Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Stine S. Stavland

Convolutional Neural Networks for Part-of-Speech Tagging

Masteroppgave i Datateknologi
Veileder: Björn Gambäck
Januar 2020

NTNU
Kunnskap for en bedre verden

Stine S. Stavland

Convolutional Neural Networks for Part-of-Speech Tagging

Masteroppgave i Datateknologi
Veileder: Björn Gambäck
Januar 2020

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

NTNU

Kunnskap for en bedre verden

Abstract

Inspired by recent work in the fields of Deep Learning and the use of Convolutional Neural Networks for sequence processing, this Master's thesis concerns the use of Convolutional Neural Networks for the otherwise well-explored Natural Language Processing task of Part-of-Speech tagging. The main contribution was the development of two taggers using Convolutional Neural Networks with different activation functions and the results of training and testing the taggers on one of the Norwegian treebanks and one of the English treebanks of the Universal Dependencies project.

The results of the experiments show that a Part-of-Speech tagger based on Convolutional Neural Networks can achieve comparable performance to a Long Short-Term Memory based tagger, but state-of-the-art results were not achieved.

Sammendrag

Konvolusjonelle nevrale nettverk har vanligvis ikke vært brukt til sekvensprosesseringsoppgaver, men nylige forskningsarbeid har gitt inspirasjon til å teste ut bruk av konvolusjonelle nevrale nettverk for ordklassemerking. To taggere med ulike aktiveringsfunksjoner ble utviklet og testet, en av dem mer grundig enn den andre. Taggerne ble testet på de norske og engelske delene av Universal Dependencies prosjektet.

Eksperimentene viste at konvolusjonelle nevrale nettverk kan oppnå resultater som kan sammenlignes med tilbakevendende nevrale nettverk på ordklassemerkingsoppgaven, men de konvolusjonelle nettverkene når ikke opp til de beste resultatene per i dag.

Preface

This Master's thesis was conducted at the Norwegian University of Science and Technology (NTNU) in Trondheim by Stine S. Stavland as the final part of the degree of Master of Science with specialization in Computer Science. The work was supervised by Björn Gambäck, whose knowledge and patience were vital to the completion of the thesis.

Stine S. Stavland
Trondheim, 9th January 2020

Contents

List of Figures

List of Tables

1 Introduction

While people might be fine with pushing buttons and swiping screens to make our computers do what we want them to do today, being able to talk to them and be understood has been dreamed of since the very beginning of computer science. Humans have a tendency to anthropomorphize objects and animals, and computers are not exempt from the human need for social interaction. For computers to be able to fulfil such tasks, Natural Language Processing (NLP) is required.

NLP can be done on several levels, from parsing to sentiment analysis, and the higher levels are often dependent on the lower. Part-of-Speech tagging (POS tagging) is one of the lower level methods, but it is important all the same. Many taggers exist, using a variety of methods. Recent research has among other things focused on developing taggers for languages with little existing language processing resources and on applying new machine learning algorithms to the task.

Machine learning is a field of computer science concerned with how computers can learn from experience. In practice this generally means algorithms that take data as input and use the given data to produce a function capable of performing the desired task. There are many different algorithms that fall under the machine learning term, ranging from simple regression to the use of deep artificial neural networks, often called Deep Learning.

Deep learning is a field in machine learning that has recently become more relevant. Though many of the algorithms have been known for years, the computational power and data amount necessary to make good use of them are relatively recent developments. As such, it has also become interesting to apply deep learning algorithms to NLP tasks, POS tagging among them.

One of the more well-known deep learning algorithms is the Convolutional Neural Network (CNN). It is commonly used for image processing as it preserves topological information and can be layered to extract increasingly higher-leveled features of the input. It can also be used for one-dimensional input, or sequences, though other algorithms aimed at sequence-processing have normally been preferred.

This report will focus on the use of CNNs for POS tagging.

1.1 Background and Motivation

Part-of-speech tagging lends itself well to supervised machine learning algorithms as they are concerned with predicting labels for input. In this case the labels would be the word classes associated with the words in the input.

More recent developments in machine learning are deep learning algorithms that are more effective and efficient. Some research has been done on the application of specialized deep neural networks such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks to the task of POS tagging. Good results have been achieved with these methods. As POS tagging is a sequence-to-sequence problem, the most common deep learning models used are those based on RNNs, which have long been preferred for sequence processing. Plank et al. (2016) use a Bi-directional Long Short-Term Memory (bi-LSTM) network on many languages of the Universal Dependency treebanks (Nivre et al., 2015), demonstrating the effectiveness of the bi-LSTM in POS tagging. As the UD treebanks are freely available in many languages and use a standard format, they make for a good data source for POS tagging and other NLP tasks.

