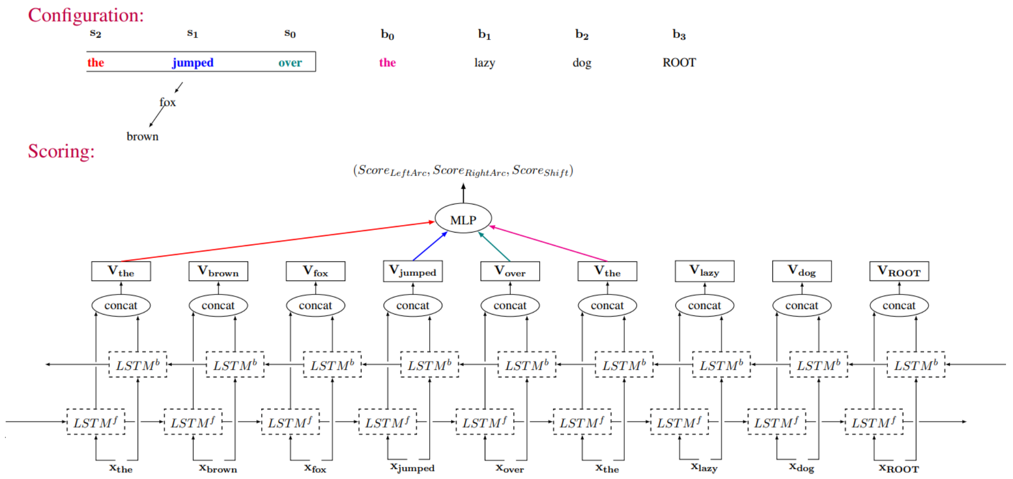


Figure 7. The scheme of the transition-based version of BIST parser. The current state of the stack and buffer are shown on the top and the neural network is on the bottom. The implementation actually uses two layers of BLSTMs even though only one is depicted. The figure has been taken from Kiperwasser and Goldberg [15].

The scheme of the arc-hybrid transition-based parser can be seen in Figure 7. The parser uses the BLSTMs to create a contextual vector representation for each token using the concatenation of its word and part-of-speech tag embeddings as input.

At each parsing step, the vectors for the top three tokens in the stack and the first word in the buffer are concatenated and scored using two multi-layer perceptrons. One MLP is used to determine the best scoring valid parsing operation and the other MLP is used to pick the best scoring dependency label.

3 Transition-based Language Model

We propose a generative dependency language model that uses the logic of an arc-hybrid transition-based dependency parser while generating new words to the buffer using the top items in the stack as input. We hope that when the parser removes words from the stack, we are also removing irrelevant words from our prediction context and improving our ability to model long-term dependencies as the tokens removed from the stack cannot be the head of any upcoming tokens that we generate. Due to its similarities to transition-based parsers, we will be referring to this solution as the transition-based language model.

Our transition-based language model (TLM) will be based largely on the existing implementation of BIST parser. The model will process word embeddings in each sentence using LSTMs to create contextual encoded vector representations for each token as can be seen in Figure 8. Unlike in BIST parser, these LSTMs will not be used bidirectionally due to the constraints of generative language modeling where the upcoming tokens are not available to the model while the current token is being processed. The word embeddings are randomly initialized lookup parameters and tuned with other model parameters during training.

