TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JAAKKO PASANEN

NATURAL LANGUAGE SYNTACTIC PARSING WITH NEURAL NETWORKS

Master of Science thesis

Examiner: Prof. Ari Visa
Examiner and topic approved on
1st November 2017

ABSTRACT

JAAKKO PASANEN: Natural Language Syntactic Parsing with Neural Networks
Tampere University of Technology
Master of Science thesis, 43 pages, 0 Appendix pages
November 2017
Master's Degree Programme in Automation Technology
Major: Learning and Intelligent Systems
Examiner: Prof. Ari Visa
Keywords: Natural Language Processing, Syntactic Parsing, Neural Networks, Deep Learning

Conversational user interfaces have made strong appearance during the last couple of years. Most growth with conversational UIs can be seen with customer service moving from phone to chat. As big as the hype surrounding the conversational user interfaces is, they often cannot surpass traditional alternatives for the use cases they are used without actually understanding the language user speaks. This has crated a large demand for artificial intelligence (AI) which can understand users in their natural language.

Natural language used by humans is tremendously complex without people actually realizing that. Understanding something this complex often requires system which divides the problem into smaller sub-problems and tries to tackle those, a divide and conquer paradigm. Natural language processing tools are often built as pipelines where more information is mined from the text in each step. Finding syntactic features, such as part of speech and lemma, is one such step, and is the focus of this thesis.

Main objective for this thesis was to build a neural network architecture which can classify lemmas and parts of speech for the input text. Research hypothesis was then to determine if such architecture could be modified to do the both tasks at the same time and if such change would improve classification performance of the model.

Doing experiments with Finnish Universal Dependencies dataset revealed that lemmatization benefits from jointly learning to POS-tag, but POS-tagging performance could not be improved. Best absolute lemmatization results were gained by using correct POS-tags as input features, but since they are not available for live predictions the result has no practical meaning.

TIIVISTELMÄ

JAAKKO PASANEN: Luonnollisen kielen syntaksin parsiminen neuroverkoilla
Tampereen teknillinen yliopisto
Diplomityö, 43 sivua, 0 liitesivua
Marraskuu 2017
Automaatiotekniikan koulutusohjelma
Pääaine: Oppivat ja älykkäät järjestelmät
Tarkastajat: Prof. Ari Visa
Avainsanat: Luonnollisen kielen käsittely, syntaksin parsiminen, neuroverkot, syväoppiminen

Keskustelevat käyttöliittymät ovat nousseet vahvasti perinteisten käyttöliittymien rinnalle viimeisen parin vuoden aikana. Suurin kasvu tällä hetkellä on nähtävissä asiakaspalvelun siirtymisessä puhelimesta chattiin. Vaikka hype keskustelevien käyttöliittymien ympärillä on valtavaa, eivät ne useinkaan pysty päihittämään perinteisiä käyttöliittymiä ilman että käyttäjää ymmärretään tämän omalla kielellä. Tästä on syntynyt suuri kysyntä tekoälylle joka ymmärtää käyttäjiä heidän luonnollisella kielellään.

Ihmisten käyttämä luonnollinen kieli on valtavan monimutkaista ilman että ihmiset useinkaan ymmärtävät sitä. Näin monimutkaisen asian ymmärtäminen usin vaatii että ongelma jaetaan pienenpiin osaongelmiin ja pyritään ratkaisemaan ne. Työkalut luonnollisen kielen käsittelyyn rakennetaan usein sarjaksi askelia joista jokainen kaivaa lisää informaatiota tekstistä. Syntaktisten piirteiden, kuten sanaluokkien ja perusmuotojen, etsiminen on yksi tällainen askel ja onkin tämän opinnäytetyön painopiste.

Ensisijainen tavoite tälle opinnäytetyölle oli rakentaa neuroverkkoarkkitehtuuri joka pystyy luokittelemaan sanoille sanaluokat ja perusmuodot. Tutkimushypoteesi oli selvittää voiko arkkitehtuurin muokata sellaiseksi että se suorittaa molemmat tehtävät samanaikaisesti ja saavutetaanko muutoksella parannus verkon suorituskykyyn.

Suomenkielisellä Universal Dependencies datasetillä tehdyt kokeet paljastivat että perusmuotojen luokittelu hyötyy samanaikaisesta sanaluokkien luokittelusta, mutta kuitenkaan suorituskykyä sanaluokkien luokitteluun ei pystytty parantamaan. Paras absoluuttinen tulos perusmuotojen luokittelulle saavutettiin käyttämällä oikeita sanaluokkia lisäpiirteenä verkolle, mutta koska täysin oikeita sanaluokkia ei ole käytettävissä todellisessa tilanteessa ei tällä tuloksella ole juuri käytännön merkitystä.

PREFACE

This thesis was done at ultimate.ai, a customer service software company, as part of ongoing research into natural language understanding during 2017.

I would like to thank my examiner professor Ari Visa for his guidance and valuable feedback in formulating the hypothesis, selecting thesis scope and writing process.

Tampere, 13th November 2017

Jaakko Pasanen

CONTENTS

LIST OF ABBREVIATIONS AND SYMBOLS

ANN	Artificial Neural Network
CPU	Central Processing Unit
GPU	Graphics Processing Unit
MLP	Multi-layer perceptron
NLP	Natural Language Processing
POS	Part-of-speech; also called lexical category
RNN	Recurrent neural network
UI	User Interface

1. INTRODUCTION

Understanding human language is perhaps the most rewarding problem to solve in artificial intelligence and machine learning. In the era of the internet almost all information is available anywhere, anytime. The problem with information available in the internet is that it is in largely machine incomprehensible format, natural human language. The prospect and implications of having such computer system which understands human language and can thus process the enormous knowledge quantity available from the internet are virtually world changing. In addition to huge publicly available information encoded in natural language, the internet also holds and delivers private information between organizations and individuals. Discovering a system which can process and understand the public open domain information from internet is still probably several years if not decades away. However starting from more intimate and well scoped problem domains is providing a platform for creating value by natural language understanding systems.

One of the better channels for tapping into that intimate knowledge exchange is the recently popularized conversational user interfaces (UIs). Chat as a personal communication channel has been around for very long time, one of the first multi-user chat channels working over internet was the IRC, developed in 1988 and still used by some enthusiasts. Even in times when chat had not found a way into lives of vast majority first bots were written in IRC which could do certain tasks or provide replies mostly based on simple keywords in the processed messages. Long time has passed since the glory days IRC and chats have finally become inseparable part of most peoples lives. The change started with the emergence of smartphones after the introduction of first Apple iPhone in 2007 and today most of the remote communication between, at least younger, people is handled over various chat applications.

China had been in the leading edge for consumer to corporate communications in chat applications with the massive growth of WeChat application. Western markets are only now starting to heat up for the conversational user interfaces which have been a popular way for companies to communicate with their customers since the introduction of WeChat in 2011. No WeChat like application has gained similar foothold in wester markets, but conversational user interfaces are growing rapidly

with customer service. Customer service is rapidly moving from calls by phone to a chat served on the internet or mobile application. The hype for conversational interfaces is so big that some parties are implementing them with little consideration for their suitability for the use case. The conversational UI is rarely better option compared to traditional UIs unless the user can use it with her own language. Command-line interfaces which can be considered as sub-type of conversational UIs have been around for a very long time but have always been an efficient tool for experienced user, not very suitable for occasional use in user friendly manner.

Natural language understanding with computer systems has improved to the point that it is starting to be feasible to develop an artificial intelligence that can serve the user without expressions of deep disdain from the user. However natural language is still enormously complex and requires the problem to be split into smaller sub-problems which are easier to manage, a divide and conquer paradigm. One of these sub-problems is the obtaining of syntactic features such as part of speech (POS) and lemma for a given word. Syntactic features convey a lot of information about the text and can therefore benefit various other tasks when used as input features for the methods.

The main objective of this thesis is to develop and test artificial neural network (ANN) model which is capable of finding parts of speech and lemmas for the input text. Research hypothesis is that such neural network architecture can be modified to learn and predict both features at the same time, without the need for two models, and that the joint model can outperform separate models in either or both of the two tasks. Several experiments were deviced and conducted to test the hypothesis.

Chapter 2 of this thesis goes through the problems associated with using text as input data for neural networks, and problems that arise from processing highly morphological languages such as Finnish. Syntactic parsing tasks most relevant to this thesis are introduced along with previous research done in attempts to solve them. Chapter 3 goes through the neural network architecture used, outlines the experiments to be conducted to test the hypothesis, presents the test setup for the experiments and displays the results obtained. Chapter 4 contains discussions about the results, their generalizibility as well as assumptions and simplifications made about data, test setup and results. Chapter 5 wraps up the work, presents conclusions and opportunities for future research.

2. NATURAL LANGUAGE PROCESSING

Natural language processing (NLP) is vastly wide problem field where different tasks differ from each other very much, really in some cases the only connecting feature is, as name suggests, the natural human language as input data. During the last few years the field has seen some very impressive improvements as well as new methods and techniques that have been able surpass previous state of the art results in very short time. Tasks vary from machine translation, which tries to translate text from language to another, to simpler problems such as sentiment analysis, which in turn tries to deduce if the text has positive or negative emotional load. With such diverse and broad field it is not meaningful, or even possible, to fit a cross section about the entire field in a humbly master's thesis. Therefore it was important to select narrow sub-field for this thesis and simply leave other tasks for other research opportunities.

