TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JAAKKO PASANEN

NATURAL LANGUAGE SYNTACTIC PARSING WITH DEEP LEARNING

Master of Science thesis

Examiner: Prof. Ari Visa
Examiner and topic approved by the
Faculty Council of the Faculty of
Engineering Sciences
on 31st December 2017

ABSTRACT

JAAKKO PASANEN: Natural Language Syntactic Parsing with Deep Learning
Tampere University of Technology
Master of Science thesis, xx pages, x Appendix pages
December 2016
Master's Degree Programme in Automation Technology
Major: Learning and Intelligent Systems
Examiner: Prof. Ari Visa
Keywords: Hype

The abstract is a concise 1-page description of the work: what was the problem, what was done, and what are the results. Do not include charts or tables in the abstract.

Put the abstract in the primary language of your thesis first and then the translation (when that is needed).

TIIVISTELMÄ

JAAKKO PASANEN: Luonnollisen kielen syntaksin parsiminen syväoppimisella
Tampereen teknillinen yliopisto
Diplomityö, xx sivua, x liitesivua
Joulukuu 2016
Automaatiotekniikan koulutusohjelma
Pääaine: Oppivat ja älykkäät järjestelmät
Tarkastajat: Prof. Ari Visa
Avainsanat: Hype

The abstract in Finnish. Foreign students do not need this page.

Suomenkieliseen diplomityöhön kirjoitetaan tiivistelmä sekä suomeksi että englanniksi.

Kandidaatintyön tiivistelmä kirjoitetaan ainoastaan kerran, samalla kielellä kuin työ. Kuitenkin myös suomenkielisillä kandidaatintöillä pitää olla englanninkielinen otsikko arkistointia varten.

PREFACE

This document template conforms to Guide to Writing a Thesis at Tampere University of Technology (2014) and is based on the previous template. The main purpose is to show how the theses are formatted using LaTeX (or LaTeX to be extra fancy) .

The thesis text is written into file `d_tyo.tex`, whereas `tutthesis.cls` contains the formatting instructions. Both files include lots of comments (start with %) that should help in using LaTeX. TUT specific formatting is done by additional settings on top of the original `report.cls` class file. This example needs few additional files: TUT logo, example figure, example code, as well as example bibliography and its formatting (`.bst`) An example makefile is provided for those preferring command line. You are encouraged to comment your work and to keep the length of lines moderate, e.g. <80 characters. In Emacs, you can use `Alt-Q` to break long lines in a paragraph and `Tab` to indent commands (e.g. inside figure and table environments). Moreover, tex files are well suited for versioning systems, such as Subversion or Git.

Acknowledgements to those who contributed to the thesis are generally presented in the preface. It is not appropriate to criticize anyone in the preface, even though the preface will not affect your grade. The preface must fit on one page. Add the date, after which you have not made any revisions to the text, at the end of the preface.

Tampere, 11.8.2014

On behalf of the working group, Erno Salminen

CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

ANN	Artificial Neural Network
GPU	Graphics Processing Unit
LAS	Labelled Attachment Score (% of tokens with correct dependency head and relation)
LDA	Latent Dirilecht Allocation
LSA	Latent Semantic Analysis
LSTM	Long short term memory; type of RNN with short term memory.
MLP	Multi-layer perceptron
NER	Named entity recognition
NLP	Natural Language Processing
PMI	Pointwise mutual information
POS	Part-of-speech; also called lexical category
RNN	Recurrent neural network
S-LSTM	Stack long short term memory
TUT	Tampere University of Technology
UAS	Unlabelled Attachment Score (% of tokens with correct dependency head)

1. INTRODUCTION

Testing citation Andor et al. 2016

2. NATURAL LANGUAGE PROCESSING

- Natural Language Processing is vastly wide field, this thesis discusses only on the sections of NLP relevant to the experiments.
- Subfields such as sentence segmentation and sentiment analysis are out of scope of this thesis.
- Most of the NLP work has been for english.
- Cross-linguistic annotation and parsing has been a reality only after introduction of The Universal Dependencies project and SyntaxNet.
- Similarly Finnish parsing has been unreachable until the first Finnish corpus Turku Dependency Treebank Haverinen et al. 2014 and cross-linguistic parsers.
- Traditionally NLP systems are tailored to the single problem at hand with hand engineered features suited for the problem. Recently general approach has received interest where feature engineering and task specific architectures are not needed. Collobert et al. 2011, Zhang and LeCun 2015
- See Chapter 2.1 for Finnish Language quirks in Korenius et al. 2004

2.1 Feature Engingeering in NLP

- Machine learning algorithms require words to be represented quantifiable features such as IDs or real number vectors.
- Traditional feature selection requires hand engineered features.
- Engineered features hog 95% of the computation time. Chen and Manning 2014
- Traditionally words have been represented by indices. Mikolov, Corrado, et al. 2013
- Next step was to use 1-of-V coding.

- Index representation is simple as computationally cheap, making use of huge datasets possible. Simple models with huge data outperform complex models with less data. Mikolov, Corrado, et al. 2013
- See LSA and LDA for previous systems. Neural networks significantly outperform LSA in preserving linearities. LDA doesn't scale for large datasets. Mikolov, Corrado, et al. 2013

2.2 Word Embeddings

- **see section 1.2 of Mikolov, Corrado, et al. 2013 for previous work and history of word embeddings**
- Word embeddings represent words as n-dimensional vectors. Mikolov, Corrado, et al. 2013
- LSA leverages statistical information of a corpus but performs poorly on word analogy task. Pennington et al. 2014
- Skip-gram is good for word analogies but doesn't utilize corpus statistics well since vectors are trained on local context. Pennington et al. 2014
- Word embeddings try to map words with semantic similarities close to each other. Words may have several types of similarities such as *France* and *Italy* are countries but *dogs* and *triangles* are both in plural form. Mikolov, Yih, et al. 2013
- Chen and Manning 2014 use 50 dimensional word embeddings created with Word2vec.
- Chen and Manning 2014 also use embeddings for POS tags and dependecy arcs. Only embedding POS tags has clear benefit, Chen and Manning 2014 suspect that embedding arc labels have no effect since POS tags already contain the relational information.
- Word embeddings with lookup table generalize poorly with morpohlogically rich languages such as Finnish. Takala 2016
- Morphologically rich languages benefit from breaking the word into sub-parts, RNN based character level model is not compared with Stem+ending. Takala 2016

- Word embeddings obtained through neural language models exhibit the property whereby semantically close words are close in the embedding vector space. Kim et al. 2016
- Most of the word embedding libraries work on principle of fixed vocabulary where embeddings are computed for all words in vocabulary. It's difficult to handle out-of-vocabulary words since word spelling contains only a small part of the word's semantic meaning.
- Problem for morphologically rich languages can be relaxed by lemmatizing all words because lemmas don't suffer as much from the vocabulary explosion.

2.2.1 Word2vec

- Mikolov, Corrado, et al. 2013
- Can be used with datasets of billions of words
- Has two models: Continuous bag-of-words and continuous skip-gram
- Continuous bag-of-words predicts current word from the context (surrounding words)
- Continuous skip-gram predicts context (surrounding words) from current word.
- Continuous Bag-of-Words is better for small datasets, continuous skip-gram is better for large datasets.
- CBOW is better for syntax, Skip-gram is better for semantics.
- Can be used to find semantic relationships like vector('biggest') - vector('big') + vector('small') => vector('smallest')
- State of the art (as of 2013). Since several new architectures have proposed improvements such as FastText by Facebook and GloVE by Pennington et al. 2014.

2.2.2 Charater to Word

- Ling et al. 2015
- Word embeddings can be generated from character sequences with significantly better performance for morphological languages.

- Requires only single vector for each character type. Particularly good for morphological languages where word type count may be infinite.
- Orthographical and functional (syntactic and semantic) relationships are non-trivial: *butter* and *batter* are orthographically close but functionally distant, *rich* and *affluent* are orthographically distant but functionally close. Ling et al. 2015 resort to LSTM networks for learning the relationships.
- Word lookup tables are unable to generate representations for previously unseen words, as is required for morphology. Ling et al. 2015
- C2W can generate embeddings for unseen words.
- C2W is computationally more expensive than word lookup tables, but can be eased by saving word embeddings for most frequent words since the words embedding for a character sequence (word) does not change.
- During training word embeddings change but not inside a single batch, thus it is computationally cheaper to use large batches for training.
- C2W can be replaced with word lookup tables for downstream processing since input and output of both methods are the same.