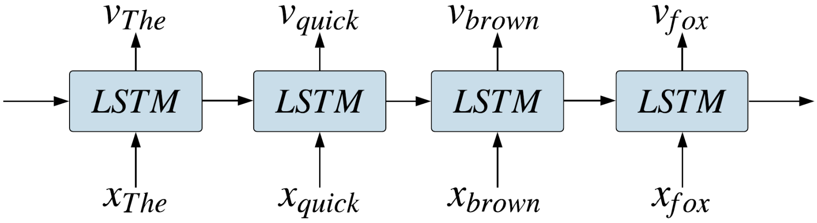
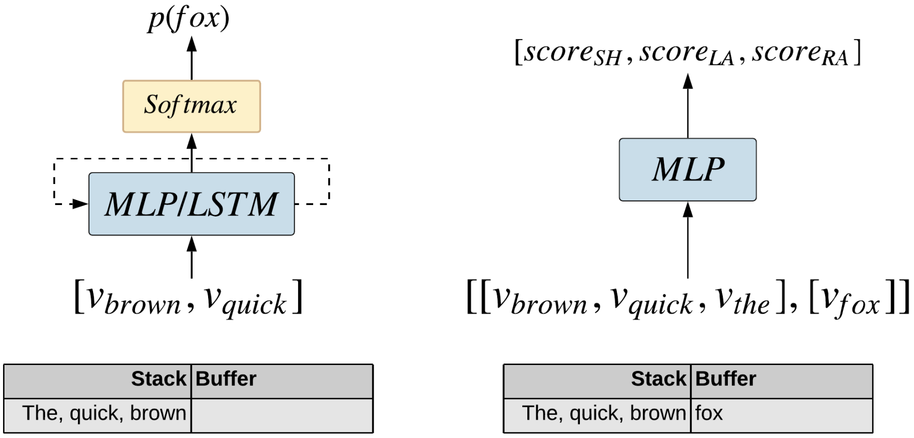


Figure 8. Illustration of how LSTMs are used in the proposed language model to transform the embedding x_i of each word i to its encoded representation v_i.

The transition-based language model contains functionality for dependency parsing and for token generation, and the neural network parts dedicated to these tasks are illustrated in Figure 9. The token generation functionality will be used whenever the stack is empty, therefore after each Shift operation, until the end-of-sentence marker (<EOS>) is generated. The latter also serves as the root node in the tree. The parser functionality is used at all other times until the terminal configuration is reached. All parts of the model are trained together.


$p(fox)$
$Softmax$
$MLP/LSTM$
$[v_{brown}, v_{quick}]$
$[score_{SH}, score_{LA}, score_{RA}]$
MLP
$[[v_{brown}, v_{quick}, v_{the}], [v_{fox}]]$

Stack	Buffer
The, quick, brown	

Stack	Buffer
The, quick, brown	fox

Figure 9. Illustration of the transition-based language model with the part of the network dedicated to generating new words on the left and the part responsible for scoring parsing actions on the right.

The parser functionality is similar to BIST parser as it takes the vectors of the top three stack items and the first item in the buffer and uses a multi-layer perceptron to opt for the highest scoring parsing action. However, when BIST parser looks for the best parsing action and the best dependency label, our implementation disregards the dependency label completely.

For the token generation part, we tested two different variations of the model. In the first approach (TLM-MLP), the generation functionality is similar to the parsing functionality and uses the top items in the stack and an MLP to predict the upcoming word. In the second approach (TLM-LSTM), an LSTM is used instead of an MLP to produce the predictions for new tokens. Therefore, with TLM-MLP, the output depends only on the current parser configuration whereas with TLM-LSTM the configurations during previous time steps also affect the token prediction. In both variants, a softmax function is applied to convert the outputs to a probability distribution over the entire vocabulary using the following formula to find the probability of each element y_i where y is the vector of scores and $y = (y_1, ..., y_K)$:

$$\frac{e^{y_i}}{\sum_{j=1}^{K} e^{y_j}}$$

It should be noted, however, that this approach limits us from implementing several techniques that are used in language modeling. For example, the tokens cannot be split into smaller units because the syntactic annotation and parser transitions can only be used on the token level. In addition, the approach is unsuitable in applications which do not deal with full sentences but shorter phrases.

4 Related Work

This section provides an overview of some of the previous research that has explored different methods of using syntactic information to improve the quality of language modeling and explains how they compare with our proposed approach.

4.1 Statistical Dependency Language Modeling

There has been a considerable amount of research emerged from the need to overcome the constraints of the n-gram model which is unable to predict words based on context beyond the n-gram's reach.

One of the first attempts of creating language models that use syntactic information date back to 1997 when Chelba [4] developed a language model that would, similarly to our proposed approach, process a sentence left to right either predicting upcoming words based on the parse or parsing existing words using the logic of a transition-based parser. The goal of this was to filter out irrelevant past words by knowing their syntactic structure. The initial experiments proved to perform better than the baseline n-gram models when comparing perplexity scores. In 2000, this idea was successfully developed further by Chelba and Jelinek [5] who applied the same type of model to a speech recognition application. This use case was also the reason why it was considered important to process the sentence left to right.