Bai et al. (2018) showed that models based on CNN can also perform well on sequence problems. So the interest arose to explore them further and to test the effectiveness of CNN for POS tagging, using the UD treebanks.

1.2 Goals and Research Questions

The initial aim was to use Deep Learning algorithms for part-of-speech tagging. The focus then came to be on using CNN models for the task. Compared to other Deep Learning algorithms, like LSTM, little work had been done with CNN for POS tagging specifically. It was therefore desired to test the performance of a CNN model and consider whether CNN could be considered a viable alternative to the more commonly used algorithms for POS tagging.

Goal *Investigate the usefulness of CNNs for part-of-speech tagging:*
Test and consider whether CNN can be thought of as a useful alternative to more traditional sequence-processing algorithms for POS tagging.

These research questions were then relevant to consider:

Research question 1 *Can a CNN achieve good results for Part-of-speech tagging?*
What level of performance can be expected from a CNN on the POS tagging task, and how does it compare to the results of other algorithms?

Research question 2 *How does a CNN-based Part-of-Speech tagger perform on a Norwegian dataset?*
The Norwegian language is one of many languages for which there are few existing NLP-resources. Can a CNN-based Part-of-Speech tagger be useful for tagging Norwegian?

Research question 3 *How does the CNN perform on different languages?*
Is the performance consistent across different languages? How does it compare to different algorithms on different languages?

1.3 Contributions

The contributions of this Master's thesis are the development of a CNN-based Part-of-Speech tagger using Gated Linear Units and a CNN-based Part-of-Speech tagger using Rectified Linear Units, and the results of training and testing the two taggers on the Norwegian and English parts of the Universal Dependencies project. The two taggers will be made available on GitHub.

1.4 Report Structure

The thesis has five chapters in addition to this introduction. The chapters and their contents are as follows:

Background and Resources *introduces methods, algorithms and terminology related to Part-of-Speech tagging and Deep Learning.*

Related Work *contains a review of some work related to CNN and Deep Learning for Part-of-Speech tagging.*

Architecture *describes the architecture and data used in the experiments.*

Experiments and Results *describes the experiments performed and their results.*

Discussion *contains the discussion of the results reported in the Experiments chapter, a final summary of the work, and suggestions for future work.*

2 Background and Resources

This chapter gives an introduction to and overview of some methods, algorithms and terminology related to Deep Learning and Natural Language Processing.

2.1 Natural Language Processing

Natural Language Processing (NLP) refers to the processing of language and communication in the forms most commonly used by humans. Even for humans, using and understanding languages require years of training and experience, and even then we often make mistakes. Computers are not yet as versatile as the human brain, and even if they have the advantage of faster processing, they still require good algorithms and large amounts of suitable data.

NLP can be used for many purposes. As previously mentioned, it can be used to facilitate communication between humans and computers, both spoken and written. It can also be used to facilitate communication between humans, for example by translating between different languages. Another important use is to extract information, especially from written text. The internet contains massive amounts of readily available written text that can be used, and many sites have great amounts of user generated content, allowing processing methods like opinion mining.

There are some inherent challenges in NLP that even humans sometimes struggle with. One of them is ambiguity, that words and sentences can mean different things. Another is the difficulty of conveying tone with text, making it hard to convey whether a statement is meant seriously or sarcastically. Often it comes down to a lack of context. Knowledge of the speaker and the situation that surrounds the statement is often required to interpret a statement properly. Another challenge lies in the sheer varitey of languages. It is estimated that between 5000 and 7000 languages are in use today. Naturally this means there is a great variety in phonology, grammar, writing systems as well as culture. There is also variation in the languages themselves; different dialects, sociolects, pronunciations and other forms of non-standardized language and language features. Finding good solutions to these problems is one of the goals of current research in NLP fields.

NLP encompasses many specific tasks. Some are lower level tasks that operate on the word and sentence-level, such as Parsing, Lemmatization, and Part-of-Speech tagging. Others, such as Speech Recognition, Machine Translation, and Text Generation, are higher level tasks that often make use the lower level tasks as part of of the process.