This chapter focuses on outlying core tasks and problems associated with the objective and research problem of this thesis. As was stated in the introduction, the objective of this thesis is to develop a neural network architecture for lemmatization and POS-tagging using Finnish language. Therefore this section focuses on syntactic parsing of natural language and mainly to said POS-tagging and lemmatization tasks. Other major theme for this chapter is to go through the general problems with using text as input data for neural nets and more specifically problems associated with highly morphological languages such as Finnish.

Vocabulary explosion is a term used to describe a situation with some highly morphological languages where the number of possible unique written forms for the words is extremely high. Finnish suffers from this phenomenon very badly, there are about 2000 forms for a noun, 6000 for an adjective and a verb may have up to 12000 different inflectional forms (Korenius et al. 2004). Another contributing factor to vocabulary explosion for Finnish is the richness of compound words in the language. Two thirds of the words in Dictionary of Modern Standard Finnish are compound words (Korenius et al. 2004). But compound words do not end there, Finnish allows to form new compound words with relatively few limitations as to what words can be combined into a compound word. The lemma vocabulary formed for word2vec binary trained with Finnish Internet Parsebank (Kanerva et al. 2014)

contains some 1,7 million unique lemmas , words that are not inflected, majority of those are compound words. And even this massive lemma vocabulary does not provide good coverage even for Finnish Universal Dependencies which is only a bit over 200000 tokens (Haverinen et al. 2014) as is discussed in the section 3.1.1. If one tries to inflect all those millions of lemmas needed for proper coverage with thousands of different inflections, the vocabulary size would grow to several billions. It is obvious that vocabulary of this magnitude introduces problems which cannot be overcome by representing words as independent unique elements.

This section also takes a look at the current and previous states of available datasets and annotations that are and have been used for natural language processing tasks. Most notably Universal Dependencies is discussed as the first truly cross-lingual project for syntactic parsing datasets. As for Finnish, before the introduction of Turku Dependency Treebank, which was later translated to Universal Dependencies annotation scheme, there has not been many high quality datasets for syntactic parsing tasks. Dependency parsing especially has been unreachable for Finnish because of total lack of available corpuses for the task.

2.1 Feature Engingeering in NLP

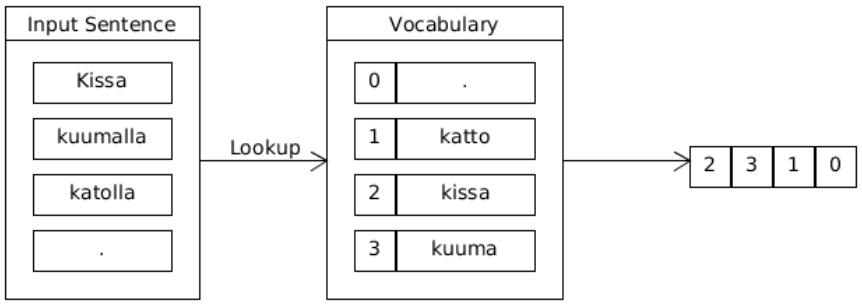
Machine learning methods, regardless if they are statistical models or neural network based, require text to be represented in some numerical form. Deep down all computer programs function with numbers only as processors arithmetic beings and as such cannot process anything but numbers. For most programming tasks the responsibility of representing text, character strings, lies with the programming language. Such is not the case with natural language tasks because, but the developer is responsible of turning text into numerical representation which conveys information about the text as is needed by the task at hand.

What information needs to be encoded into numerical representations is determined by the requirements of the natural language processing task. Traditionally machine learning methods for natural language processing includes hand engineered features. The researcher or developer uses her domain and task specific knowledge and experience in determining what features should be selected and how they should be represented. Selected features can be anything from independent words as indices to vocabulary to word counts in the sentence or the last morpheme of each word. Feature selection and engineering requires a lot of knowledge about the task and is usually tuned for each separate task. This makes machine learning methods developed to natural language processing very unique to each task making usage of one general architecture, with minor modifications, for multiple tasks impossible.

Additionally generating such features may hog up to 95% of computation time used for the task as was found by Chen and Manning 2014 in their work for dependency parsing. Feature engineering is something we can never get completely rid of because something always has to be represented, but it is possible, and even desirable, to strive for more general features.

Two simplest feature representation schemas that are usually figured out by a new developer approaching natural language processing are indices and one hot coding. Index representation can distinguish the represented features, whether they are characters, words or morphemes, from each other and this way provide a way for the mathematical method to use the input features. Index representation is nice in it's simplicity but the drawback is that the indices them selves cannot encode any additional information or meaning about the feature. Simplicity comes with the benefit of being computationally cheap, enabling use of massive datasets. Usually simple method trained with huge amounts of data outperforms more complex model trained with less data (Mikolov, Corrado, et al. 2013). The dataset assumption is of course only true for unsupervised learning, practically manually annotated corpuses with several billions of tokens do not exist. Step up from the index representation is one hot coding, also called one-of-V coding, which represents features as binary vectors where only the current feature is one and all others are zeros. This has the potential benefit of being better at separating the unique labels as each vector is equal in it's size and variation. Figures 2.1 and 2.2 illustrate the two basic feature representation schemes index representation and one hot coding respectively used for words.

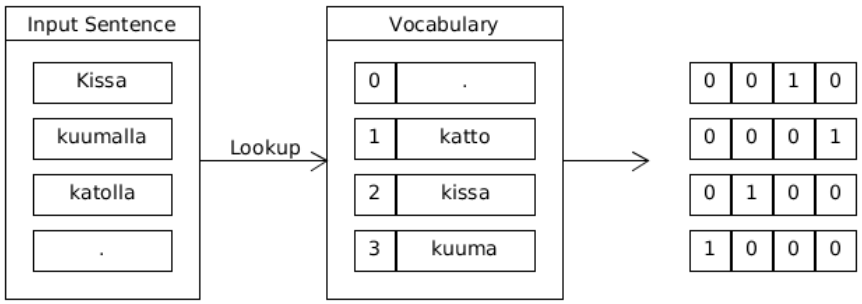
Figure 2.1 *Index representation for words*



2.2 Word Embeddings

Using words as the engineered features for natural language processing tasks is at the same time intuitive and natural choice because words are the basic units of meaning

Figure 2.2 *One hot coding representation for words*



for human languages. Therefore it's not surprising that a lot of research has been conducted into representing words as numerical vectors for machine learning methods. Numerical representation for words is also called word embedding (Mikolov, Yih, et al. 2013), and comes from the idea of embedding additional information about the word into the vector representation.

History of word embeddings goes long way back, at least 30 years. Some earlier experiments with word embeddings were done by Hinton, McClelland, et al. 1990, Hinton, Rumelhart, et al. 1985 and Elman 1990. Creating the word representations has since stabilized into two model families, which are global matrix factorization methods and local context window methods. Global matrix factorizers work on global corpus level and are based on co-occurrence counts in the corpus and are therefore good at representing the statistical properties of the words (Pennington et al. 2014). Local window methods omit the corpus level statistics since they operate only in the small window of local words and as such are good at word analogy tasks but poor at representing statistical features (Pennington et al. 2014). Local window methods try to create a vector space for the words such that semantically similar words are close to each other (Mikolov, Yih, et al. 2013), the semantic representations explain their good performance on word analogy tasks.

All word embedding methods, whether they are global matrix factorization based or local window based, generalize poorly with morphologically rich languages such as Finnish (Takala 2016), this is probably due to vocabulary explosion problem discussed earlier. Takala 2016 proposes a word embedding method for morphologically rich language wich is based on breaking the word into sub-parts: the word stem and ending. Takala 2016 discovered that sub-word level word embeddings are simple to implement and can outperform word level embedding methods on several natural language processing tasks. How stem and ending compare to character level recurrent neural network (RNN) based methods was not tested in his work.

One problem rising from using word level word embedding methods for natural language processing tasks in highly morphological languages is that even if the word embedding vocabulary is large there will still be out of vocabulary words. This problem can be relaxed by lemmatizing the words before doing lookup to word embedding vocabulary. Lemmatization done sufficiently accurately may remove the problem for some morphologically rich languages almost entirely, but Finnish still suffers from the rich compound word scheme. Handling these out of vocabulary words in a meaningful manner turns out to be challenging because the word representation cannot be approximated from the written form. Written form of the word contains only a small part of the semantic meaning of the word, that is the reason why local window methods mine the semantic information purely from the context where the word appears in.