In 2010, Popel and Mareček [30] proposed a different solution for improving statistical language models where each word is predicted using its parent (head) or parent and grandparent. Gubbins and Vlachos [11] have also proposed a similar solution for sentence completion where each word is conditioned by its entire ancestor sequence.

4.2 Neural Dependency Language Modeling

Although neural language models, especially the ones using RNNs, are able to use context beyond an n-gram's reach, Linzen et al. [18] have shown that LSTMs are not able to fully capture syntax-sensitive dependencies. Therefore, using syntactic dependencies to improve language modeling has been relevant even with the increasing utilization of recurrent neural networks.

Like Gubbins and Vlachos did in 2013 with n-gram models [11], Mirowski and Vlachos [22] implemented a generative RNN language model to be used for sentence completion where words were also conditioned by their ancestor sequences and were successfully able to outperform baseline RNN language models. Similarly, Zhou et al. [38] have proposed another generative language model by implementing top-down tree generation where the tree is later flattened to form a sentence as the decoder of their conversational sentence generation application.

As LSTMs are designed for modeling sequences, Zhang et al. [37] developed a different LSTM structure called TreeLSTM for predicting trees. In their approach, the probability of each node was conditioned by its subtree. As a result, they implemented a generative language model which was tested in a sentence completion application. Similarly, Alvarez-Melis and Jaakkola [2] have also proposed a new neural network architecture to decode encoded vector representations into trees. Their implementation of a generative language model is a doubly-recurrent network which models both the relationships between head and dependant nodes and between siblings.

In 2015, Buys and Blunsom [3] proposed a feed-forward generative parser which used the logic of a transition-based arc-standard parser, but generated new words with their part-of-speech tags with every shift operation. To do this, they used the context of the top four items in the stack and the left and right dependants of the top two stack items. This implementation is the only neural language model that is somewhat similar to our proposed approach as it processes the data left-to-right and uses the logic of a transition-based parser, however, our approach uses recurrent neural networks to generate encoded vector representations for all tokens to better fit the sequential nature of linguistic data. In addition, in our approach the dependants of the top stack items are not used during token generation, because a token that has already been removed from the stack cannot be the head of any upcoming tokens and therefore we consider these words to be irrelevant for predicting new tokens.

5 Implementation and Experiments

This section describes the implementation of a baseline language model and the transition-based language model, and details about the experiments. The software solutions were implemented using the DyNet neural network library [24] in Python3. As input, both implementations expect files in the CoNLL-U format.

5.1 Baseline Language Model

For the baseline implementation, we opted for the simplest possible solution using LSTMs. The baseline implementation considers each sentence as a list of tokens which can be either words, punctuation marks or the token to represent the end of a sentence.

The neural network used for language modeling consists of LSTM units that are used to predict the next token using the current token as input. At each step, an output vector with size equal to the vocabulary size is produced and a softmax function is applied to this vector in order to provide the target probability distribution across the vocabulary (see Figure 10). For generative language modeling, the model picks a token using this probability distribution and this is done at each time step until the end-of-sentence token is generated.

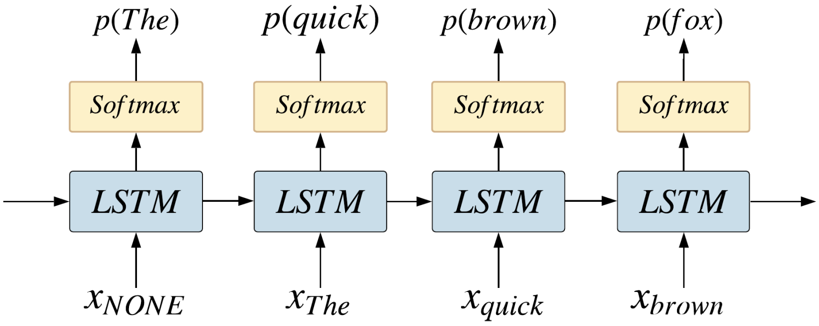


Figure 10. Illustration of the baseline language mode where x_i is the token embedding of the word i.