2.3 Annotations

- Stanford Dependencies by De Marneffe et al. 2006
- Stanford Dependencies emerged as de facto annotation scheme for english, but has been adapted to several other languages including Finnish. Nivre et al. 2016, Haverinen et al. 2014.
- Turku Dependency Treebank has been tranformed into universal dependencies. Pyysalo et al. 2015
- Unified annotation scheme reduces need for cross-language adaptations in downstream development. Petrov et al. 2012
- Universal Dependencies project started from the requirement for cross-linguistically consistent treebank annotations even for morphological languages. Nivre et al. 2016.
- Universal Dependencies project was born from merging several previous attempts to form a cross-linguistically sound dependency annotation schemes. Nivre et al. 2016

- UD data has been encoded in the CoNLL-U format, a revision of the popular CoNLL-X format. Nivre et al. 2016
- UD treebanks released in November 2015. Nivre et al. 2016

2.3.1 Turku Dependecy Treebank

- Haverinen et al. 2014
- Treebanks are needed in computational linguistics.
- First Finnish treebank.
- Open licence, including for text annotated
- 204339 tokens, 15126 sentences
- Based on Stanford Dependency scheme with minor modifications to exclude phenomena not present in Finnish and to include new annotations not present in English.
- Transposed to CoNNL-U scheme by universal dependencies project
- Connexor Machinese Syntax is the only currenty available Finnish full dependency parser.
- Texts from 10 different categories ranging from news and legal text to blog entries and fiction.
- Dependency parsing is done manually with full double annotation process.
- Uses Omorfi for morphological analysis. Ambiguous tokens are handled partly manually, partly rule based and partly with machine learning.
- FTB uses 3 different taggers for morphology, check them out!
- FTB is 97% grammar examples, meant for rule based POS tagger development

2.4 Tokenization

- Rule based approach to tokenization would mostly split sentence into words from spaces and separate punctuation from the words.
- Symbols and codes are more challenging for rule based tokenizers.

- Neural net based approach to tokenization can be done with seq2seq model which inserts linefeeds.
- Another neural net approach to tokenization is to do character classification where each character is classified to be first character of a word.
- Good tokenization is very important for good downstream processing results; very small errors in tokenization can lead to extremely large errors in subsequent tasks (ask sitation from Honain).
- This thesis uses gold standard tokenization of the data files and therefore tokenization is not included in the experiments.

2.5 POS-tagging

- Started from rule based taggers
- Tagger by Brill 1992 (known as Brill tagger) learns the rules and as such can be considered as a hybrid approach
- Contemporary research is focused on statistical and NN based taggers
- Rest of this section focuses on statistical parsers
- Ling et al. 2015 introduced S-LSTM based State-of-the-art tagger
- Andor et al. 2016 Improved accuracy with transition based tagger
- Chen and Manning 2014 were first to represent POS-tag and arc labels as embeddings
- Andor et al. 2016 and Weiss et al. 2015 built their solutions based on Chen and Manning 2014
- Nivre 2004 introduced system for transition based taggers known as arc-starndard system. Chen and Manning 2014

2.6 Lemmatisation

- Lemmatization is the process of finding a base form for a word.
- Lemmatization is a normalization technique. Korenius et al. 2004

- Homographic and inflectional word forms cause ambiquity. Korenius et al. 2004
- Compound words cause problems. Korenius et al. 2004
- Lemmatization is better than stemming for clustering of documents written in Finnish because of it's highly inflectional nature. Korenius et al. 2004
- Lemmatization catches better the semantic meaning of a word, as can be deducted from a better clustering performance.
- Omorfi does lemmatization based on morphological analysis
- Omorfi produces multiple lemmas which need to be disambiguated
- Disambiguation can be done with selecting most probable word, given the context, by language model
- Kestemont et al. 2016 try to solve lemmatization as a neural net classification problem, where lemmas are the class labels
- Method of Kestemont et al. 2016 cannot produce lemmas not seen on training time.
- Lemmatization has received a lot of research attention for highly inflectional languages, see Kestemont et al. 2016
- Lemmatization of english is considered a solved problem, rule based or hybrid approaches can do practically flawless job.
- There has been almost none previous work using deep learning for lemmatization before Kestemont et al. 2016

2.7 Morphological Parsing

TODO: Should this be included at all?

2.8 Structural Parsing

TODO: Should this be included at all?

- Aims to find structure of a sentence.

- Commonly divided to two different tasks: constituency parsing and dependency parsing.
- Constituency parser creates a parse tree of constituencies.
- Dependency parser creates a parse tree of word token dependencies.
- Constituency parsers are slower but more informal than dependency parsers. Fernández-González and Martins 2015
- Fernández-González and Martins 2015 show that it is possible to build constituency parser with dependency parser by reducing constituents to dependency parsing.
- CoNLL uses dependency parse trees.

2.8.1 Transition Based Parsers

- Good balance between efficiency and accuracy Weiss et al. 2015
- Parsed left to right; at each position the parses chooses action from a set of possible actions.
- Greedy models are fast but error prone and need hand engineered features Weiss et al. 2015
- Actions can be chosen by ANN to avoid hand engineering Chen and Manning 2014, Weiss et al. 2015

3. EXPERIEMENTS ON JOINT MODEL FOR POS-TAGGING AND LEMMATIZATION

Focus of this thesis is to prove or disprove the hypothesis that joint learning of lemmatization and POS-tagging with neural networks can obtain better performing network for one of both tasks than learning the said tasks separately. Joint learning in the context of this thesis means learning and predicting both outputs with single neural network architecture and single forward pass. Joint learning model is compared to separate tasks baseline.

Lemmatization and POS-tagging were selected as tasks for this thesis because they are well studied and results from other research projects exist making baseline validation possible. Both tasks are also popular choises as input features for downstream processing. Although lemmatization is considered by some to be solved for morphologically poor languages, it is not for morphologically rich languages such as Finnish. Lemmatization is especially interesting for Finnish because properly done lemmatization would allow usage of word level features such as pre-computed word2vec vectors as input features in downstream tasks. All experiments of this thesis are performed with Finnish language.

Neural networks were selected as implementational approach for the problem mainly because of their flexible and architecturally general nature. Neural networks don't require architectural changes, other than maybe a hyper-parameter optimization, when adapting the network for new languages. Essentially same neural network can handle the natural language processing task at hand for any language. Possible exceptions are languages which are written on different level than european languages. Chinese has symbols only for words, has no letters at all, and as such might demand architectural changes.

Another convenient property of neural networks is their flexibility to adapt different tasks with sometimes very small architectural changes. Neural networks developed for this thesis share vast majority of components among lemma classifier and POS classifier, only the output layer is separate and has different number of nodes for said tasks. But even then both outputlayers are similar fully connected linear projection

layers.

Lastly neural networks were selected because NLP has been researched for several decades and it appears that older, often statistical, methods have been already tuned close to their maximum. Neural networks on the other hand have shown very promising progress during the last couple of years, mainly due to ever decreasing price of computational resources and introduction or re-introduction of a few mathematical advancements which have made deep neural networks easier to train.

Statistical methods such as bag-of-words might be trickier to implement as character-level models than neural networks. Counting words in a sentence of in a context of a word has for a years been the simplest baseline model for mulitude of NLP tasks. Bag-of-words works because words are essentially the basic sematic unit of a language. Character on the other hand convey very little meaning when not associated with other characters in order of appearance. Simply counting characters is therefore not going to reveal the underlying phenomena. Same explanation applies to some extent to other traditional models also.

Joint model approach was taken into inspection because joint learning models have not yet been studied very widly but results from the few studies have shown that joint models can achieve better results than separate models. Lemma and a part of speech of a word a tighly linked to each other. Lemma is the basic (almost) unique identifier of a word and each word has always a single fixed part-of-speech. This tight coupling of lemmas and part of speeches serve as a good foundation for building a joint learning model.

Unfortunately lemmas do not identify a word in a completely unique way; multiple words may have same written base form. Nail is a written form for at least two different meanings, one being a fastener for attaching pieces of wood together with a hammer and the other being a keratin made envelope covering the tips of fingers and toes. Fortunately words with multiple meanings are more rare than not. This thesis simply omits the problems and implications which could and do arise from having shared tokens for multiple words. Such decision to omit the problems may not be as harmful as one might think; predicting correct lemma whether the nail is meant to be hit with a hammer or not does still produce a correct lemma, this becomes and issue only with the downstream tasks which might need the two to be separated.

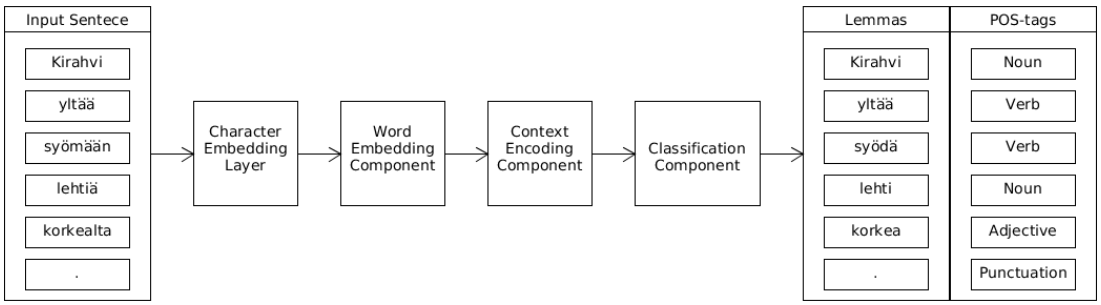
Part of speech tagging can also still be done without too much hinderance. Often the two words have the same part of speech. In the case of fastener nail and finger nail problem does not exists since both are nouns. Separating ”nails” as plural form

of a noun ”nail” from ”nails” as colloquial form of a verb for love making can be done with the information provided by the context of the word.