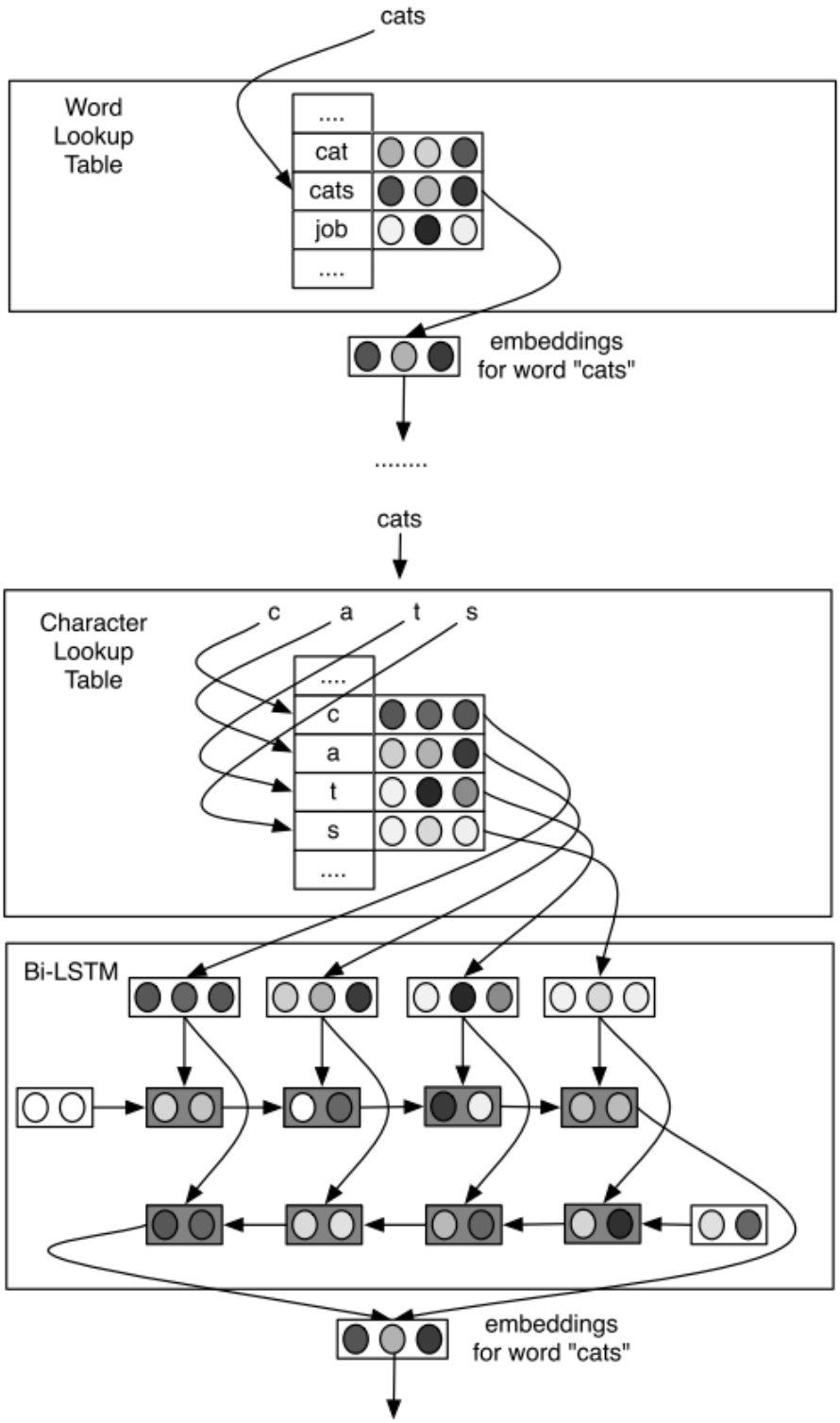
2.2.1 Character to Word

One possible solution to problems experienced with word embeddings using morphologically rich languages is called character to word (C2W) and is proposed by Ling et al. 2015. In the solution proposed by Ling et al. 2015 the word embeddings are not on word level but are generated from character sequences by feeding the words as character sequences into an RNN. Figure 2.3 illustrates the architectural difference between word level word embedding method and C2W by Ling et al. 2015. The upper part of the figure is word level method and lower part is the C2W.

Even character level models require a lookup table which has fixed number of vectors, one for each character. The benefit however comes from the drastically smaller vocabulary size that is required for characters than words. Character vocabulary cannot contain every possible character that can be ever encountered but achieving a good coverage is much easier. English alphabet contains 26 lowercase and 26 uppercase letters, totaling 52 letters to which punctuation and appropriate set of other symbols has to added in order to get good coverage over English text. Other languages may have some additional letters added to that set or even have completely different character set. Regardless of which is the case for most languages the character vocabulary size is still measured in dozens, with some exceptions with thousands of characters such as Chinese. Chinese is a bit odd one out because that language doesn't have letters but everything is represented as words, each word has it's own writing sign, and is not therefore of great concern for this method.

Additional and very much appreciated feature of C2W is that it is capable of, unlike word level embedding methods, producing word representations for previously unseen, and thus out of vocabulary, words natively without any regression add-ons

Figure 2.3 *Architectural difference between word and character level word embedding models. Image is taken from the original C2W paper by Ling et al. 2015*



(Ling et al. 2015). How well the syntactic and semantic information is embedded with C2W for previously unseen words is not discussed in the original paper of Ling et al. 2015. The main drawback of C2W is the added computational cost compared to word level methods such as word2vec by Mikolov, Corrado, et al. 2013. The computational cost can be relaxed during the training by using larger mini-batches since the word embeddings do not change within one mini-batch, as is the case for all network parameters (Ling et al. 2015). After the neural network model has been trained and is used for live predictions the computational cost for C2W can be relaxed even further by caching the most common words and using them in a similar way as other word level embedding methods (Ling et al. 2015). Rare and previously unseen words are generated during prediction phase with the RNN encoder. This scheme has the nice properties of being at the same time fast to execute because of the caching and having very limited memory requirements because rare words don't have to be cached.

Ling et al. 2015 show that their character level embedding method outperformed state of the art methods for several different languages. Biggest improvements were gained for morphologically rich languages, but C2W was shown to perform very well even on morphologically poor languages such as english (Ling et al. 2015). Suitability, in architectural wise as well as performance wise, make C2W a good candidate for neural networks which are meant to be used in multi-lingual setting.

2.3 POS-tagging

POS-tagging is the procedure of detecting parts of speeches for the words in the input text. First POS-tagging tools were rule based systems where researcher or engineer manually created set of rules which are used to determine which of the POS-tags is the correct one for the current word. The problem with rule based POS-taggers, as with any natural language processing task, is the vast complexity of human language. It is very difficult to develop and update large enough rule set that can respond to all different cases that appear in natural language. This is the reason why POS-tagging was from very early on treated as machine learning task.

One of the earliest and best known machine learning POS-taggers is a tagger introduced by Brill 1992 and simply known as Brill tagger. Brill tagger can be considered a hybrid approach between rule based tagger and machine learning tagger because the Brill tagger uses annotated corpus in a machine learning manner to create the rule set for the language. Since then the research has been mainly focusing on statistical and neural network based methods for POS-tagging.

POS-tagging as natural language processing task is fairly simple. It is a very traditional word level classification task where the label vocabulary size is limited. Multiple approaches can be take into the classification problem. Toutanova et al. 2003 tackle the POS-tagging task with cyclic dependency network. Approach taken by the UDPipe author Straka 2016 in his work is a supervised, rich feature averaged perceptron with Viterbi decoder. Ling et al. 2015 use recurrent neural networks for learning word embeddings which are then used for part of speech classification.

2.4 Lemmatisation

Lemmatization is the process of finding a base form for a word appearing in the text in inflected form. As such lemmatization can be seen as normalization technique where multitude of inflectional words are normalized to their basic expression (Korenius et al. 2004).

Another normalization technique often compared to lemmatization is stemming. Where lemmatization tries to find the base form of the word, stemming tries to find the stem of the word. Korenius et al. 2004 compare both methods for task of clustering documents written in Finnish language and find that lemmatization is better than stemming in Finnish because of it's highly inflectional nature. Stemming also suffers more from ambiguity problems since multiple different words can have same stem. The same problem also exists for lemmatization in form of homographic words, words that have same written form but different meanings, but is relaxed compared to stemming.

One publicly available tool for Finnish lemmatization is Omorfi (Pirinen 2008, Lindén et al. 2009, Pirinen 2017). Omorfi is a finite state transducer for doing morphological parsing (Pirinen 2008, Lindén et al. 2009). Because Omorfi does the full morphological analysis for the input word, it can also produce the lemma for the word by reversing the morphology from the inflected form. However caveat of this tool is that it produces multiple possibilities for the lemma, this is a direct consequence of relying simply for morphological parsing in lemmatization, multiple different lemmas can be inflected in various ways to produce the current written form. Haverinen et al. 2014 used Omorfi for building the Turku Dependency Treebank, which was later translated to Universal Dependencies scheme by Pyysalo et al. 2015. Haverinen et al. 2014 used machine learning method for disambiguating the Omorfi outputs with fall-back to manual annotations.

Kestemont et al. 2016 take neural network approach for lemmatization by classifying lemmas into a predefined vocabulary. This approach of lemmatization as word level

classification task only has the drawback that the system of Kestemont et al. 2016 cannot produce lemmas that were not encoutered during training. There were very few if any neural network based methods for lemmatization up until the writing of paper by Kestemont et al. 2016. The reason for lack of lemmatization research is that lemmatization is considered practically a solved problem for English, and since most of the natural language processing research is done for English there is not much research interest seen in studying lemmatization. Lemmatization has certainly seen interest in non-English research teams, but neural network approaches are very few. Maybe the most notable lemmatization cabable neural network method is the aforementioned UDPipe by Straka 2016 which can achieve reasonably good results for various languages.