2.1.1 Language Modeling

Language modeling is the task of modeling the probability of word sequences. It is an important NLP task which can be used as the basis for many higher level processing tasks, for example Machine Translation and Text Generation. Part-of-Speech tagging (POS tagging) can also benefit from Language modeling.

Language modeling is a sequence processing task. Deep Learning algorithms can learn a language model by predicting the next word in a word sequence based on the previous words. Markov chains and Hidden Markov Models (HMMs) do the same using n-grams. N-grams are sequences of n linguistic units, for example words or letters, and are often useful in creating a statistical language model. The language model learned by a neural network is distributed and continuous due to being represented by the weights of the network (see section 2.2).

2.1.2 Part of Speech Tagging

All words in a language can be categorized into word classes, for example nouns, verbs and adjectives. The word classes are also known as parts-of-speech, and the act of assigning word classes to words is often called POS tagging. The class of a word gives a lot of information about the word itself and its neighbours and is therefore often very useful in NLP. A set of word classes that are used for tagging is called a tagset. There are many different tagsets for different languages and different collections of annotated texts.

Some challenges in POS tagging are disambiguation of words, tagging unknown words and tagging in highly inflectional or agglutinative languages. Most POS tagging methods require annotated data for statistical inference. Another challenge then is to make taggers for languages for which such resources do not exist, and which might have little in ways of digital corpora as well. Code-switching texts that alternate between two languages and texts that use a form of language that is not standard, such as informal twitter posts or text messages, are also challenging to tag.

In machine learning terms, POS tagging is a sequence-to-sequence task, meaning that both the input and the output is a sequence. In this case the input sequence and the output sequence must have the same length, since every word needs to have a corresponding tag.

2.1.3 Embeddings

Embeddings are real-valued vectors that have low dimensionality and can, among other methods, be learned by relatively shallow neural networks. The embeddings can be used as input format for higher-level processing. In the case of word embeddings, the networks use the context of a word, its neighbours, to create a vector that is close to vectors of other words that appear in similar contexts. The vectors created in this way then contain semantic and syntactic information which can be useful for many NLP tasks. Similarly, character embeddings are real-valued vectors representing individual

characters.

2.2 Deep Learning

Deep learning algorithms are variants of Artificial Neural Networks (ANNs) with multiple hidden layers. ANNs are machine learning algorithms, which are characterized by their ability to improve their performance on a task with experience.

Machine learning algorithms are often divided into two categories: supervised learning algorithms and unsupervised learning algorithms. Supervised learning algorithms take labeled input and learn to associate the input with its label. Classification and regression are common supervised tasks. Unsupervised learning algorithms do not take explicitly labeled input. Instead, they learn features of the data they are given. Most machine learning algorithms can be described as trying to minimize a loss function.

ANNs are inspired by biological neural networks. The neurons are represented by weights and activation functions applied to the input. The artificial neurons, also called processing units, are usually organized in layers through which the input is transformed and then passed on. The layers can have different amounts of artificial neurons, and there can be different kinds of connections between the layers. There are also many different activation functions that can be used by the artificial neurons in a given layer. ANNs can be considered to be universal function approximators if they use nonlinear processing units and have an appropriate number of them. However, in practice different ANNs with different layer structures are required for good performance. ANNs with complex layer structures tend to fall under the Deep Learning category.

There are many different network models and methods of training used in Deep Learning. Apart from Feed-Forward Neural Networks (FFNs) with linear layers, the major types of deep learning networks are Recurrent Neural Networks (RNNs) and Convolutional Neural Networks (CNNs). To train the networks, optimization algorithms are used to update the weights. There are many variations of training methods, though most build on the same algorithms.

2.2.1 Optimization algorithms

ANNs learn by iteratively having their weights modified to minimize a loss function. The algorithm used to modify the weights is called the optimization algorithm. The most commonly used algorithms in Deep Learning are based on gradient descent, which is an iterative optimization algorithm. For every step it calculates the gradient of the loss function and moves in the opposite direction, which is the direction of steepest descent for the function. The backpropagation algorithm is normally used to calculate the gradient of the individual weights. Gradient descent is often used in the form of Stochastic Gradient Descent (SGD), which updates weights with every sample.

The learning rate determines the size of the update to the weights and thus the step size for gradient based algorithms. If it is kept constant, or too big, it might make descent into minima of the loss function difficult if the area around is too variable. One way to

amend this is to use momentum, which adds some of the gradient at the previous step to the update to keep it moving in the most useful direction. Another way to solve the potential problem is to change the learning rate during training to avoid unproductive behaviour. Adagrad (Duchi et al., 2011) and Adam (Kingma and Ba, 2014) are two examples of optimization algorithms using adaptive learning rates.