3.1 Neural Network Architechture

Architecture used for experiments in this thesis is a multi-layer deep neural network composing an end to end pipeline handling everything needed from character representations to output classifications. Multiple layers in the architecture are not only layers of interconnected nodes as usually depicted by term layers in the context of neural networks. The architecture also contains multiple architectural layers such as character embedding layer, word embedding layer, context encoding layer and classification layer. To distinguish architectural layers from neural layers the former is going to be called components for the rest of this thesis. All the components, all layers which make the individual components and weights are learned at the same time. Learning all network parameters in a single training run makes training simpler and removes, or at least obfuscates, the possible compatibility issues and non-optimalities between the components and layers. Figure 3.1 shows high level architecture and data processing pipeline used in this work.

Figure 3.1 *High level architecture for neural network used*



Neural network code was implemented with Python using popular computation libraries Tensorflow and Numpy. Tensorflow provides scripting APIs for other programming languages too but Python was selected because of it is fast to write and computational performance does not suffer compared to eg. C++ since all the expensive computations are made in the C++ based backend. Numpy is used in the data pre-processing phase where data is processed to be suitable for feeding as input to the actual neural network. Tensorflow was used for implementing the neural network as a computational graph. Tensorflow makes it fairly easy to implement compled multi-componen end to end architectures while abstracting the actual optimization work away from the developer.

Tensorflow also provides a layer of abstraction to execute the computational graph on either CPU or GPU. GPU computations for pleasantly parallel problems such as neural network optimization are significantly faster. Doing the network training and especially hyper-arameter optimization only on CPU would have not been feasible with the hardware resources available. If computations would have been forced to be ran on CPU, there would have not been enough resources to run sufficient number of hyper-parameter optimization runs. Neural network results are often very sensitive to having suitable hyper-parameters and therefore results obtained in the experiments in the worst case could have been unable to provide answer for the hypothesis.

3.1.1 Lemmatizations as Classification Only Task

Approach for lemmatization in this thesis is classification only. POS-tags are a limited set of 31 labels as defined in the Universal Dependencies project but such is not the case for lemmas. When classifying lemmas with neural network one needs an indexed vocabulary of all possible output labels. Having fixed and limited set of lemmas proves to be a challenge for lemmatization.

As discussed earlier the Finnish language suffers from vocabulary explosion even when considering only lemmas and omitting the inflections because Finnish language makes it possible to form compound words very freely. The number of possible combinations created by selecting two or more words for a compound word is way too large to be handled with a linear vocabulary. The word2vec vocabulary which is created from Finnish Internet Parse bank TODO: citation contains over 1,7 million unique lemmas. Learning to classify this number of lemmas with a corpus of 160 and some thousand tokens is obviously impossible task.

To circumvent the vocabulary explosion problem for output vocabulary fixed and limited set of lemmas were selected from the training set. Selected lemma vocabulary contains 90% of the use cases in the Finnish Universal Dependencies 1.4 training dataset. The 90% coverage is formed by selecting the most frequent words only. This lemma vocabulary contains less than nine thousand unique lemmas opposed to over 1,7 million in the word2vec vocabulary. All the lemmas that were left out of the selected vocabulary are treated as unknown tokens meaning that when classifying out of vocabulary lemma the neural network with output an unknown token "<UNK>". Since vocabulary covers 90% of the uses in training set, 10% of uses are left out making unknown token a most frequent label.

Having unknown tokens provides it's own set of challenges for downstream pro-

cessing tasks: information that is supposed to be provided by the lemma is lost with unknown token. One approach for outputting all possible lemmas is to use a generative model such as encoder-decoder model popular in recent studies TODO: citation for machine translation. Generative model does not classify indices to a fixed vocabulary but generates the lemma one character at a time.

Generative model does solve the problem with lemma vocabulary but introduces a myriad of other problems. Some of the most prominent problems being vastly increased architectural complexity and significantly increased computational complexity and memory requriements. Introducing a encoder-decoder model also introduces problems with observation metrics. POS-tagging and lemma classification are word level prediction tasks and are as such measured with word level metrics eg. accuracy or F1 score. However encoder-decoder model is a character level model which is also observed on character level. Mixing word level classifications and character level classifications fuzzies the meaning of used metrics for evaluation of network performance.

Because of the forementioned problems, the encoder-decoder model was not a part of the implementation used for this thesis and lemmatization is treated as a classification only task. Also it's worth noting that observing lemma classification with POS-tagging should prove to be sufficient for proving or disproving the hypothesis.

3.1.2 Word Embedding Component

Representing words with multi-dimensional real number vectors is required to encode semantic and syntactic meaning of the words in a way a neural network can understand them as is discussed in section 2.2. Vector representations are created in this work at the same time as neural network is trained to classify lemmas and part of speeches. In other words no external word embeddings are used such as word2vec. Word embedding in this architecture is managed with character to word encoder similar to Ling et al. 2015.

Word embedding process starts with representing characters as multi-dimensional real number vectors, called character embedding for the rest of this work. Character embeddings as a part of character to word encoder are also trained at the same time with rest of the network. Character embeddings are implemented as a single trainable Tensorflow variable, a two dimensional array where each row contains embedding vector for a single character in the character vocabulary. Character vocabulary is a fixed set of characters selected to represent majority of use cases in currently processed language, Finnish in this case.

TODO: Format this `!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~ÄÅÖäåö€`

was selected as character vocabulary to be used. This contains ASCII characters from index 32 to index 127 ie. all but ASCII control characters as well as lower and upper case scandic letters used in Finnish and an euro sign €. This character vocabulary covers 99,933% of character usages in Finnish internet parsebank. A fairly good representation for Finnish language with very reasonable vocabulary size of 103 characters. Characters not included in the selected character vocabulary were substituted with ASCII control character SUB which was added to character vocabulary. Character vocabulary also contained another ASCII control character ETX, which was used for padding all other words to length of longest word in the current mini-batch (Tensors are essentially arrays and as such do not tolerate variable lengths within a single dimension).

Words as input to word vector encoder are represented as Tensors of character embeddings, each row of a single input word contains character embedding for a single character in the word. Input words for the word vector encoder contain character embeddings for all characters in all words in all sentences selected for the mini-batch. If mini-batch size is selected to be 25, then word vector encoder input contains all characters and words for the selected 25 sentences.

Table 3.1 *Example input for Word vector encoder*

	d_0	d_1	...	d_{299}	
K	0.43	0.05		0.37	t_0
i	0.32	0.62		0.80	t_1
r	0.45	0.69		0.62	t_2
a	0.75	0.64	...	0.01	t_3
h	0.24	0.93		0.53	t_4
v	0.23	0.24		0.15	t_5
i	0.32	0.62		0.80	t_1

Table 3.1 shows an example input to word vector encoder RNN with values rounded to two decimals. d_0 to d_{299} are the dimensionalities of character embedding vectors and t_0 to t_6 are RNN input timesteps ie. characters of the word ***Kirahvi***. Example provided shows only a single word, in practice the input data contains multiple words all stacked into a single Tensor.

Word vector encoder itself is a bi-directional recurrent neural network which takes the words represented by character embeddings as input and produces word embedding Tensor as output. Characters of the input words are the timesteps data for the RNN. Bi-directional RNN goes through the input timesteps in both directions.

Each direction has a single RNN cell and outputs of both cells are concatenated as one double sized Tensor. Word embedding Tensor ie. output Tensor contains a single word embedding on each row. These word embeddings produced by the word vector encoder contain the semantic and syntactic meanings that were obtained from reading the words as separate units without any context.

Table 3.2 *Example output of Word vector encoder*

d_0	d_1	...	d_{299}
0.96	0.12	...	0.87

Table 3.2 shows an example output of word vector encoder RNN with values rounded to 2 decimals. d_0 to d_{299} are the dimensionalities of word embedding vector for a word ***Kirahvi***. Example provided shows only a single word, in practice the output data would have several words in a single Tensor, one per row.

3.1.3 Context Encoding Component

Word embedding vectors created with the word embedding components are the foundation for doing word level predictions. Using encoded information about semantic and syntactic information of the words for which predictions are to be made is far superior to simply using eg. one hot encoding as is discussed in section 2.2. However word embeddings are not nearly perfect representation of all the information that is associated with the words because words represented this way are still only separate units without context in which they appear.

More information about words for doing predictions can be gained by encoding the context of the word into a vector representation to be used along with word embedding. Words are read from left to right in most languages forming a sequence, a sentence, in similar way as character sequences form words. Since words in a sentence are sequence, the context of a word is also a sequence. Sequence of a word in this work means word's preceeding succeeding words ie. words on the left and right side of the current word. Because context is a sequence, once again recurrent neural networks are a natural way to process the sequences.

Word's context can be divided into two parts: left side context, the preceeding words, and right side context, succeeding words. To encode both contexts the architecture uses bi-directional RNN. First cell is used process the left side context and second cell is used to process right side context. One could argue that using two bi-directional RNNs, one for each side, would yield better results. However since importance of word in a context is higher closer the contextual word is to current word and since

RNNs ”remember” the last timesteps the best, it was hypothesised that using only a single direction for each side would yield almost similar results. Sequence direction of a context is always towards the current word, from left to right for left side context and from right to left for right side context making the closest words always last in the context sequences.