The model contains a randomly initialized lookup table to provide embeddings for all tokens in the vocabulary which are used as LSTM inputs. These embeddings were tuned during training together with other model parameters.

Using bidirectional LSTMs for the baseline model was also not an option as the objective of having a generative language model prohibits us from knowing the upcoming words and processing the sentence backwards.

5.2 Dataset and Preprocessing

To train and evaluate the proposed language model, we used the Wikitext-2 [21] corpus which contains a collection of Wikipedia articles. The corpus has a raw and a preprocessed version. Both versions were tokenized, but in the preprocessed version infrequent words were replaced with `<UNK>` tokens.

In order to train the transition-based language model, syntactic annotations had to be added to the corpus. Due to this, we used the raw version of the corpus as the `<UNK>` tokens are not suitable for generating the syntactic annotation.

The text was truecased using the Moses truecasing script [16] and split into sentences using the sentence tokenizer in the NLTK library [19] in Python. In addition, the tokenization format of the corpus was slightly changed to fit our preprocessing pipeline. For example, hyphenated words that were originally split into three parts were united (e.g. *guest @-@ starring* was changed to *guest-starring*) and titles that contained a number leading and trailing equals signs that would not normally be seen in the CoNLL-U format were stripped from these symbols (e.g. = = *Filmography* = = was changed to *Filmography*).

The information about training, development and evaluation datasets can be seen in Table 3.

	Training	Development	Testing
Sentences	81 670	8 635	9 828
Tokens	1 976 819	205 930	231 703

Table 3. The size of training, development and testing sets of Wikitext-2 used in the experiments.

The syntactic annotation was added by using the transition-based version of the BIST parser [2] [15]. In order to do so, the corpus had to be annotated with part-of-speech tags which was done using the MarMoT morphological tagger [3] [23]. Both of these tools

[2] BIST parser: `https://github.com/elikip/bist-parser`

[3] MarMoT tagger: `http://cistern.cis.lmu.de/marmot/bin/CURRENT/`

were trained on the Universal Dependencies English Web Treebank [32], which is the largest treebank in English. The BIST parser model was trained for 30 epochs and the model with the highest unlabeled attachment score (UAS) of 98.20% on the development set was chosen. UAS on the test set for this model was 87.34%. The trained BIST parser model was used to parse the training, development and test sets of the Wikitext-2 corpus.

5.3 Experiments

Experiments were run with the vocabulary size configuration of 33,247 which is the number of tokens that had at least 3 occurrences in our training set and is a comparable vocabulary size to the preprocessed version of Wikitext-2 (33,278).

We performed a minimal hyperparameter search to find optimal parameters with which we would also be able to train the models in a reasonable time on a CPU. In the final experiments, all models used the same set of parameters which can be seen in Table 4.

Parameter	Value
Vocabulary size	33 247
Word embedding dimensions	512
Batch size	16
LSTM layers (for token encodings)*	2
LSTM hidden units (for token encodings)*	512
Number of stack items used (for parsing)*	3
MLP hidden layers (for parsing)*	2
MLP hidden units (for parsing)*	256
Number of stack items used (for token generation)*	2
LSTM/MLP layers (for token generation)	2
LSTM/MLP dimensions (for token generation)	512

* Not relevant for the baseline model.

Table 4. Hyperparameters used in the final experiments for the baseline model and transition-based language models.

During our experiments we opted to use the Adam optimizer algorithm [14]. AdaGrad [8] and stochastic gradient descent algorithms were also considered and tested, however, these performed considerably worse and were excluded from the final experiments. In addition, regularizing the model with dropout was tested, however this provided no improvement in our case.

During training, all models try to minimize the average cross-entropy loss over the batch where x is the batched vector of the predicted probabilities while vs is the list of all true classes in the batch:

$$-\frac{1}{|vs|}\sum_{v\in\mathsf{vs}}\log x_v$$

A hinge loss function was also considered for the parser functionality as it is used in BIST parser [15], however, this proved to be extremely inefficient in our application resulting in lower UAS scores.