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 choices 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 output layers 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 multitude of NLP tasks. Bag-of-words works because words are essentially the basic semantic 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 widely 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 are tightly 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.

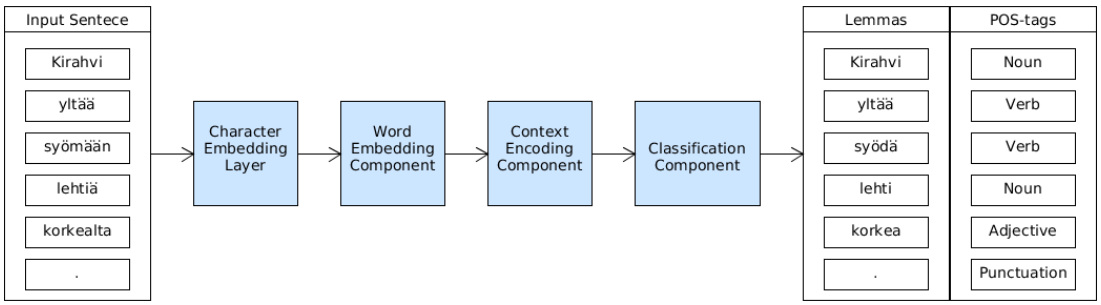
Part of speech tagging can also still be done without too much hindrance. Often the two words have the same part of speech. In the case of fastener nail and finger nail problem does not exists since both are nouns. Separating ”nails” as plural form of

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 complete multi-component 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 central processing unit (CPU) or graphics processing unit (GPU). GPU computations for pleasantly parallel problems such as neural network optimization are significantly faster. Doing the network training and especially hyper-parameter 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 (Kanerva et al. 2014) 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 eill 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 process-

ing 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 (Sutskever et al. 2014, Cho et al. 2014, Chung et al. 2016, Bahdanau et al. 2014, Liu and Lane 2016, Chung et al. 2016) 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 requirements. 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 aforementioned 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.

Selected character vocabulary 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 preceding 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 hypothesized 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 be 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 formed by concatenating word embedding vectors and context embedding vectors

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

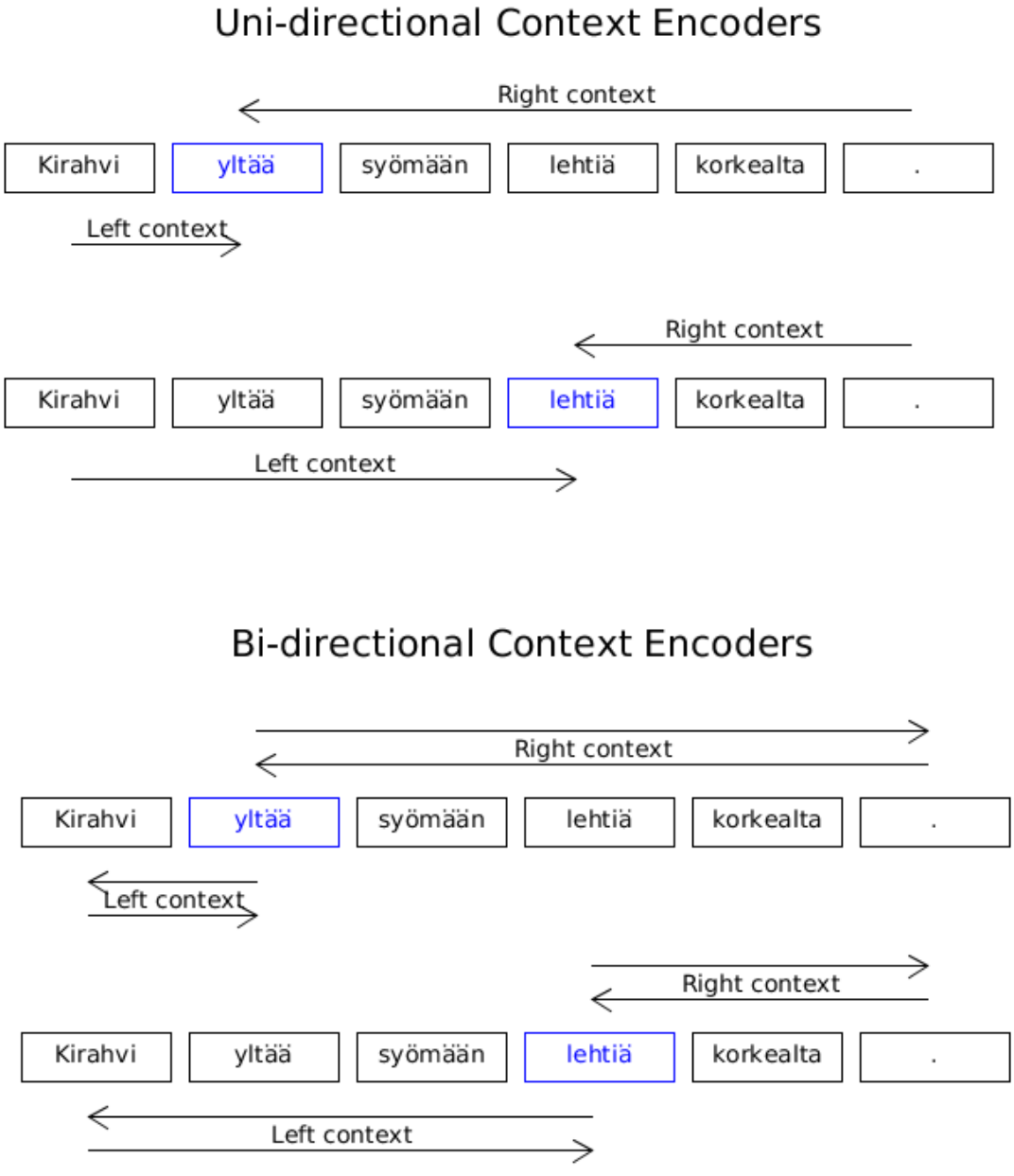


Table 3.3 *Example input of context encoder*

	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

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*

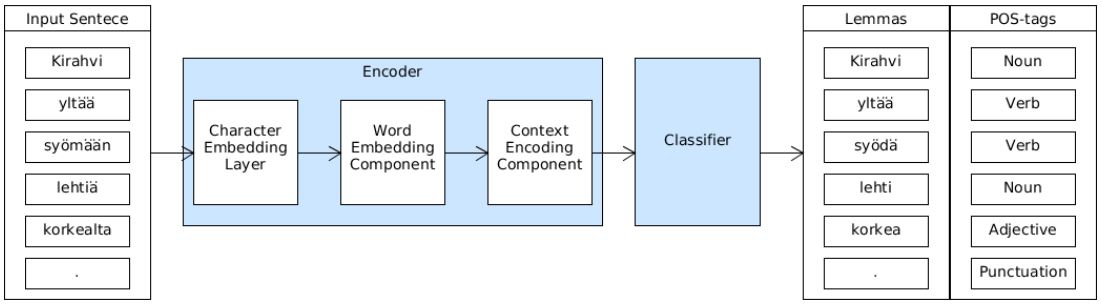
	d_0	d_1	...	d_{299}	d_{300}	d_{301}	...	d_{599}	
Kirahvi	0.96	0.12		0.87	0.69	0.88	...	0.74	t_0

Higher level constructs for information encoding, such as looking at preceding 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 conceptualized 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 and 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.

The softmax cross-entropy loss for the separate lemmatization and POS-tagging tasks is a straight forward computation from the single output projection layer values and one-hot encoded vector for correct labels. Softmax for j:th node of output layer is defined in equation 3.1, where z is the vector of output projection layer values and K is the size of output projection layer.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K}(e^{z_k})} \tag{3.1}$$

Cross entropy between softmax vector σ and one-hot encoded vector of correct lables Y is defined in equation 3.2, where K is the size of output layer.

$$H(\sigma, Y) = -\sum_{k=1}^{K}(\sigma_k \log(Y_k) \tag{3.2}$$

However loss function for the joint model is a bit more complicated since there are two optimization targets to aim for. In order to optimize both tasks at the same time, a single joint loss function has to be defined for the optimization target. Joint loss function for this work was selected to be a weighted sum of separate loss functions for lemmatization H_l and POS-tagging H_p

$$H_{joint} = \alpha H_l + H_p \tag{3.3}$$

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 algorithm. The optimizer 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.