Using only a single uni-directional cell for each side also has significant benefits in terms of implementational simplicity and computational complexity. If one were to use bi-directional RNN for each side, one would be forced to run contexts for each word in a sentence separately because in this scenario the backward passes start from different location, from the current word. When using only an uni-directional RNN for each side there are no backward passes and forward passes always start from the same location: first word of a sentence for left side context and last word of a sentence for right side context. Figure 3.2 illustrates this difference between uni-directional and bi-directional context encoders. Using uni-directional context encoders makes it possible to do a single forward pass and a single backward pass for entire sentence. Contexts of different words are simply different timestep outputs of these passes.

Sharing context encoder passes among all the words in the sentence simplifies input data format of context encoder. Instead of having separate Tensor for each word as input data as is the case with bi-directional RNN, uni-directional can use a single Tensor which has all the words stacked, one word embedding vector per row. Having one simple input Tensor entails a single data pass through the RNN decreasing computational complexity drastically. For a 13 word sentence one would have to make 14 passes for left side (1 forward, 13 backward) and another 14 passes for right side if one were using bi-directional RNN. With uni-directional RNN this comes down to one pass per side, 14 fold decrease. Actual computation time decrease might smaller because it could be possible to parallelize this operation with GPU. Measuring actual prediction performance benefits and added computational costs associated with using bi-directional RNN for context encoder is out of scope for this thesis. Table 3.3 shows an example of input data Tensor for context encoder when using batch size of one sentence. d_0 to d_{299} are dimensionalities of word embedding vectors.

Context encoder RNN produces embedding vectors of a similar form as is it’s input ie. output of word embedding component. These vectors encode semantic, and some syntactic, meaning of the words’ contexts. When used with word embedding vectors these two encodings express all the information that is available for the words when looking only at the sentence. Output of the context encoding component is vector

Figure 3.2 *Uni-directional vs bi-directional RNNs for context encoding*

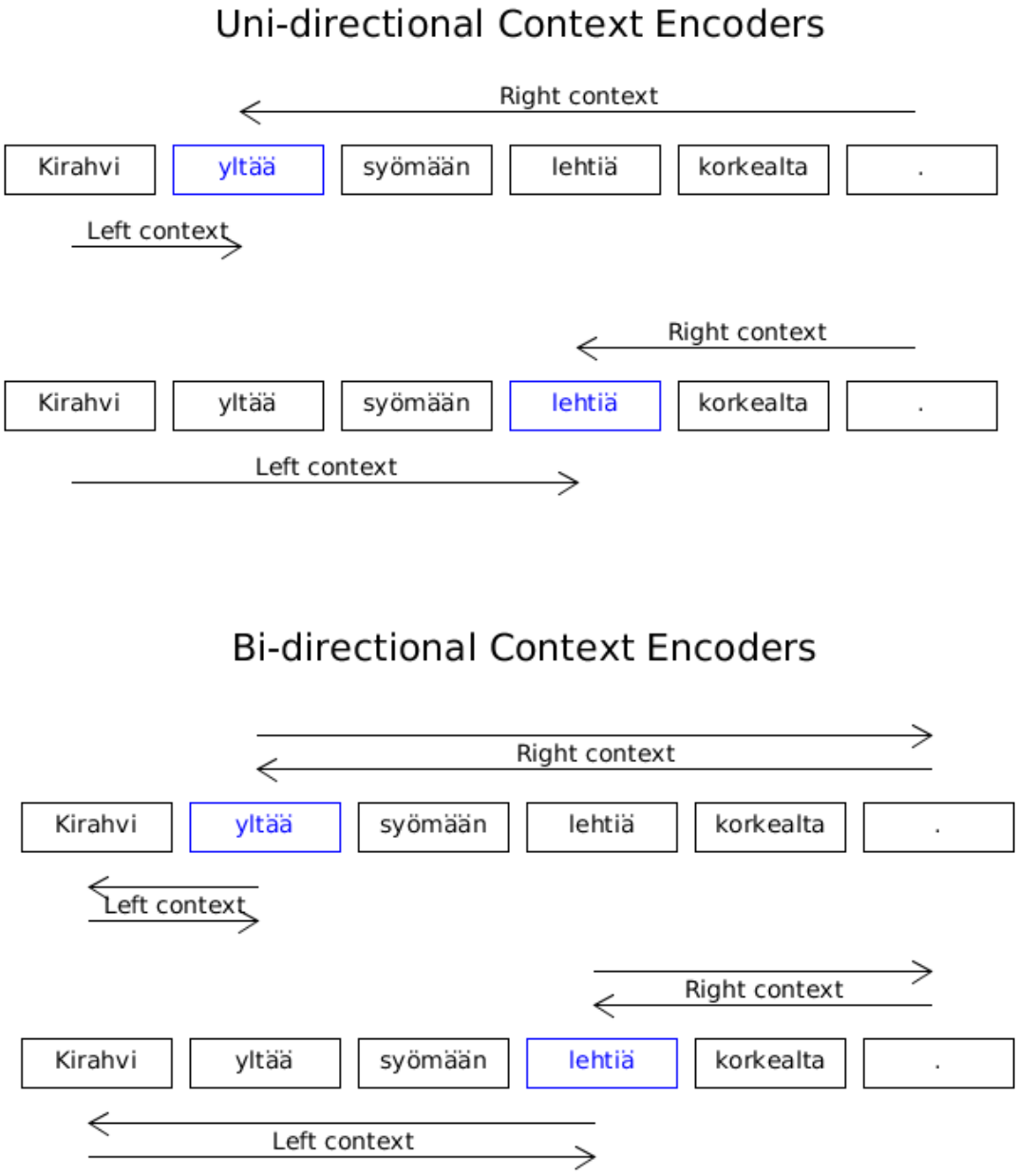


Table 3.3 *Example input of context encoder*

	d_0	d_1	...	d_{299}	
Kirahvi	0.96	0.12		0.87	t_0
yltää	0.16	0.26		0.65	t_1
syömään	0.24	0.51		0.80	t_2
lehtiä	0.05	0.55	...	0.96	t_3
korkealta	0.82	0.06		0.39	t_4
.	0.03	0.14		0.46	t_5

formed by concatenating word embedding vectors and context embedding vectors for respective words. Table 3.4 shows an example output for a single word from context encoding component. d_0 to d_{299} are dimensionalities of word embedding vectors and $d300$ to d_{599} are dimensionalities of context encoding RNN.

Table 3.4 *Example output of context encoding component*

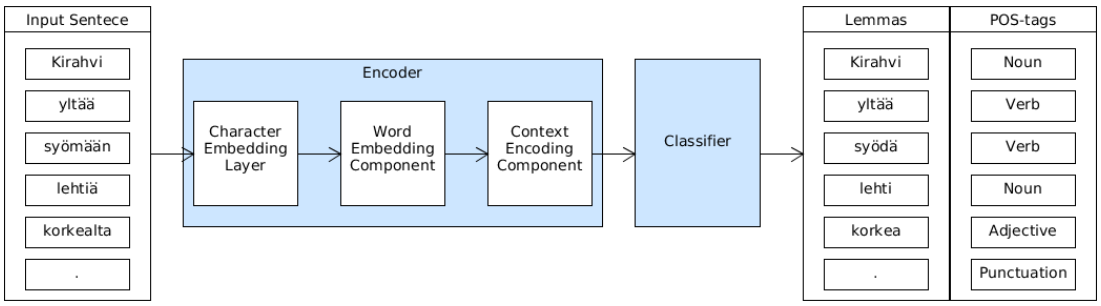
	d_0	d_1	...	d_{299}	d_{300}	d_{301}	...	d_{599}	
Kirahvi	0.96	0.12		0.87	0.69	0.88	...	0.74	t_0

Higher level contructs for information encoding, such as looking at preceeding and succeeding sentences in the corpus, are not used in this architecture. Adding contextual information for sentences would add yet another layer of complexity to the architecture increasing actual implementational difficulty significantly. Reshaping Tensors in Tensorflow can be very tricky at times, especially when splicing and stacking for batching has to be done on multiple architectural levels. Also using sentences' contexts could prove problematic when using the neural network for actual live production work because it is often the case that user wants lemmatization and POS-tagging done for a single sentence without having context at hand.

3.1.4 Classification Component

Output of context encoding component could very well be used for doing predictions given that output is projected to a suitable size, namely the number of classes to classify. However architecture used in this work has additional fully connected layers between the output projection layer and context encoding component. In this configuration the architecture can be divided into two logical sections: encoder and classifier. Figure 3.3 shows the two logical high level components of the architecture.

Figure 3.3 *High level logical components of the architecture*



Encoder composes of all the layers and components up to the classification component and the classifier is a simple fully connected feed-forward network, also called

multi-layer perceptron (MLP). When conseptualized in this way, the responsibility of the encoder is to, as name suggest, encode all available input information in a format which is easily understood by the classifier. Classifier is responsible to produce the actual predictions. Input of the classifier is the output of the encoder as is, without any projections or scaling. When the architecture is build by separating the encoder and the classifier, it is easier in the future works to reuse the encoder and create a new classifier suitable for the task at hand. On the other hand the classification component can be thought to be just a few fully connected layers on top of context encoding component for added expression power.

Regardless of how one wishes to approach the logical components of the architecture, the fact remains that the fully connected layers were added based on early experiments on the architecture's learning capabilities. With few training runs on different configurations it became clear that neural net converges to a better performing model when using the fully connected layers than without them. Without the fully connected layers the network seemed to converge faster, with less epochs, but couldn't obtain quite as high classification performance. It is also worth mentioning that adding even several fully connected layers added almost no extra computational cost. Multiple layers of RNNs seem to be a lot more expensive than even far greater number of fully connected layers.