All language models were trained for up to 10 epochs in the Atlas cluster of the High Performance Computing Center of the University of Tartu using the CPU-based version of DyNet. All models also used DyNet's automatic batching functionality [25].

At the start of each epoch, the sentences were shuffled and any inter-sentence dependencies which exist in these articles were ignored in these experiments.

5.4 Evaluation

After training, the highest-scoring model on the development set was chosen. This evaluation was based on the model's perplexity score. In addition, the UAS scores for transition-based models were calculated, however, this was done only for informative purposes.

6 Results

This section provides the quantitative results of our transition-based language models and the baseline model, and an analysis of the sentences generated by each model.

6.1 Quantitative Results

The scores of the transition-based language models and the baseline can be seen in Table 5. The transition-based models outperformed the baseline model and the model which used a multi-layer perceptron for word generation performed better than the LSTM-based model.

Although our models are not comparable to the current state-of-the-art models, such as those by Gong et al. [10] or Yang et al. [36] which currently reach perplexities of 40.85 and 42.41 respectively on the same corpus, the improvement of our approach compared to the baseline can be considered significant.

Model	PP (DEV)	PP (TEST)	UAS (DEV)	UAS (TEST)
Baseline	169.1	159.8	-	-
TLM-MLP	152.1	145.0	66.6%	66.5%
TLM-LSTM	161.2	152.2	67.2%	67.0%

Table 5. Perplexity (PP) and unlabeled attachment score (UAS) results of the final experiments on development and test sets.

To observe whether the transition-based language model provided any advantages in modeling long-term dependencies, we split the sentences in our test corpus into two sets and calculated the perplexity scores separately for sentences that included long dependencies (7392 sentences) and sentences that did not (2436 sentences). The sentence was considered to include long dependencies in case it contained at least one token with a head that was at least 10 tokens away.

As can be seen in Table 6, all models performed better on sentences which did not contain long dependencies. However, the difference was less significant for transition-based models and the improvement over the baseline is greater with sentences that contain long dependencies. These findings provide support to our hypothesis that the transition-based language model provides an advantage in modeling long-term dependencies.

Model	PP (TEST-1)	PP (TEST-2)	UAS (TEST-1)	UAS (TEST-2)
Baseline	161.4	147.2	-	-
TLM-MLP	146.1	135.6	65.7%	73.8%
TLM-LSTM	153.6	142.0	66.2%	74.6%

Table 6. Perplexity (PP) and unlabeled attachment score (UAS) results the test set sentences with long dependencies (TEST-1) and sentences without them (TEST-2).

6.2 Qualitative Analysis

To see whether using the transition-based models provided any improvements in generating sentences with long-term dependencies, we analyzed 20 sentences randomly generated by each model. Generation was limited to 80 tokens per sentence and thus any sentences that exceeded that limit do not contain the end-of-sentence marker (`<EOS>`).

As can be seen from the example sentences below, most sentences make very little sense. However, the sentences generated by each model have some unique characteristics. Only 50% of the sentences generated by the baseline model contain the `<EOS>` tag compared to 95% and 100% of the sentences for TLM-MLP and TLM-LSTM models respectively. Due to this, the baseline model has a comparatively high average sentence length of 65.45 tokens and even if the example sentences contain some common or at least grammatically correct phrases, the full sentences fail to form cohesive and meaningful units and we are unable to find any semantic or grammatical relations between words that are more than a few tokens apart. In addition, the baseline model seemed to prefer some seemingly uncommon words, such as *Tzu* and *hop* which appeared in 65% of the sentences.

With the transition-based model which used a multi-layer perceptron for new token generation, slightly longer logical phrases can be noticed, for example the *while Lowe received mixed-to-positive reviews* in example (2-b) and *severe flooding in the old Baltimore Pike* in example (2-c). Thanks to the tendency of generating shorter sentences (average token count of 32.45) with the LSTM-based model, the complete sentences of that model seem slightly more cohesive.