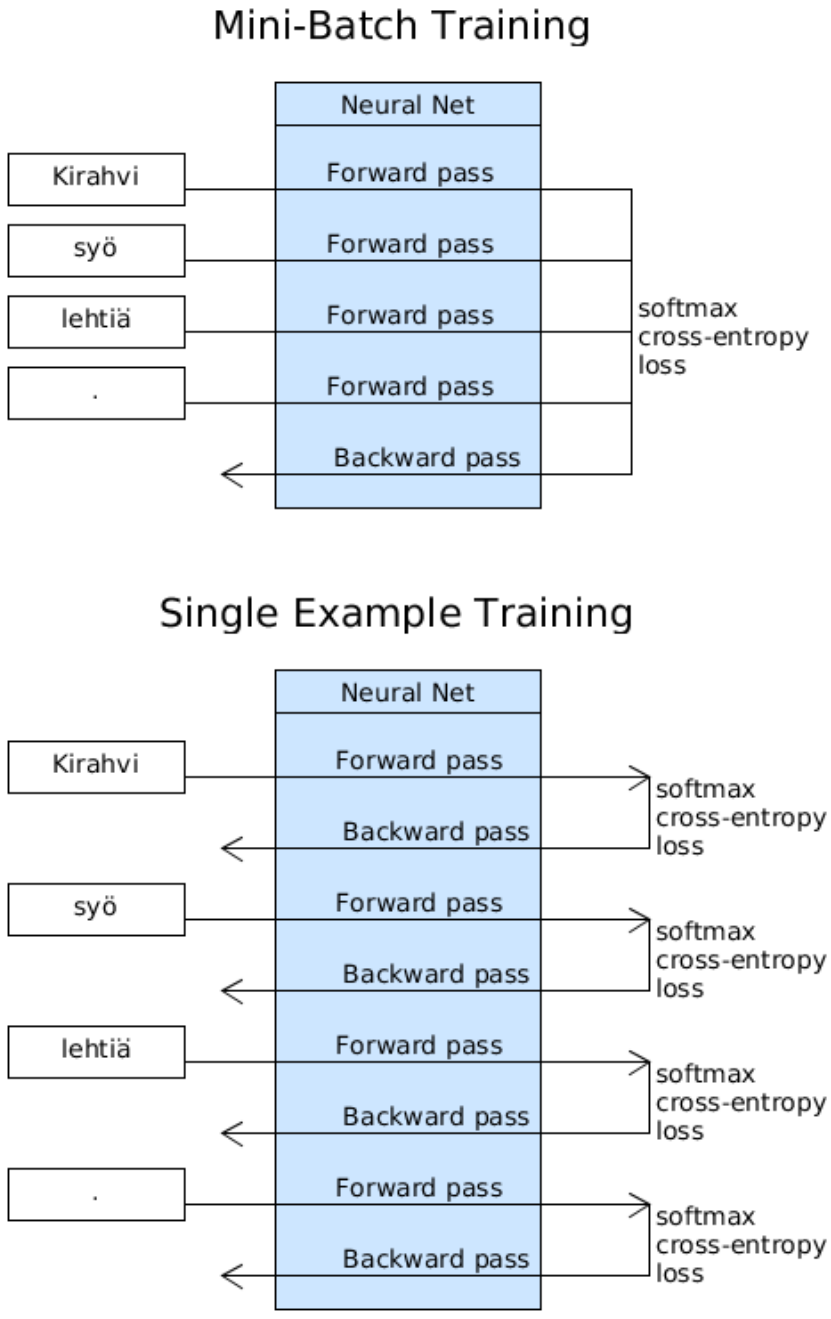
Gradient descent update was done with mini-batches of 25 sentences. This means that entire mini-batch is back-propagated through the network before adjusting the network weights and biases. Alternative is to update the network after each example but that has serious drawbacks for computational efficiency. Mini-batch training can be done in a pleasantly parallel manner, meaning that the examples 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 or 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 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

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



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 configuration, 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 optimize 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 (Claesen et al. 2017). 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.

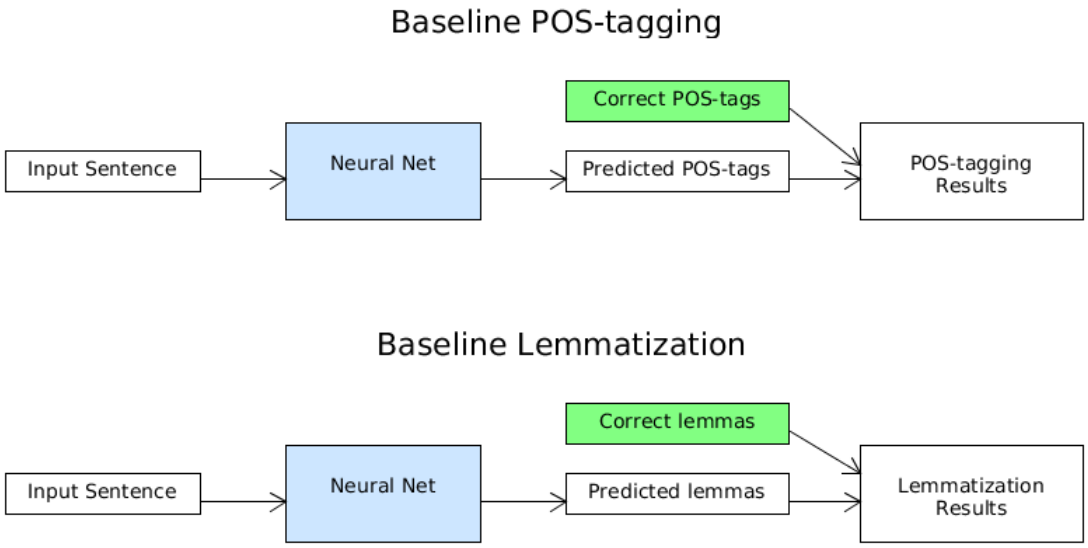
After all of the training runs for hyperparameter optimization were finished, hyperparameter configuration which provided the best model was saved for using in

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.

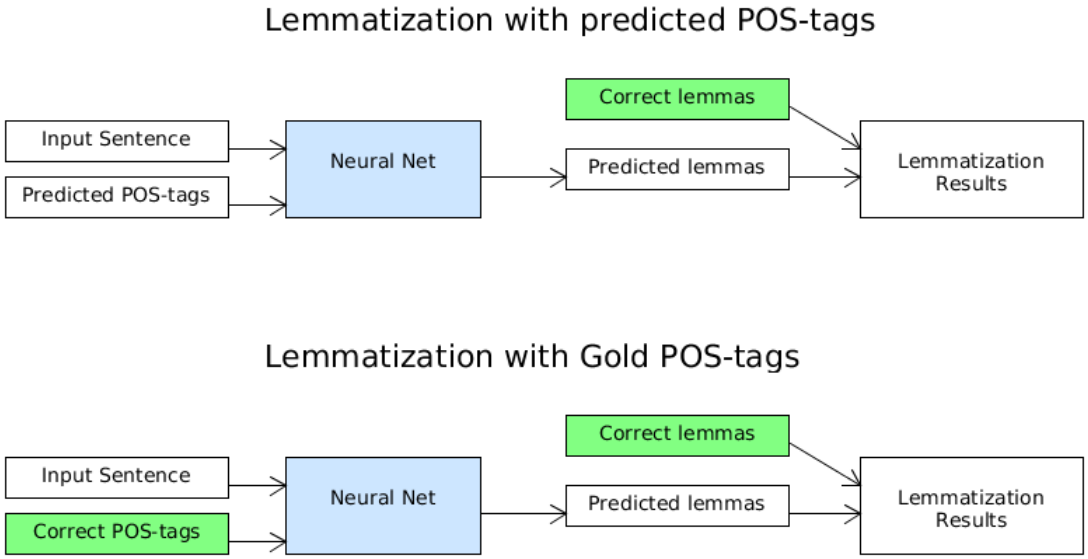
Figure 3.5 *Separate lemmatization and POS-tagging as a baseline*



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

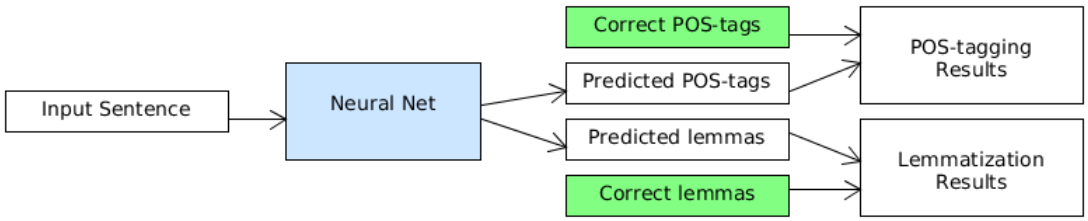
network using gold-standard parts of speech available in the dataset. Figure 3.6 shows the setup for using parts of speech as input feature for lemmatization.

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.

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



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. 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 (process of separating words in a sentence) 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 be thought 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 Dependencies 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.