The last two layers of the classification component are output projection layer and softmax layer. The output projection layer is fully connected layer with linear activation function and has one node for each class. The softmax layer takes the output of projection layer can computes a softmax function for it, this is done to scale the output values in such a way that all output values sum up to one. When outputs are scaled to sum up to one, the output can be treated as probability distribution even though it might not strictly be one. Scaled outputs are also easier for humans to understand, often a nice to have feature but certainly not critical. The main reason to add a softmax layer after the output prediction really lies with the softmaxes innate properties which make it suitable for optimization with cross entropy loss.

As a matter of fact the softmax layer only exists in the training pipeline. Softmax scaling is not necessary for doing the predictions because the prediction of a classifier is the output node with highest activation value. Softmax is a monotonic function ie. higher input values will always give higher output values and since the classification is only about selecting the highest value, the softmax is not required. In addition the softmax can be computationally relatively expensive especially when the output size is large, as is the case with large lemma vocabulary.

To produce the actual lemmas and part of speeches, the indices of the highest values in the output layer need to be used for indexing the label vocabularies. If the third node on the lemma output layer has highest value, one must select third item in the lemma vocabulary as the final prediction. Vocabularies are not needed for testing and validating because the ground truth lemmas and part of speeches have also been turned into indices and comparison can be done with the indices.

3.1.5 Training and Optimization

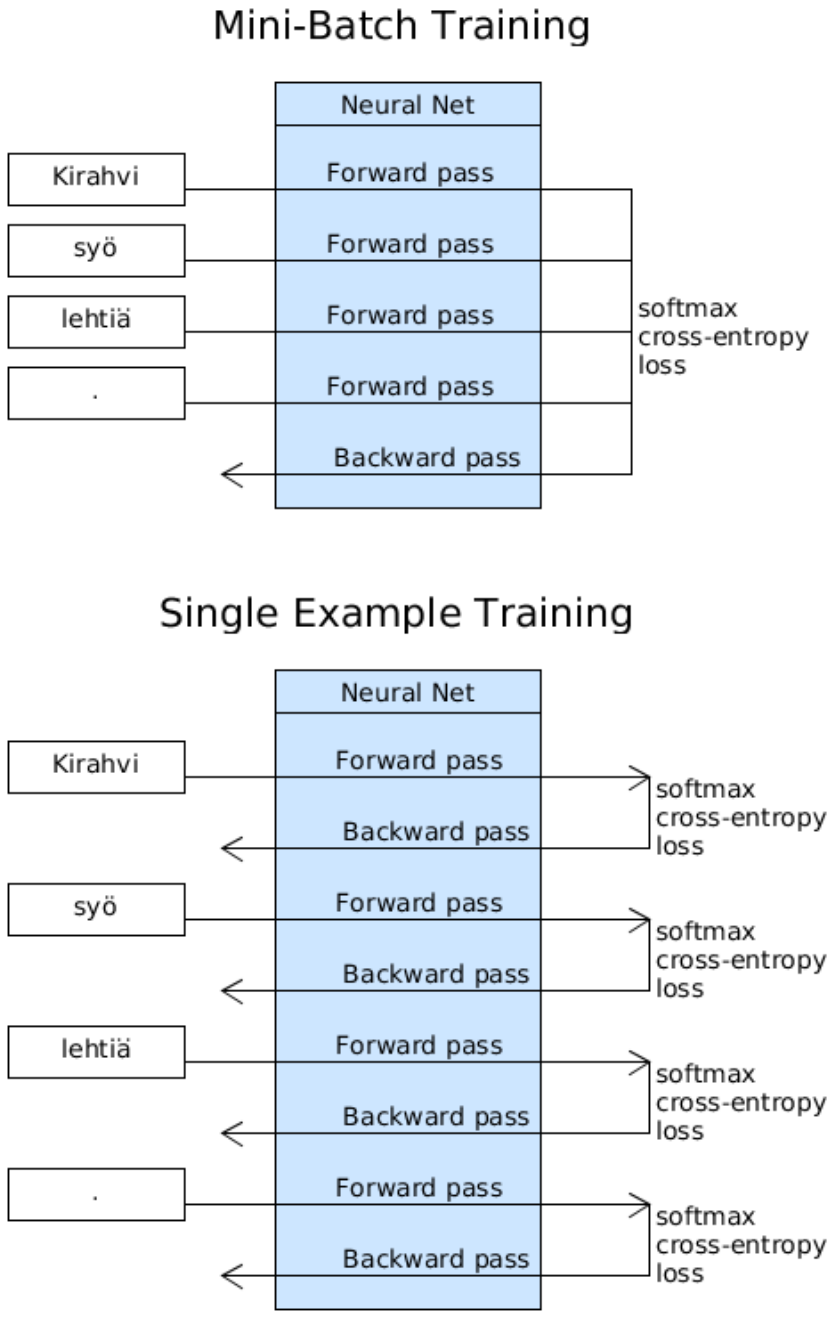
All neural network trainings in this work were done with stochastic gradient descent back-propagation algorigthm. The optmizer algorithm and back-propagation through the computation graphs are handled by Tensorflow's built in features. First the input data is fed to neural network and propagated in forward direction through the whole network. When the last network layer, the softmax layer, has been computed, the softmax values are used to compute cross entropy loss with one-hot coded ground truth vector. Error gradient is then propagated through the entire network in backward direction, calculating error gradient for each layer. Error gradient in each layer is used to update the layer weights and biases.

Gradient descent update was done with mini-batches of 25 sentences. This means that entire mini-batch is back-propagated through the network before adjusting the network weights and biases. Alternative is to update the network after each example but that has serious drawbacks for computational efficiency. Mini-batch training can be done in a pleasantly parallel manner, meaning that the exapmles of the mini-batch have no cross-dependencies and thus can be all trained at the same time, parallel. Parallel computations are very suitable for GPU workloads and as a matter of fact mini-batch training is critical component for enabling the significant, often an order of magnitude or more, increases in computation speed on GPU. Figure 3.4 illustrates the difference between the propagation sequences in single example training and mini-batch training.

Also a true single example training is not even possible, at minimum a mini-batch contains all the words of a single sentence. The reasons for this are the facts that the single example of input-output value pair is a single word and it's lemma of POS-tag and a word depends on it's context ie. other words in the sentence. Therefore it's mandatory to use at least a single sentence mini-batching. If architecture didn't contain the context encoding component, all words of the sentence could be managed as separate units and could be trained independently.

Mini-batches of 25 sentences used for network optimization are formed by dividing

Figure 3.4 *Propagation sequences in single exmaple and mini-batch training*



the entire training dataset randomly into the said mini-batches. Training procedure goes through all the mini-batches in an epoch. In the end of the epoch after all of the training data has been used, the network classification performance is evaluated on the validation dataset by calculating loss. Loss is the compared to the best obtained loss value from all the previous epochs, if the validation loss has decreased training procedure continues onto the next epoch. If validation loss doesn't decrease training can be stopped to avoid over-fitting. However classification performance on validation dataset does not typically improve on every epoch, especially after the network is starting to converge. Therefore it's wise to allow the training to continue for a fixed number of epochs even if there is no improvement. When the maximum number of epochs without improvement has been reached, the training is stopped and classification performance is tested on the testing dataset.

The network training procedure described above is only a single training run. Single training run allows the optimization of network parameters (weights and biases) but not hyperparameters. Hyperparameters are all the parameters that determine the network architecture, such as number of layers on each component, number of nodes on each layer, activation functions used, dropout etc. Traditionally hyperparameters have been optimized by manual process of training with certain cofiguration, inspecting the results, adjusting hyperparameters and training again. While manual hyperparameter optimization can provide good enough results, it is still very tedious work and often requires deep understanding of neural networks and of the task at hand. Recently automatic hyperparameter optimization has gained popularity, mainly because computation resources have increased and thus it's more feasible to run a lot of training runs while tuning the hyperparameters. Automatic hyperparameter tuning is particularly beneficial when the number of hyperparameters to optmize gets large, humans are typically poor at processing more than 3 dimensional data.

Hyperparameter optimization in this work was done automatically using a optimization library called Optunity TODO: citation. Optunity provides a convenient way to abstract the optimization work for basically any given task. There are many different optimization algorithms for hyperparameter tuning and not one has gained similar de facto status as gradient descent has for the neural net optimization. Algorithm used in this work for hyperparameter tuning is particle swarm which is a meta heuristic optimization algorithm. Particle swarm is said by the Optunity authors to be most versatile of all the algorithms available in Optunity. Hyperparameter optimization algorithm comparisons were not done since particle swarm provided reasonable results with the time results of this project. Besides hyperparameter optimization algorithm selection is not in the scope of this work and has very little

relevance with proving the hypothesis.

Hyperparameter optimization was done by running a fixed number of training runs and tuning the hyperparameters with particle swarm between each training run. Since automatic hyperparameter selection may ramp up all the hyperparameters to their maximum allowed values, can training times scale up to unsustainable durations. To avoid dealing with single training runs that take tens of hours, maximum time budget was forced for the training runs. If training run reached the maximum allowed time, best model obtained with that training run was selected for final comparison and hyperparameter tuning. Training runs in the hyperparameter optimization also used early stopping based on validation loss, this helps to cut the training with some hyperparameter configurations to just few training epochs, speeding up the hyperparameter optimization drastically.