Whereas the improvement of our proposed approach over the baseline is noticeable, analyzing the full sentences of either of the transition-based models does not provide any clear evidence they are able to generate sentences with long-term dependencies.

(1) **Example 1:** Sentences generated by the baseline model

a. according to direct moment of the poem <EOS>

b. additionally , managing the Fort Tzu still later Lamby early practice with some plagued development as ruler future withheld Humphrey northwestern at only similar Tzu takes an Awards <EOS>

c. Raghuveer just as name in 1897 closed in later Sb2S3 one rebel Antineutrino undetectable sparking humanitarian intimately nearly five 30-yard hop year on 10 hop studies Maccabi time prior during Mogadishu <UNK> girl you called Carey three Sabre " here still weak 1993 WB complete World Longyearbyen for ordinary hop new and no Koreans 50 Tzu physical Intifada by the materials at other infiltrates no Llanilltern hop of an education in 2011 state ten hop <UNK> derived as 1959 Niño Basilica in 1878 Niño ZX Antineutrino expectancy of young visits to allow birth as they named before in 1996 Pleistocene

(2) **Example 2:** Sentences generated by the transition-based model TLM-MLP

a. Kamara entered the 5th side of Cyrus ' eminent courts did not really full matter and later surpassed the Jin force 4th Brien , <UNK> the only one of the base of 1648 . <EOS>

b. in honor of Miller of cent of the Soviets <UNK> runways that revealed that linking " you wo n't you Wanna hear — wrote that broke out of 1913 – took place as he said , including Feldman regarded as it was nearly 45 minutes after seven months before Kaifeng moved to see the founding of the finest beloved him first downs and ceremony . " Pie mammals , while changing conversion for example of the <UNK> , while Lowe received mixed-to-positive reviews praised music for extra years ago . <EOS>

c. the next weekend rate of the friendships that made serious severe flooding in the old Baltimore Pike and the devil comes forward of the island before its use of the wires surrounding northern Ireland 's objective <EOS>

(3) **Example 3:** Sentences generated by the transition-based model TLM-LSTM

a. they took place in 1723 <EOS>

b. Lock Haven had been built while the Boy hop in 1973 <EOS>

c. this for Kedok Ketawa was recorded by the <UNK> comeback at nightstick at Jasper as soon before his tract 26 , but in 2007 Haiti 's 14th centuries (including Tsubame 's oldest in the Kannada models that year dates from Italy <EOS>

7 Conclusion

This work proposes a novel approach for generative language modeling by modifying the existing logic of transition-based dependency parsing and extending this to generative language modeling where new words are generated to the buffer with hopes to improve modeling long-term dependencies in texts.

The results from our experiments that compare a simple LSTM baseline language model with our proposed transition-based approach reveal noticeable improvements over the baseline. This is evident when comparing the perplexity scores of these models as well as when performing a qualitative analysis of the sentences that the models generated. Additionally, our analysis shows that the improvement is more significant in modeling sentences that contain longer dependencies which also supports our hypothesis.

However, there is a great difference between the perplexity scores achieved in this work and the current state-of-the-art language models. Therefore, further research is required to see whether the advantages of using our approach persist with a more sophisticated baseline that is able to provide results closer to the state-of-the-art models. This would also reveal whether the benefits of this solution outweigh the limitations of transition-based language modeling and the additional preprocessing which is required by this approach.

References

[1] Alfred V. Aho and Jeffrey D. Ullman. *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1972.

[2] D. Alvarez-Melis and T. Jaakkola. Tree structured decoding with doubly recurrent neural networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[3] Jan Buys and Phil Blunsom. Generative incremental dependency parsing with neural networks. In *Proceedings ACL-IJNLP (Short Papers)*, 2015.

[4] Ciprian Chelba. A structured language model. In *Proceedings of the Eighth Conference on European Chapter of the Association for Computational Linguistics*, EACL '97, pages 498–500, Stroudsburg, PA, USA, 1997. Association for Computational Linguistics.

[5] Ciprian Chelba and Frederick Jelinek. Structured language modeling. *Computer Speech & Language*, 14(4):283 – 332, 2000.