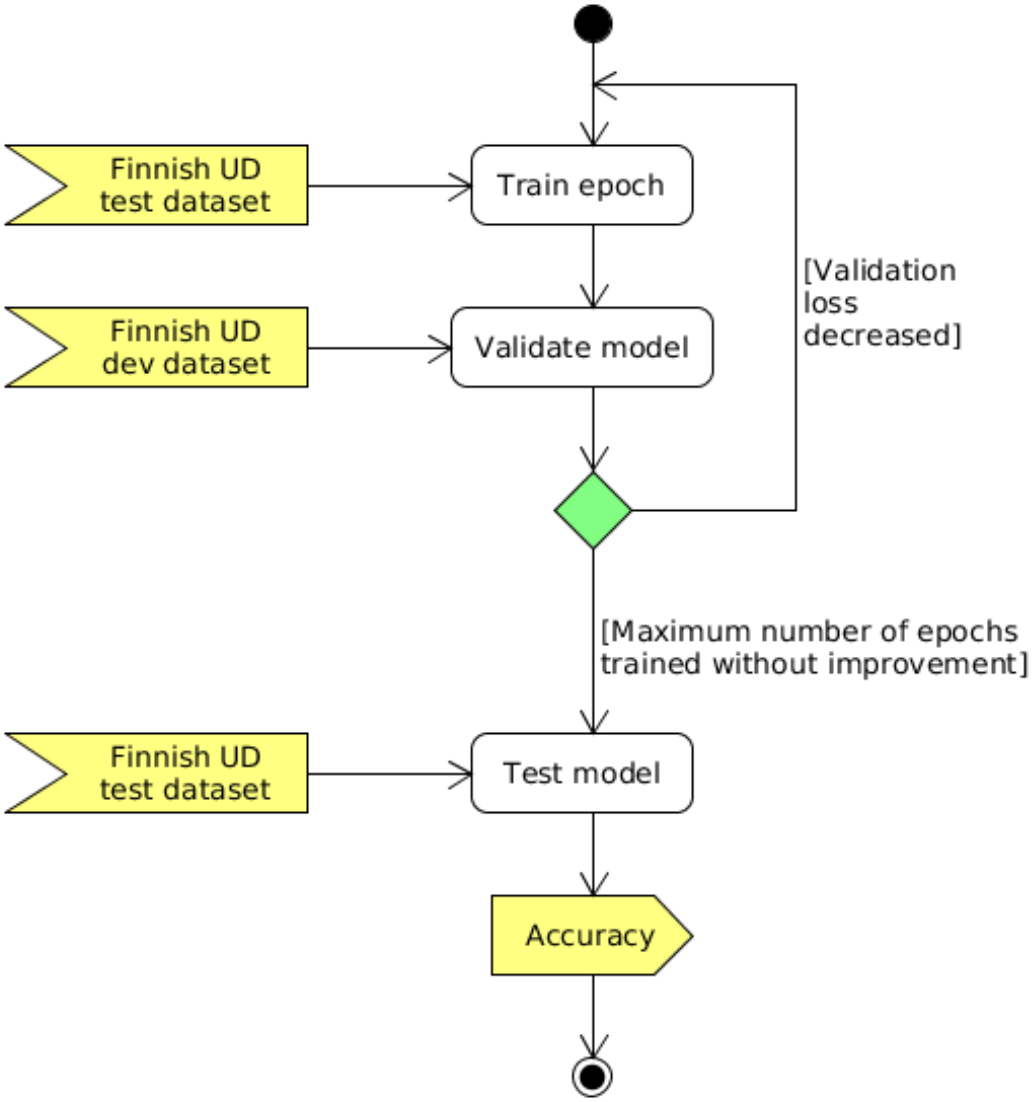
3.3 Test Methods

The dataset use for experiments in this thesis is Finnish Universal Dependencies 1.4 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.

Other corpuses exist for Finnish but format and annotation schema varies in those corpuses and are only available for Finnish. Universal 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. percentage of correct prediction over all examples in the testing dataset. F1 score

Figure 3.8 *Test setup for doing the experiments*



is another widely 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 optimizing 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

Results as accuracies for experiments described in section 3.2 are shown in table 3.5. UDPipe is used as a baseline for comparing absolute results achieved in this work. UDPipe achieves accuracies of 94,9% for POS-tagging and 86,8% for lemmatization on Finnish Universal Dependencies 2.0 dataset (Straka 2017). POS-tagging result comparison between UDPipe and this work are not perfectly fair since UDPipe results reported on UDPipe User's Manual page are obtained with Universal Dependecies 2.0 Finnish dataset whereas this work uses 1.4. Lemmatization results are not comparable at all because UDPipe lemmatizer is a generative model which forms the lemma from the stem of the word (Straka and Straková 2017). As a generative model the UDPipe lemmatizer does not suffer from vocabulary related issues, but also can produce typographical errors and other mistakes common for generative models.

Table 3.5 *Results as accuracies for the experiments.*

Experiment	POS-taging	Lemmatization
Separate POS-tagger	**94,30%**	-
Separate lemmatizer	-	94,25%
Lemmatization with classified POS-tags	-	94,40%
Lemmatization with gold POS-tags	-	**96,22%**
Joint learning model	94,14%	**95,24%**

From the results we can see that the best POS-tagging accuracy is obtained by the baseline, doing POS-tagging without lemmatization, with 94,30%. Baseline lemmatizer achieves very similar results of 94,25%. Baseline POS-tagging accuracy is

remarkably close to the UDPipe accuracy even though the learning approaches are different. The small difference of only 0,6%-points between POS-tagging accuracies in UDPipe and the baseline of this work could possibly be explained by the limitations of dataset; perhaps larger (or smaller) dataset could turn the POS-tagging accuracy into a favor of one or the other.

It's worth noting that the similarity of results between baseline POS-tagging and lemmatization don't mean that both tasks are similarly difficult; lemma vocabulary coverage has significant effect on lemmatization performance. Higher lemma vocabulary coverage will lead to having a lot more lemmas. Doubling coverage almost doubles the number of unique lemmas in the vocabulary. These new lemmas also have significantly less examples in the dataset making them much harder to classify correctly. Similarity of results between the baseline tasks is more of a coincidence than feature of the tasks.

The more traditional approach to shared information learning, using parts of speech as input features for lemmatization, yields a small improvement over the baseline lemmatization. Difference between the results is mere 0,15%-points. There are two possible explanations for this difference: using parts of speech for lemmatization does not in fact improve results and the difference is within of margin of error or errors in part of speech classification used to produce input features for lemmatization are so significant that they undermine any possible benefits.

Result accuracy for lemmatization using gold standard POS-tags, ie. POS-tags available in the Finnish Universal Dependencies corpus, provides clear insight that the latter of the proposed explanations is the correct one. Lemmatization using correct parts of speech can achieve accuracy of 96,22% which is significantly higher than the baseline and lemmatization using classified parts of speech. Improvement gained using gold standard parts of speech is 1,97%-points, which is 34,26% decrease in accuracy error. This clearly shows that there is benefit from using parts of speech as input features for lemmatization and that the shared information between the two features is useful for the tasks at hand.

Joint learning model, which is the main focus of this work, can achieve 94,14% for POS-tagging and 95,25% for lemmatization. Lemmatization performance gain from the joint model is 0,99%-points (17,22% decrease in accuracy error), which is high enough that results cannot be explained with numerical instability. Joint model lemmatization result is also the best practical result among the experiments. Lemmatization with gold standard parts of speech did obtain better accuracy than lemmatization of joint model, but in real life predictions cannot be done with gold

standard parts of speech since they are created by a team of linguists and as such are not available at that time.

Maybe the more interesting result of the joint model accuracies is the POS-tagging accuracy. POS-tagging accuracy in the joint model went down 0,16%-points when compared to the baseline separate POS-tagger. What could explain this accuracy decrease is not clear at this time. Perhaps lemmatization as more information rich task dominates the POS-tagging in the joint model not allowing the gradient descent to have enough power to adjust network weights and biases favorably for the POS-tagging task. On the other hand the difference is small enough to be at least partly explained by the numerical instability. It is also possible that doing several additional experiments with different weights for the joint loss function would have resulted into better accuracy for the POS-tagging task without much deterioration of lemmatization accuracy.

4. DISCUSSION

Results obtained in this work show that shared information in POS-tagging and lemmatization can be used to improve model performance when jointly learned. Strictly speaking the results are valid only for the specific configurations and test setup used in the experiments. Whether these results generalize for other languages, and what languages is still up for debate. Finnish is very different from most other languages spoken in Europe and therefore it could be that POS-tagging doesn't benefit lemmatization in joint model when testing on for example Swedish, German, English, French or Spanish. However part of speech and lemma are very similar concepts in basically all languages of the world with strong shared information between the features, therefore it is not far fetched to hypothesise that one can obtain similar results with other languages too.

Speculating how the results generalize to other tasks in natural language processing is another question entirely. If results obtained in this work are indeed mainly due to the strong shared information between the part of speech and the lemma of the word, then other similar task pairs that share mutual information are good candidates for joint learning models. Liu and Lane 2016 showed in their study "Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling" that there exists at least one other task pair which benefits from joint learning with neural networks. With the results of Liu and Lane 2016 and of this work it is safe to say that neural networks are indeed suitable for jointly learning natural language processing tasks with shared information.

Even further speculation about how well these results generalize can be done for tasks in other machine learning domains than natural language processing and to task pairs that don't share any direct mutual information but are still strongly correlated. It is possible that the two tasks don't actually need to share mutual information for gaining benefit from joint learning as long as correlation in the data between the two tasks exists. If this is the case then probably even the results obtained in this thesis could be explained by that correlation. Finding such a discovery of very general nature of joint task benefits would be extremely helpful in designing future machine learning systems and could be applied across several problem

domains. For example a neural network could be developed for predicting difficult to measure phenomena in industrial setting (eg. a factory or bioreactor) which uses multitude of correlated measurements as input, lowering the cost structure for measurements.