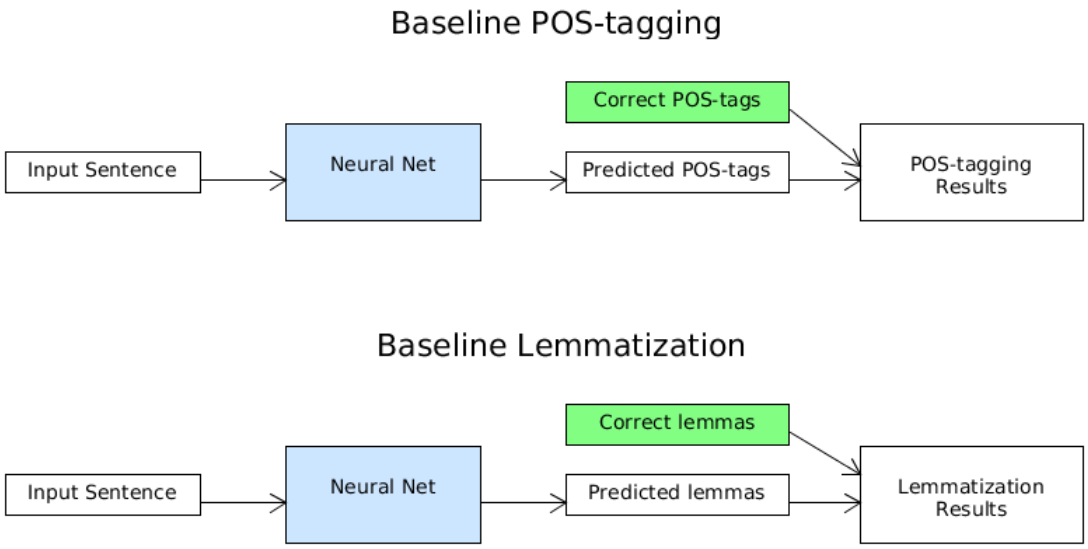
After all of the training runs for hyperparameter optimization were finished, hyperparameter configuration which provided the best model was saved for using in experiments.

3.2 Experiments

Experiments done for this thesis consist of minimal set of tests that can prove or disprove the tested hypothesis. To test whether jointly learning lemmatization and POS-tagging in a single neural network architecture the following experiments were done. Firstly a baseline was established by training and testing the neural net with lemmatization and POS-tagging as separate tasks. Having a well established baseline allows the comparison of different style shared information tasks and their benefits and drawbacks. Figure 3.5 shows the setup for separate tasks.

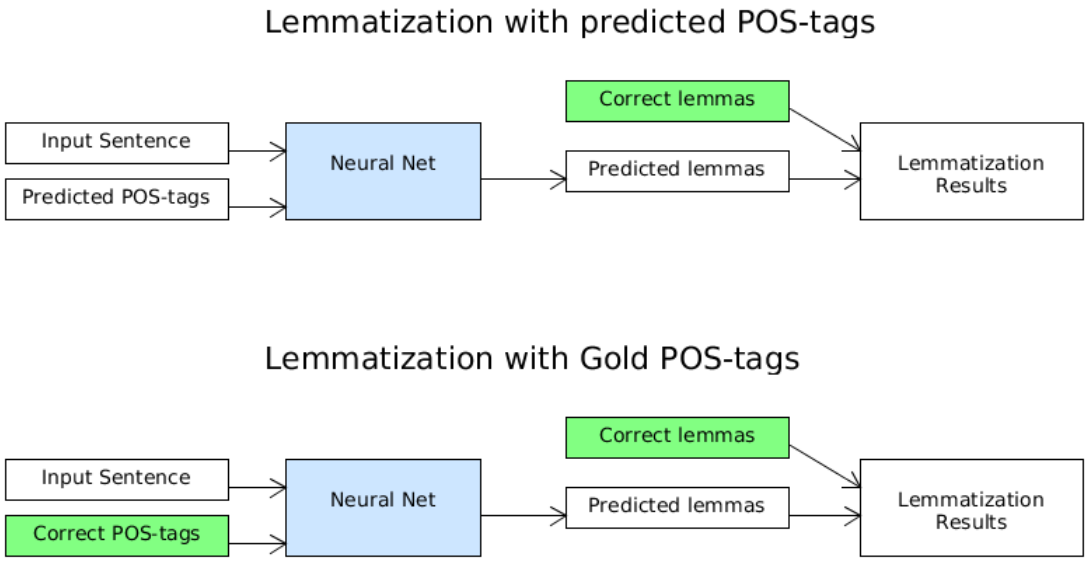
First shared information experiment was done to determine if more traditional approach of using one as input features in classification of other. This experiment was done only by using parts of speech as one-hot coded input features when doing lemma classification. Expecting to gain benefit from using parts of speech as input feature to lemmatization is reasonably well founded because knowing the part of speech for a given word limits the number of possible lemmas significantly. The big question in this hypothesis is whether the POS-tagging was done well enough to reveal the possible benefits. It's very well possible that errors done in POS-tagging cascade to lemmatization task so badly that lemmatization performance actually decreases from the baseline. To study the effects of POS-tagging performance to lemmatization performance, a second experiment was deviced: to train and test the network using gold-standard parts of speech available in the dataset. Figure 3.6

Figure 3.5 *Separate lemmatization and POS-tagging as a baseline*



shows the setup for using parts of speech as input feature for lemmatization.

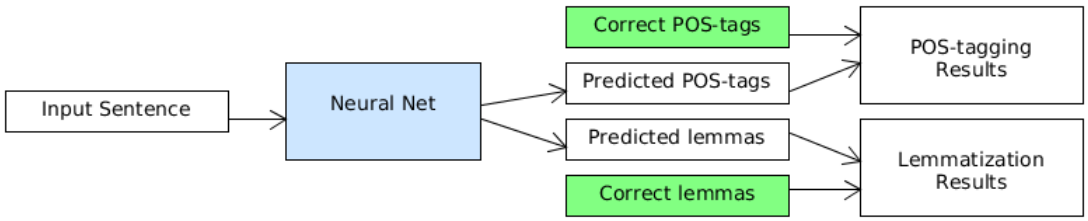
Figure 3.6 *Using parts of speech as input feature for lemmatization*



Last experiment was the training and testing of the joint learning neural network architecture for lemmatization and POS-tagging. Figure 3.7 illustrates the test setup for joint model.

Comparing results from the baseline, POS-tags as input features for lemmatization and joint model should be enough to test the hypothesis. Experiments don't include other languages than Finnish or other datasets than Finnish Universal Dependencies.

Figure 3.7 *Joint model for lemmatization and POS-tagging*



Experiments in this thesis with POS-tagging all use the universal POS-tags, effect of using language specific POS-tags with or without universal POS-tags was not tested. Similarly effect of tokenization performance was not tested but all experiments used gold standard tokenization available in the Finnish universal dependencies dataset.

The effect of morphological information such as word inflections and forms were not tested for this thesis even though there is a well founded arguments for improving results by using the morphology also. Since morphological features describe how the word was inflected they should assist the lemmatization because lemmatization can though as a reverse process of inflecting the word. However scope of the thesis was kept as narrow as possible, but wide enough for testing the hypothesis, and therefore morphology was left out. For that same reason also use of language specific POS-tags available in Universal Dependecies datasets was left out of scope of this thesis.