There are further methods that are used to improve on the training of the ANNs, referred to as requlariza tion techniques. One of them is dropout, where the artificial neurons have a given possibility of being ignored during training. Some other methods are weight regularization, batch normalization, and gradient clipping.

Learning rate, momentum, and other values relating to the architecture and training of an ANN are often referred to as hyperparameters. There can be many hyperparameters to choose from, and many values one must choose between before finding a set one wishes to use for training and testing. There are two main ways of finding a good set of values for the hyperparameters. One is by using Grid Search to run through all combinations of values in the chosen ranges. This would reveal the optimal values within the given ranges, but it is rather computationally expensive as all combinations must be checked. The other alternative is to use Random Search to run through a given number of randomly chosen sets of values for the hyperparameters in the given ranges. This approach does not guarantee any optimal solutions, but it generally finds a set of values that work well enough without needing to check all options.

2.2.2 Convolutional Neural Networks

CNNs (LeCun and Bengio, 1995) are a kind of ANNs that are often used for image processing because of their ability to capture different levels of spatial features. This ability can also be extended to temporal features in time series. CNNs do this by using the operation that gives them their name, the convolution. Convolution in this case refers to a function, often called a kernel or a filter, being applied to different subsections of the input and producing output that has the same spatial positioning as the subsections, allowing the network to both detect features across neighboring input units and keep important spatial information. By arranging multiple such layers in a hierarchical structure, a CNN can detect increasingly higher-level features.

An important consideration for CNNs is how to pick out the subsections. The size and dimensions of the chosen filter determines the number and orientation of the input units chosen, but not the method by which the input units are chosen. Two important parameters here are stride and dilation (Yu and Koltun, 2015). Stride refers to the distance in input units between the different subsections picked out. Dilation refers to distance between input units used by the filter. Both parameters can be adjusted to achieve a balance between excessive redundancy and possible loss of features and information.

It is common to use pooling layers, using for example the max function or the average function, between the convolutional layers. These layers downsample their input and provide spatial invariance for the features detected by their input layers.

The input and output to a convolutional layer can have several planes, called channels.

These can have individual filters, or all input channels can be used to produce all output channels. Each filter produces its own output channel.

Residual networks are a variant on CNNs where the input of a convolutional layer or block of layers is added to its output, which helps alleviate the problem of vanishing gradients (see section 2.2.3). It means the network needs to learn the residual, the difference between input and desired output, rather than the desired output itself, which in practice seems to work well (He et al., 2016).

For classification after the input has passed through the CNN, the output is normally flattened and used in a regular FFN.

2.2.3 Recurrent Neural Networks

RNNs are a kind of ANNs that use internal loops to imitate memory, which makes them good for processing sequences. These loops provide an input from the same cell at a previous stage. Conventional RNNs suffer from the vanishing gradients problem, since the gradients can become very small when propagated through time, or many layers. A similar problem is exploding gradients, when the gradients grow very big for the same reasons. Different variants have been developed to deal with these problems, but they are more complex and have more parameters that must be learned.

Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997; Chung et al., 2014) networks, using LSTM units, are a solution to the vanishing gradients problem in RNNs. They have a cell state separate from the input and output of the cell and uses sigmoid gates to control what is learned and forgotten at each step. The cell has three inputs: the cell state, the hidden state and the outside input. The hidden state and the cell state are separate.

Another variant is the Gated Recurrent Unit (GRU) (Cho et al., 2014; Chung et al., 2014), which is similar to LSTM units, but has among other things merged the input and forget gate into an update gate and the hidden state and cell state into one state to produce a somewhat simpler design that still works well.

2.3 Resources

Of the available deep learning frameworks and NLP resources, the following were chosen as they were easily available and easy to work with.

2.3.1 Universal Dependencies

Universal Dependencies (Nivre et al., 2017) is a framework for cross-linguistical grammatical annotation created by open collaboration of various people and teams. Their stated goal is to facilitate research and development that requires or benefits from resources in multiple languages. The Universal Dependencies project does this by providing a consistent annotation of similar language constructs, that also allows for language-specific extensions.