All in all there seems to be enough evidence to justify further research into joint learning models with neural networks. Especially interesting evidence are the results of Liu and Lane 2016 because the two jointly learned tasks are classified at different hierarchical levels. Slots can be considered keywords of the sentence relative to spoken language understanding and intent is a single label of semantic meaning for the sentence. This means that there exists now evidence for benefits from two word level tasks as well as word level and sentence level tasks. It should be fairly safe to assume that the task pairs can be selected from other hierarchical levels also, for example two document level tasks or document level task and a sentence level task or a word level task such as POS-tagging and a character level task such as lemma generation. The new problem which must be addressed when using two tasks of different hierarchical level is the choice of validation metrics: should for example the generative lemmatization be evaluated at word level or at character level.

4.1 Assumptions and Simplifications

Assumptions made in this thesis are mainly related to the dataset used, test setup and neural network tools used. The Finnish Universal Dependencies dataset is a machine annotated corpus with manual corrections Haverinen et al. 2014. As such the dataset was assumed to be sufficiently correctly annotated. Errors in the input dataset are fatal for machine learning methods since these methods don't have understanding about the real world, they just approximate the hidden functions of the input data. Even though the machine annotations have been manually corrected by linguists, there still exists a possibility for human errors. In fact it's not possible to be entirely sure about having 100% absolute true annotations in any corpus because the ground truth is ultimately determined by the humans who create, correct or check the corpus. Errors can only be reduced by having multiple annotators to do the same annotations and doing a cross reference, but since none of the annotators can be 100% sure, the probability for final annotations' correctness cannot be 100%.

Next assumption about the dataset used is that dataset represents Finnish language well. This assumption may not hold true as well because of the way texts in the Finnish Universal Dependencies were selected. Finnish UD dataset consists mainly of news articles, Wikipedia articles, blog posts, fiction and some other formal sources such as Europarl speeches Haverinen et al. 2014. None of these sources include texts

from conversations such as online discussion forums or even personal communication chat where form of the language may differ significantly from the more curated and formal texts. Also different dialects of Finnish language is not well represented in the corpus. All these limitations of the dataset distribution limit the model's ability to generalize to said situations and writing styles. It is also not known how big of an impact for example colloquial speech would have to benefits of POS-tagging and lemmatization joint task. At least the increased typographical variation introduced with colloquial speech is sure to raise problems related with data sparsity unless massive dataset is used. How big these issues will be is left to be tested.

When training, validating and testing the neural net model an assumption has to be made that data distribution in each of these sets are similar. By looking at the datasets one can easily deduce that at least the sentences divided into training, development and test datasets were not randomized before doing the division; there are several subsequent sentences that are obviously from the same text source in an order which looks like it could be the original one. If, and when, the sentences are not shuffled before diving into datasets, there is no guarantee that datasets share the same distribution. It could be even that for example all of the blog entries have ended up into development and training datasets, leaving no training examples about blogs to training dataset. This would surely bias the model's capability to POS-tag and lemmatizer these texts which are most likely written in more informal language that eg. news articles.

In addition to assumptions about dataset, assumptions were made about the test setup. Perhaps the most notable assumption about test setup is that limiting the lemma vocabulary and using unknown tokens for out of vocabulary lemmas would represent more realistic situation where lemma vocabulary is not artificially limited. Lemma vocabulary was limited by selecting certain coverage of uses in the training set, this entails an assumption that same coverage would be found in the development and testing datasets, or at least neural net model could generalize to learn to classify unseen lemmas as unknown tokens.

A quick lemma vocabulary coverage about the Finnish UD datasets reveals that a lemma vocabulary which covers 100% of uses in the training set only covers about 75% of uses in validation and testing datasets. This is also an indication about different data distributions in the three datasets. Lemma coverage and data distribution issues seemed not to be a big problem since the lemmatization accuracy for the test set was over 95%. Whether the accuracy would have been even higher if the distributions had matched better was not tested in this thesis. It's also worth noting that neural network model learned remarkably well to classify unseen words as unknown

tokens, if it hadn't the lemmatization accuracy for testing dataset would have been significantly lower. The final selected lemma vocabulary coverage was 90% meaning that unknown tokens cover 10% of the lemma usages in training dataset. The second most frequent term was less than 5%, so there was significant priori bias to select unknown token. It was not tested among the experiments how the lemmatization accuracy of unknown tokens would change if the unknown tokens didn't have such a big priori bias.

Another big assumption about the test setup is the complete omission of all possible problems associated with treating all words with same written forms as a single word. In reality these words have very different semantic meanings and will disturb learning of word embedding vectors by word embedding component. As was discussed in previous sections, this might not have a big impact on the performance of the model for these tasks. However problem might be really bad in downstream processing tasks which use lemmas as input features; if multiple words with different meanings but same written form need to be distinguished from each other, lemmatization done with this model is of no help.

Last significant assumption about the test setup was that running all experiments with same hyperparameters would not prevent from producing significant difference in each experiment. In optimal case each experiment would have had it's own hyperparameters optimized for the said task. Unfortunately multiple hyperparameter searches was not possible with the resources available. It is not known at this time if differences would have been greater or smaller if hyperparameters had been optimized for each taks. Hyperparameters were optimized for the joint task which showed a notable improvement for lemmatization over the baseline model, perhaps optimizing separate set of hyperparameters for the baseline model would have lead to smaller difference. However it is unlikely that the whole performance gain is explained by the better hyperparameters.

More minor simplification of the test setup is the fact that experiments were done with gold standard tokenization. Incorrect tokenization can completely ruin word level classification tasks for the incorrectly tokenized words. However tokenization is a lot simpler task which can be inferred from the UDPipe tokenization results (Straka and Straková 2017). UDPipe tokenizer achieves F1 score of 99,69 for Finnish Universal Dependencies 2.0. This means that there is approximately one incorrect tokenization in 77 sentences, when average sentence length is 13. If worst case scenario is assumed ie. one incorrect tokenization ruins lemmatization and POS-tagging for the entire sentence, causes tokenization errors $1/77 = 1,3\%$ error for both tasks. However it's not very reasonable to assume such an effect for all the words in the

sentence. Tokenization error can certainly affect the POS-tagging and lemmatization results, but the effect should be relatively small. This same simplification also applies to sentence segmentation. Whether the sentence segmentation is necessary step of the pipeline depends on the task. If the purpose is just to provide a tool for linguists to help with lemmatization and POS-tagging of separate sentences, the sentence segmentation is not required.

Hyperparameter optimization was also exposed to some simplifications in addition to not doing it for all of the experiments separately. Firstly the hyperparameter optimization had fixed maximum time budgets for the model trainings in order to counter the possibly very long training times with certain hyperparameter configurations. Limiting the maximum time budget also puts severe constraints on the models which are affected by the time limit, these models are not allowed to fully converge but are stopped prematurely. This means that the hyperparameters are not globally optimal, but only optimal within the given constraints. Hyperparameter optimization was also constrained by the maximum number of training runs used for the search, this entails a situation where the hyperparameters are not fully converged to their optimal values but rather are the best ones that were obtained with the limited number of runs. Hyperparameter search ranges were also limited for each hyperparameter so the hyperparameters found with the search can only be optimal within those ranges. If all simplifications of hyperparameter search were to be removed, countless number of runs would have been needed and a single run could have taken time that approaches infinite as the layer sizes and counts rise, certainly infeasible situation.

Different kind of limitation to hyperparameter search was the exclusion of weight α in joint model loss function defined in equation 3.3. α cannot be optimized along with other hyperparameters because adjusting the α will cause the optimization target to change. Changing optimization target between optimization runs renders the entire optimization process meaningless. Determining the value for α is really a decision about what is appreciated, if POS-tagging accuracy is valued higher than lemmatization accuracy then more weight has to be given to POS-tagging loss in the joint loss. One is not worse than the other, it's simply a question of preference.

Minor assumptions were made for neural network methods and tools used for implementing the model. The neural network weights and biases were initialized with small random values before starting the training. Using random values will produce different initializations for each training run which in turn will eventually lead to different results obtained from the two models. How big this difference depends largely on the size of the dataset, the larger the dataset the smaller is the impact

of initialization. Finnish Universal Dependencies dataset contains 204399 tokens of which about 10% is reserved for development set and about 10% for test set, leaving over 160000 tokens for the training dataset. 160000 tokens is sufficient to rule out significant impact of the random initializations. However smaller differences such as the 0,15%-point increase on lemmatization accuracy when using classified POS-tags as input features and 0,16%-point decrease in POS-tagging accuracy for the joint model could be partly explained by the random initializations.

It was also assumed that typical numerical instability always present at limited precision computer systems would have near to zero impact on the results and variance between the results. Other sources of numerical errors are for system memory and GPU memory; the neural network was trained on regular consumer hardware which doesn't error correction for the memories used. It was also assumed that these errors would not play notable part in the results because neural network training can be considered as nothing more than an error correction process.

5. CONCLUSIONS AND PROPOSALS FOR FUTURE RESEARCH

Objective of this thesis was to build a neural network architecture for syntactic parsing, namely lemmatization and POS-tagging. Research hypothesis was if the architecture built for separate tasks could be modified in such a way that both tasks, lemmatization and POS-tagging, could be learned and predicted at the same time and if the joint learning model would be better at classifying lemmas and POS-tags. Objective was reached and research hypothesis was proved partly correct. Required architectural modifications were minor to produce the joint model, and it was observed that joint model is better at classifying lemmas but not parts of speech.