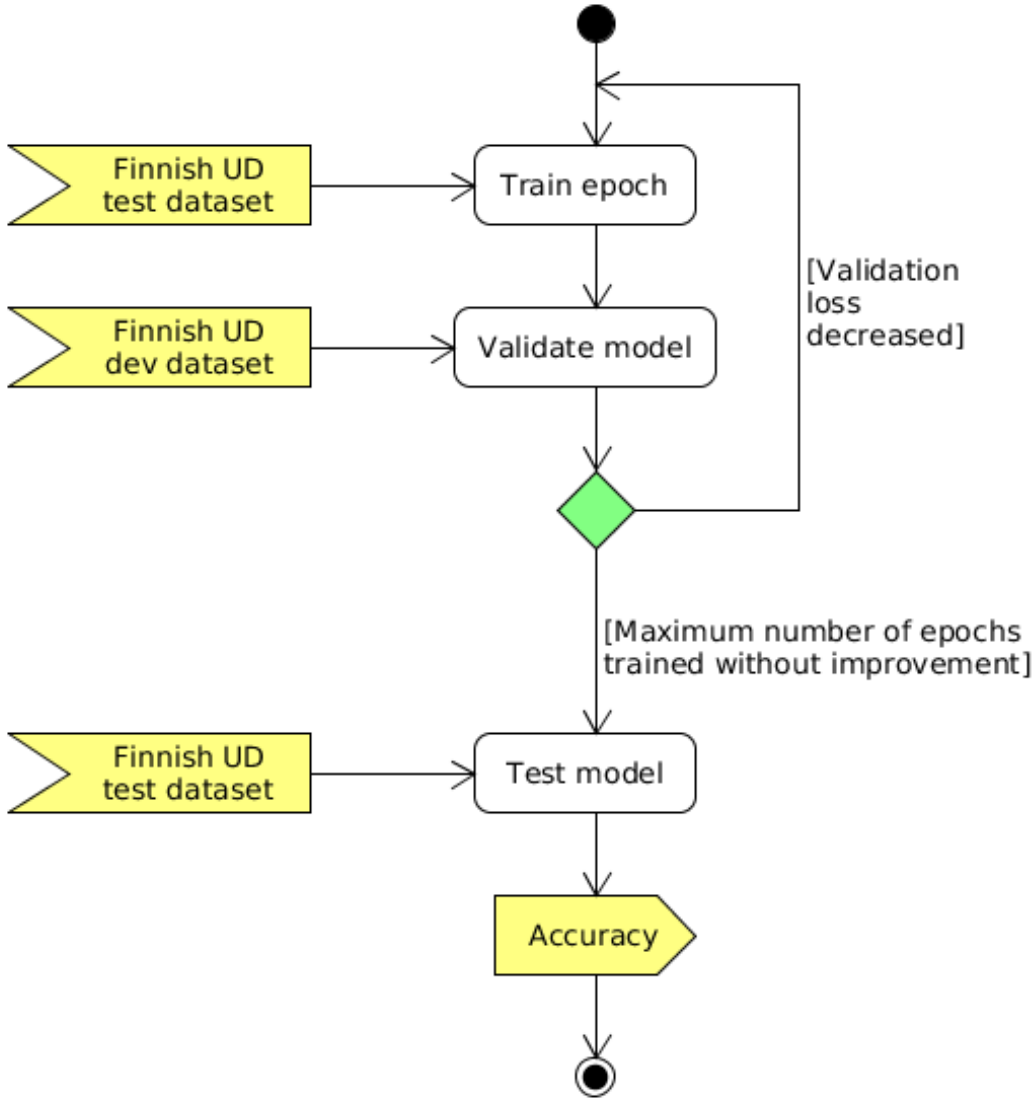
Attentive reader might have noticed that experiments included in this thesis have parts of speech as input features for lemmatization but not lemmas as input features for POS-tagging. Reason for excluding experiment of lemmas as input features for POS-tagging is that the described task is not very interesting or even meaningful: if a lemma of a word is known, part of speech for that word can be looked up from a dictionary with the exceptions of words with multiple meanings.

3.3 Test Methods

The dataset use for experiments in this thesis is Finnish Universal Dependencies as mentioned in earlier sections. The Finnish Universal Dependencies dataset is divided into three sections: training dataset, development dataset (also called validation dataset in this thesis) and testing dataset. Each experiment outlined in section 3.2 was done using all three datasets. Neural network weight and bias optimization was done using only training dataset, model performance was observed during training by validating the network with development dataset and all final results are obtained by doing predictions with test dataset and comparing with correct labels. Training

and testing procedure for each experiment is illustrated in figure 3.8.

Figure 3.8 *Test setup for doing the experiments*



Other corpuses exist for Finnish but format and annotation schema varies in those corpuses and are only available for Finnish. Univesal Dependencies is very suitable this work because it has over 40 different languages available all with same form and annotation scheme making possible future expansions of this study easier and results comparable. Another reason for selecting Universal Dependencies is that pre-existing results are avaialble for POS-tagging and lemmatization, most notably from UDPipe (Straka and Straková 2017).

Several options exist for the metric with which experiment results can be compared when testing the hypothesis. This work settled for simple accuracy number ie. per-

centage of correct prediction over all examples in the testing dataset. F1 score is another widly used metric which takes precision and recall into account and therefore reveals the performance of the model better. However F1 score may not be intuitive and easy to understand for a reader without prior experience or education in statistical analysis. Accuracy should also provide enough information to test the hypothesis.

Another drawback of accuracy is that it tells nothing about the confidence of the network; even if model's accuracy is high, the model might give very high probabilities for the few incorrect predictions it produces. If output probabilities are required for the task, for example for using a probability threshold for determining if an answer should be given at all, this kind of model is not very suitable for the task. In this work however the training is done based on the softmax cross-entropy loss which optimizes the form of prediction probability distribution. Also the early stopping used in the model training is based on the cross-entropy loss so training is stopped before network starts to be overconfident about it's incorrect predictions. Softmax cross-entropy is a good metric for optimzing the network, but using cross-entropy loss for evaluating performance of different results suffers from the same problem as F1 score and provides even less insight for the reader about how well the model performs.

Given all of these considerations for different benefits and drawbacks of the discussed metrics, it is still most probable that any of them would suffice to reveal the classification performance differences between the experiments. Some of the metrics might emphasis the difference more than the others, but the bottom line is to test the hypothesis and provide results for the reader that are understood by the reader.

3.4 Results

- UDPipe 1.2 achieves 94,9% on POS-tagging on Finnish Universal Dependencies 2.0 and 86,8% on lemmatization. Both results with gold tokenization. Straka 2017
- UDPipe lemmatizer is a generative model (Straka and Straková 2017), so comparison is not fair or even meaningful.
- POS-tagging without lemmatization = 94,30%. Remarkably close to UDPipe results with only 0,6%-points advantage to UDPipe. Limited mainly by data?
- Lemmatization without POS-tagging = 94,25%

- Lemmatization with classified POS-tags as input features = 94,40%
- Lemmatization with gold standard POS-tags as input features = 96,22%
- Joint model. Lemmas = 95,24%, POS = 94,14%
- Joint model achieves best practical results for lemmatization, with 0,99%-points increase, since gold-standard POS-tags are not available for live inferencing.
- Joint model POS-tags are slightly inferior (0,16%-points) to baseline
- Lemmatization results with classified POS-tags are slightly better (0,15%-points) than the baseline.
- Lemmatization with gold-standard POS-tags achieves clearly the best results with 1,97%-points increase over baseline.

4. DISCUSSION

4.1 How well results generalize?

- Do these results generalize to other languages needs to be verified with additional experiments.
- Universal Dependencies datasets are created to provide as uniform language modeling as possible across all languages and therefore it is probable that the same mutual information between lemmas and POS-tags in other languages would prove beneficial in joint learning of lemma and POS-tag classification.
- Speculating how well these results generalize to other tasks in natural language processing is more complicated because other tasks may not share as strong mutual information as lemmas and POS-tags do.
- Liang et al. 2016 proved that neural networks benefit from jointly learning two tasks different from lemmatization and POS-tagging. Liu and Lane 2016 even had tasks on different architectural level, slot filling being word level task and intent classification being sentence or message level task.
- With results provided in this work and results from Liu and Lane 2016 it's failry safe to say that neural networks can obtain better results when learning jointly two tasks that share mutual information.

4.2 What was assumed?

- It was assumed that training dataset would represent Finnish language well. May not hold true since all sources of the dataset are from internet.
- It was also assumed that different datasets would have fairly similar data distribution and training done on training dataset would generalize to test dataset.
- It was noticed that selecting lemma vocabulary which covers 100% of uses in training set only covers about 75% of uses in validation and test datasets.

- Lemma classifier learned remarkably well to classifiy lemmas not seen during training time as unknown tokens. Maybe priori bias explains this?

4.3 What was simplified?

- Slightly different results would be optained by optimizing hyperparameters separately for every different experiement. Hyperparameter optimization requires about 100 to 200 training runs and as such takes significant time budget. Using time required for hyperparameter optimization before every experiement proved inconvenient for this work.
- Hyperparamenter optimization do not have variation with loss weights for the two tasks. Loss weight sets the target if optimizing based on validation loss and therefore cannot be optimized.
- Hyperparameter optimization was using time budget for each run
- Hyperparameter optimization had limited number of runs available
- Selecting a different optimization target for hyperparameters could have allowed loss weight optimization.
- Character vocabualry for C2W does not contain all characters in datasets. Even still embedding vectors for some characters are probably not well learned because of their unfrequency.
- Lemmatization as word level classification task only is also a pretty significant simplification. Decoder model for lemma generation is able to always produce some kind of answer without ever producing unknown tokens.
- Selected lemma vocabulary covers 90% of the training set -> unknowns are 10%, second most frequent is less than 5%. It was not tested if unknown prediction accuracy would break if unknowns weren't the most frequent.
- Whether having unknown tokens is fatal flaw depends on the problem at hand and on the down stream processing which uses lemmas obtained with lemmatization. For some tasks not having generated lemmas with high probability of typographical errors produced might be significantly better option.
- Experiements done with golds standard tokenization of Universal Dependencies datasets. In live production such tokenization is not available. Imperfect tokenization may completely ruin the classification if tokens are not created correctly.
- Same applies for sentence segmentation.

5. CONCLUSIONS

- Hypothesis proved to be true, at least partly
- Neural networks are very suitable for joint learning tasks.
- Uncorrelated tasks are not tested yet but those don't have any obvious theoretical foundation why they should benefit.
- Interestingly we couldn't obtain increase in POS-tagging performance.
- Maybe lemmatization as a more information rich task dominates the learning
- POS-tagging on joint model could be improved by giving more weight to POS-tagging task but that quickly deteriorates the lemmatization perfomance drastically.
- Doing neural net based lemmatization benefits from jointly learning also to POS-tag. Added architectural, implementational and computational complexities are minor.
- Author recommends always to do POS-tagging when doing lemmatization.
- If main task is POS-tagging, adding lemmatization is probably not beneficial enough for the added computational cost.
- If both tasks need to be done and POS-tagging performance is a priority it might prove to be difficult to achieve best possible results with joint model.