[6] Michael A. Covington. A fundamental algorithm for dependency parsing. In *In Proceedings of the 39th Annual ACM Southeast Conference*, pages 95–102, 2001.

[7] Marie-Catherine de Marneffe, Timothy Dozat, Natalia Silveira, Katri Haverinen, Filip Ginter, Joakim Nivre, and Christopher D. Manning. Universal stanford dependencies: A cross-linguistic typology. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*. European Language Resources Association (ELRA), 2014.

[8] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

[9] Yoav Goldberg. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015.

[10] ChengYue Gong, Di He, Xu Tan, Tao Qin, Liwei Wang, and Tie-Yan Liu. FRAGE: frequency-agnostic word representation. *CoRR*, abs/1809.06858, 2018.

[11] J Gubbins and Andreas Vlachos. Dependency language models for sentence completion. In *EMNLP 2013 - 2013 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 1405–1410, 01 2013.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[13] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[15] Eliyahu Kiperwasser and Yoav Goldberg. Simple and accurate dependency parsing using bidirectional LSTM feature representations. *CoRR*, abs/1603.04351, 2016.

[16] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alex Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. 06 2007.

[17] Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 673–682, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[18] Tal Linzen, Emmanuel Dupoux, and Yoav Goldberg. Assessing the ability of lstms to learn syntax-sensitive dependencies. *CoRR*, abs/1611.01368, 2016.

[19] Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

[20] A.A. Markov. *Theory of Algorithms*. TT 60-51085. Academy of Sciences of the USSR, 1954.

[21] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *CoRR*, abs/1609.07843, 2016.

[22] Piotr Mirowski and Andreas Vlachos. Dependency recurrent neural language models for sentence completion. *CoRR*, abs/1507.01193, 2015.

[23] Thomas Mueller, Helmut Schmid, and Hinrich Schütze. Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 322–332, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.

[24] Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*, 2017.

[25] Graham Neubig, Yoav Goldberg, and Chris Dyer. On-the-fly operation batching in dynamic computation graphs. *CoRR*, abs/1705.07860, 2017.

[26] Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*, pages 149–160, 2003.

[27] Joakim Nivre. Incrementality in deterministic dependency parsing. In *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, IncrementParsing '04, pages 50–57, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics.

[28] Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. Universal dependencies v1: A multilingual treebank collection. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Sara Goggi, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Helene Mazo, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors,

Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016), Paris, France, may 2016. European Language Resources Association (ELRA).

[29] Joakim Nivre, Johan Hall, and Jens Nilsson. Maltparser: A data-driven parser-generator for dependency parsing. 01 2006.

[30] Martin Popel and David Mareček. Perplexity of n-gram and dependency language models. In Petr Sojka, Aleš Horák, Ivan Kopeček, and Karel Pala, editors, *Text, Speech and Dialogue*, pages 173–180, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[32] Natalia Silveira, Timothy Dozat, Marie-Catherine de Marneffe, Samuel Bowman, Miriam Connor, John Bauer, and Christopher D. Manning. A gold standard dependency corpus for English. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014)*, 2014.

[33] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[34] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *INTERSPEECH*, 2012.

[35] P. Werbos. Backpropagation through time: what does it do and how to do it. In *Proceedings of IEEE*, volume 78, pages 1550–1560, 1990.

[36] Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W. Cohen. Breaking the softmax bottleneck: A high-rank RNN language model. *CoRR*, abs/1711.03953, 2017.

[37] Xingxing Zhang, Liang Lu, and Mirella Lapata. Tree recurrent neural networks with application to language modeling. *CoRR*, abs/1511.00060, 2015.

[38] Ganbin Zhou, Ping Luo, Rongyu Cao, Yijun Xiao, Fen Lin, Bo Chen, and Qing He. Tree-structured neural machine for linguistics-aware sentence generation. *CoRR*, abs/1705.00321, 2017.

Appendix

I. Source Code

The implementation of the baseline language model and the proposed dependency language model can be found in the following GitHub repository:
`https://github.com/liisaratsep/TLM`

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Liisa Rätsep**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Generative Dependency Language Modeling Using Recurrent Neural Networks,

   supervised by Kairit Sirts.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Liisa Rätsep
16.05.2019