Chapter 2 of this thesis was the theory section in which current academic status, and a bit of history, was presented on natural language processing with and without neural networks. Theory section discussed the most important problems in representing natural language with computer systems, what implications text as input data introduces to machine learning methods and what kind of approaches previous research has taken into solving them. Also problem associated with highly morphological languages such as Finnish were addressed in the theory section with different methods of input representations and their pros and cons in this context. Conclusion was that highly morphological languages are not suitable for word level representations because of vocabulary explosion problem. Character level models were introduced as a solution to inflections and morphology. Theory section also included syntactic parsing tasks most relevant for this thesis. Lemmatization and POS-taggin were the main focus of syntactic parsing tasks in theory section, as the were main focus of the research problem in experiments chapter.

Chapter 3 is the practical and research part of this thesis. First the hypothesis and research problem were introduced in experiments chapter. Introduction of experiments chapter also includes reasoning for selecting lemmatization and POS-tagging as the observed tasks and, selecting neural networks as implementational approach to forming the solution and selecting the joint model for the basis of the hypothe-

sis. Lemmatization and POS-tagging were selected as the tasks because they share strong mutual information and are strongly correlated which make suitable for joint learning model tasks. Neural networks were selected as implementational approach because they have shown tremendous progress on several different problem domains and also in natural language processing. Other reason for selecting neural networks was that they are generally fairly simple to modify for joint learning tasks and since they are trained by optimizing the loss function multiple tasks can be learned simultaneously by selecting appropriate loss function.

First part after the introduction (3.1) of the chapter 3 describes the neural network architecture used for the experiments with it's various components developed for different sub-tasks. Architecture is composed of four components each on their own hierarchical level: first is the character level representation, second word level representation, and then the word representations are joined for contextual representation which is used as an input for the classification component which outputs the predictions. Neural network architecture section also presents the reasoning for the architectural decision for doing lemmatization as word level classification task only, this was mainly to keep the architecture simpler and keep all predictions on the same hierarchical level, word level, to avoid problems associated with selecting a metric that can handle multiple hierarchical levels. The last part of the 3rd architecture section explained the neural network training and hyperparameter optimization procedures.

Experiments used to test the research hypothesis were presented in the experiments section 3.2 of the chapter 3. Experiment section also has reasoning for selection of different experiments. Baseline was established by having the two selected tasks, POS-tagging and lemmatization, as separate models. Then an experiment with joint learning model was introduced which can be compared to the separate models to gain insights about how much improvement the joint model can provide over the baseline, if at all. Additional experiments were added to determine if more traditional approach of using POS-tags as input features for lemmatization would provide similar benefits: one with classified parts of speech and one with gold standard ones.

Section 3.3 outlined the test setup used to do the experiments. Firstly the section states that Finnish Universal Dependencies were used as a dataset for training, developing and testing the neural network models. Then several possible metrics for evaluating the results was discussed with the conclusion of using a simple accuracy because it should be enough to test the hypothesis and is easier to understand for a reader without education in statistical analysis.

Last section of the chapter 3 presents results obtained by doing the experiments with the described test setup. Main discovery was that joint model does indeed improve lemmatization performance over the baseline with 0,99%-point increase and 17,22% decrease in accuracy error. Interestingly POS-tagging performance was not improved and as a matter of fact was very slightly decreased from the baseline. Experiments done with using POS-tags as additional input features reveled that no significant improvement can be gained by using classified POS-tags unless the POS-tagging accuracy is very high. This perfect POS-tags the lemmatization saw very significant improvement of 1,97%-points in classification accuracy, which is also a 34,26% decrease in accuracy error.

Chapter 4 has discussion for the generalizability of the results. It was stated in chapter 4 that results obtained in the experiments should be generalizable to other other languages since almost all languages have same concepts for lemma and part of speech. Additional note was made that results should generalize to other shared information task pairs within natural language understanding, this was further reinforced by the results of Liu and Lane 2016. It was mentioned that assuming that the results would generalize to other problem domains or even to general correlating tasks is not safe to do based on these results alone.

The latter part of discussions chapter covered assumptions made about data, test setup and results. Dataset was assumed to be annotated correctly enough, even though there still is the possibility of human error. Another important assumption made about the dataset that it would represent Finnish language well, which might not be the case entirely as was noted. Major assumptions and simplifications about the test setup were mentioned to be the treatment of lemmatization as word level classification task with the unknown tokens and running all experiments with same hyperparameters.

Assumptions, limitations and simplifications discussed in the chapter 4 open up several interesting research opportunities for the future. Firstly and most obviously it must be tested what kind of performances the model can achieve for other languages, and especially the other Finnish UD dataset Finnish FTB. The beauty of Universal Dependencies lies with the fact that all languages have uniform annotations and therefore plugging in another language from Universal Dependencies should be as easy as feeding in another data files. Taking language capabilities even further a research about multi-lingual models should be conducted. A neural network model which takes multiple datasets as input, all of which are in different language would prove very useful and at least ease the multi-lingual processing by limiting number of models to one. Whether multi-lingual model can provide as good results or even

better as separate models for each language remains to be seen. A good foundation for multi-lingual models is provided by pairs or sets of languages which are linguistically close to each other, one such pair could be Swedish and Norwegian.

Another interesting prospect for a future research is the usage of additional information available in the Universal Dependencies corpuses such as morphology and language specific POS-tags. There is a solid reasoning for using morphology for lemmatization or even adding morphological parsing to the joint learning model as third task. Morphology describes the form and inflection of a word and since lemmatization can be seen as reverse process of inflecting the word, should morphological information be of great benefit for lemmatization. Modifying the architecture to include third task is not a large work, but bigger question is whether the three tasks can be learned successfully together without one task starting to dominate the others.

Architecture developed for this thesis also has room for improvement by tuning the α weight of joint loss function 3.3. If POS-tagging is the only task to be performed, then adding a joint model which learn lemmatization also is probably not worth the added computational cost. If both tasks need to be done, but priority is for the POS-tagging, then a joint model could prove suitable if joint loss weight is tuned to favor POS-tagging. POS-tagging is recommended to be done always even if only lemmatization is required since it improves the lemmatization accuracy without adding too much computational cost. Best possible results could be obtained by training two different joint learning models, one with loss weight and other hyperparameters tuned for POS-tagging performance and one with with loss weight and hyperparameters tuned for lemmatization performance. Drawback of this approach is the doubling of memory usage, which can be a scarce resource with GPUs, and doubling of computational cost. All these speculations need to be confirmed or belied with further research.

BIBLIOGRAPHY

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.

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.

Claesen, M., J. Simm, and V. Jumutc (2017). *Optunity*. URL: http://optunity.readthedocs.io/en/latest/ (visited on 10/07/2017).

Elman, J. L. (1990). "Finding structure in time* 1". In: *Cognitive science* 14.2, pp. 179–211. ISSN: 03640213. DOI: 10.1207/s15516709cog1402_1. URL: http://doi.wiley.com/10.1207/s15516709cog1402%7B%5C_%7D1.

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.

Hinton, G. E., D. E. Rumelhart, and R. J. Williams (1985). "Learning internal representations by back-propagating errors". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* 1.

Hinton, G. E., J. L. McClelland, and D. E. Rumelhart (1990). "Distributed representations". In: *The philosophy of artificial intelligence*, pp. 248–280. ISSN: 1534-7362. DOI: 10.1146/annurev-psych-120710-100344. arXiv: 15334406.

Kanerva, J. et al. (2014). "Syntactic N-gram Collection from a Large-Scale Corpus of Internet Finnish". In: *Frontiers in Artificial Intelligence and Applications* 268, pp. 184–191. ISSN: 09226389. DOI: 10.3233/978-1-61499-442-8-184.

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.

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.

Lindén, K., M. Silfverberg, and T. Pirinen (2009). "HFST tools for morphology - An efficient open-source package for construction of morphological analyzers". In: *Communications in Computer and Information Science* 41 CCIS, pp. 28–47. ISSN: 18650929. DOI: 10.1007/978-3-642-04131-0_3.

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.

Pennington, J., R. Socher, and C. D. Manning (2014). "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empiri-*

cal Methods in Natural Language Processing, pp. 1532–1543. ISSN: 10495258. DOI: 10.3115/v1/D14-1162. arXiv: 1504.06654.

Pirinen, T. (2017). *Omorfi GitHub*. URL: https://github.com/flammie/omorfi (visited on 10/08/2017).

– (2008). "Suomen kielen {ä}{ä}rellistilainen automaattinen morfologinen analyysi avoimen l{ä}hdekoodin menetelmin". PhD thesis. Master's thesis, Helsingin yliopisto.

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

Straka, M. (2016). "UDPipe: Trainable Pipeline for Processing CoNLL-U Files Performing Tokenization, Morphological Analysis, POS Tagging and Parsing". In: *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pp. 4290–4297.

– (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.

Toutanova, K. et al. (2003). "Feature-rich part-of-speech tagging with a cyclic dependency network". In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - NAACL '03* 1, pp. 173–180. DOI: 10.3115/1073445.1073478. URL: http://portal.acm.org/citation.cfm?doid=1073445.1073478.