BIBLIOGRAPHY

Andor, D. et al. (2016). "Globally Normalized Transition-Based Neural Networks". In: *Acl 2016*, pp. 2442–2452. DOI: 10.18653/v1/P16-1231. arXiv: arXiv:1603.06042v2.

Bahdanau, D., K. Cho, and Y. Bengio (2014). "Neural Machine Translation By Jointly Learning To Align and Translate". In: *Iclr 2015*, pp. 1–15. ISSN: 0147-006X. DOI: 10.1146/annurev.neuro.26.041002.131047. arXiv: 1409.0473. URL: http://arxiv.org/abs/1409.0473v3.

Bhargava, A. et al. (2013). "Easy contextual intent prediction and slot detection". In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, pp. 8337–8341. ISSN: 15206149. DOI: 10.1109/ICASSP.2013.6639291.

Brill, E. (1992). "A Simple Rule-Based Part of Speech Tagger". In: *Applied natural language*, p. 3. ISSN: 00992399. DOI: 10.3115/1075527.1075553. arXiv: 9406010 [cmp-lg].

Chen, D. and C. D. 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)* i, pp. 740–750. ISSN: 9781937284961. URL: https://cs.stanford.edu/%7B~%7Ddanqi/papers/emnlp2014.pdf.

Cho, K. et al. (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734. ISSN: 09205691. DOI: 10.3115/v1/D14-1179. arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

Chung, J., K. Cho, and Y. Bengio (2016). "A Character-level Decoder without Explicit Segmentation for Neural Machine Translation". In: *Acl-2016*, pp. 1693–1703. arXiv: 1603.06147.

Clevert, D.-A., T. Unterthiner, and S. Hochreiter (2015). "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: *Under review of ICLR2016 ELU* 1997, pp. 1–13. arXiv: 1511.07289. URL: http://arxiv.org/pdf/1511.07289.pdf%7B%5C%%7D5Cnhttp://arxiv.org/abs/1511.07289.

Collobert, R. et al. (2011). "Natural Language Processing (Almost) from Scratch". In: *Journal of Machine Learning Research* 12, pp. 2493–2537. ISSN: 0891-2017. DOI: 10.1.1.231.4614. arXiv: 1103.0398.

De Marneffe, M.-C., B. MacCartney, and C. D. Manning (2006). "Generating typed dependency parses from phrase structure parses". In: *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*,

pp. 449–454. DOI: 10.1.1.74.3875. URL: http://nlp.stanford.edu/pubs/LREC06%7B%5C_%7Ddependencies.pdf.

Fernández-González, D. and A. F. T. Martins (2015). "Parsing as Reduction". In: *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)* 1983, pp. 1523–1533. arXiv: 1503.00030.

Harris, Z. S. (1954). "Distributional Structure". In: *WORD* 10.2-3, pp. 146–162. DOI: 10.1080/00437956.1954.11659520. URL: http://dx.doi.org/10.1080/00437956.1954.11659520.

Haverinen, K. et al. (2014). "Building the essential resources for Finnish: the Turku Dependency Treebank". In: *Language Resources and Evaluation* 48.3, pp. 493–531. ISSN: 15728412. DOI: 10.1007/s10579-013-9244-1.

Kestemont, M. et al. (2016). "Lemmatization for variation-rich languages using deep learning". In: *Digital Scholarship in the Humanities*, fqw034. ISSN: 2055-7671. DOI: 10.1093/llc/fqw034. URL: http://dsh.oxfordjournals.org/lookup/doi/10.1093/llc/fqw034.

Kim, Y. et al. (2016). "Character-Aware Neural Language Models". In: *Aaai*. arXiv: 1508.06615. URL: http://arxiv.org/abs/1508.06615.

Korenius, T. et al. (2004). "Stemming and lemmatization in the clustering of finnish text documents". In: *Proceedings of the thirteenth ACM conference on information and knowledge management*, pp. 625–633. DOI: 10.1145/1031171.1031285. URL: http://portal.acm.org/citation.cfm?id=1031171.1031285%7B%5C&%7Dcoll=Portal%7B%5C&%7Ddl=ACM%7B%5C&%7DCFID=88534260%7B%5C&%7DCFTOKEN=49348956.

Liang, C. et al. (2016). "Neural Symbolic Machines: Learning Semantic Parsers on Freebase with Weak Supervision". In: October. arXiv: 1611.00020. URL: http://arxiv.org/abs/1611.00020.

Ling, W. et al. (2015). "Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation". In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing* September, pp. 1520–1530. DOI: 10.18653/v1/D15-1176. arXiv: 1508.02096. URL: http://dx.doi.org/10.18653/v1/d15-1176%7B%5C$%7D%7B%5C%%7D5C%7B%5C$%7Dnfile:///Files/68/6810072d-e133-426e-807f-445df2840420.pdf%7B%5C$%7D%7B%5C%%7D5C%7B%5C$%7Dnpapers3://publication/doi/10.18653/v1/d15-1176%7B%5C$%7D%7B%5C%%7D5C%7B%5C$%7Dnhttp://arxiv.org/abs/1508.02096.

Liu, B. and I. Lane (2016). "Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling". In: 1, pp. 2–6. DOI: 10.21437/Interspeech.2016-1352. arXiv: 1609.01454. URL: http://arxiv.org/abs/1609.01454.

Mikolov, T., G. Corrado, et al. (2013). "Efficient Estimation of Word Representations in Vector Space". In: *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, pp. 1–12. ISSN: 15324435. DOI: 10.1162/153244303322533223. arXiv: arXiv:1301.3781v3. URL: http://arxiv.org/pdf/1301.3781v3.pdf.

Mikolov, T., W.-t. Yih, and G. Zweig (2013). "Linguistic regularities in continuous space word representations". In: *Proceedings of NAACL-HLT* June, pp. 746–751. URL: http://scholar.google.com/scholar?hl=en%7B%5C&%7DbtnG=Search%7B%5C&%7Dq=intitle:Linguistic+Regularities+in+Continuous+Space+Word+Representations%7B%5C#%7D0%7B%5C%%7D5Cnhttps://www.aclweb.org/anthology/N/N13/N13-1090.pdf.

Nivre, J. (2004). "Incrementality in deterministic dependency parsing". In: *Proceedings of the Workshop on Incremental Parsing: Bringing Engineering and Cognition Together*, pp. 50–57. DOI: 10.3115/1613148.1613156. URL: http://dl.acm.org/citation.cfm?id=1613156.

Nivre, J. et al. (2016). "Universal Dependencies v1: A Multilingual Treebank Collection". In: *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, pp. 1659–1666.

Pennington, J., R. Socher, and C. D. Manning (2014). "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pp. 1532–1543. ISSN: 10495258. DOI: 10.3115/v1/D14-1162. arXiv: 1504.06654.

Petrov, S., D. Das, and R. Mcdonald (2012). "A Universal Part-of-Speech Tagset". In: arXiv: 1104.2086.

Pyysalo, S. et al. (2015). "Universal Dependencies for Finnish". In: *Nordic Conference of Computational Linguistics NODALIDA 2015* Nodalida, p. 163.

Straka, M. (2017). *UDPipe User's Manual*. URL: http://ufal.mff.cuni.cz/udpipe/users-manual (visited on 10/01/2017).

Straka, M. and J. Straková (2017). "Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe". In: *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies* 2, pp. 88–99. URL: http://www.aclweb.org/anthology/K17-3009.

Sutskever, I., O. Vinyals, and Q. V. Le (2014). "Sequence to sequence learning with neural networks". In: *Nips*, pp. 1–9. ISSN: 09205691. DOI: 10.1007/s10107-014-0839-0. arXiv: 1409.3215. URL: http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural.

Takala, P. (2016). "Word Embeddings for Morphologically Rich Languages". In: April, pp. 27–29.

Weiss, D. et al. (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)* 2012, pp. 323–333. DOI: 10.3115/v1/P15-1032. arXiv: 1506.06158. URL: http://www.aclweb.org/anthology/P15-1032.

Xu, P. and R. Sarikaya (2013). "Exploiting shared information for multi-intent natural language sentence classification". In: *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH* 1, pp. 3785–3789. ISSN: 19909772.

Zhang, X. and Y. LeCun (2015). "Text Understanding from Scratch". In: *APL Materials* 3.1, p. 011102. ISSN: 2166-532X. DOI: 10.1063/1.4906785. arXiv: 1502.01710. URL: http://arxiv.org/abs/1502.01710.

APPENDIX A. SOMETHING EXTRA

Appendices are purely optional. All appendices must be referred to in the body text

APPENDIX B. SOMETHING COMPLETELY DIFFERENT

You can append to your thesis, for example, lengthy mathematical derivations, an important algorithm in a programming language, input and output listings, an extract of a standard relating to your thesis, a user manual, empirical knowledge produced while preparing the thesis, the results of a survey, lists, pictures, drawings, maps, complex charts (conceptual schema, circuit diagrams, structure charts) and so on.