The framework was orginally (Nivre et al., 2015) based on the Stanford dependencies (de Marneffe and Manning, 2008) and Google universal tags (Petrov et al., 2012), and used the CoNLL-X format (Buchholz and Marsi, 2006). The current Universal Dependencies is the second version, and is based on the revised versions of all the frameworks it was originally based on. It has an extended tagset, uses universal Stanford dependencies, and the CoNLL-U format[1]. The CoNLL-U format includes both the UD POS tags and optional language specific POS tags. The UD tagset has 17 tags, shown in table 2.1.

Currently there are more than 100 treebanks available using the Universal Dependencies framework, in more than 70 languages.

2.3.2 PyTorch

PyTorch (Paszke et al., 2019) is a Python package that provides tensor computations with GPU support and a framework for making deep neural networks. It allows the user to build dynamic networks by recording the operations as they are done and calculating the gradients based on them. This makes for a more flexible experience than static networks where the models have to be defined and compiled before being run.

It has a neural network library with implementations of common neural networks and an optimization module with commonly used optimization algorithms.

PyTorch was designed to be well-integrated in Python and intuitive to use. This and its dynamic network approach were the main reasons this framework was chosen over other alternatives, e.g. TensorFlow (Abadi et al., 2015).

[1] A full description is available at https://universaldependencies.org/format.html

ADJ	Adjective
ADP	Adposition
ADV	Adverb
AUX	Auxiliary
CCONJ	Coordinating conjunction
DET	Determiner
INTJ	Interjection
NOUN	Noun
NUM	Numeral
PART	Participle
PRON	Pronoun
PROPN	Proper noun
PUNCT	Punctuation
SCONJ	Subordinating conjunction
SYM	Symbol
VERB	Verb
X	Other

Table 2.1: Universal Part-of-Speech tags

3 Related Work

Many machine learning algorithms have been applied to Natural Language Processing (NLP) tasks over the years, but the focus here will be on Deep Learning algorithms. There has been a lot of work on this in recent years, and this chapter will describe that which is most relevant to the work in this report.

3.1 Natural Language Processing (Almost) from Scratch

Collobert et al. (2011) used a multitasking neural network for multiple NLP tasks, Part-of-Speech tagging (POS tagging) among them, with almost no preprocessing of the words. They used look-up tables to get vectors of word embeddings, and tried one approach in which they used a convolutional layer on the words and another where they concatenated the vectors of the word embeddings in a window that passed over the sentence. They also tried two different scoring functions, one for individual words and one for the whole sentence. The sentence-level scoring function performed better.

They trained the network as a language model on a large amount of unlabeled data, which greatly improved the word embeddings in the look-up table but took a long time. Collobert et al. then used the resulting network for jointly training the supervised tasks with the first layers shared and the top layers specialized. They used Stochastic Gradient Descent (SGD) for this, and picked random examples from different tasks for updates. The unsupervised training of the language models appeared to have the greatest effect on performance, but the multitasking also improved performance on some tasks.

For further improvement, Collobert et al. tried to include linguistic features, two-character suffixes in the case of POS tagging, and to use ten differently initialized networks in an ensemble. Both resulted in some improvement. In the end, they implemented their architecture in C and called it Senna (Semantic/syntactic Extraction using a Neural Network Architecture).

There were many interesting ideas in the work of Collobert et al.. First, there was the good effect of the unsupervised training of the language models, even if the training took a long time. Then there was the idea of training a multitasking network on several NLP tasks simultaneously, which seemed useful. Incorporating linguistic features was also helpful, though the network did already perform well without them. The ensemble idea, however, seemed rather computationally expensive and the networks were not necessarily different enough to justify it and give a good boost in performance. They also used a Convolutional Neural Network (CNN).

3.2 Learning Character-Level Representations for Part-of-Speech Tagging

dos Santos and Zadrozny (2014) extended the work of Collobert et al. (2011) with a convolutional layer to get character-level embeddings in addition to the word-level embeddings. Like Collobert et al. (2011), dos Santos and Zadrozny also used unlabeled data to learn the word-level embeddings and they also used the sentence-level scoring function. They applied their model to POS tagging on English and Portuguese with good results, with 97.32% accuracy on the Penn Treebank (Marcus et al., 1993) for English and 97.47% accuracy on the Mac-Morpho (Aluísio et al., 2003) corpus for Portuguese. Some examples they provide for comparison are Fernandes (2012), who achieved 97.12% accuracy on the Mac-Morpho corpus, and Søgaard (2011), who had 97.50% accuracy on the Penn Treebank.

The character-level convolutional layer was useful for extracting morphological information from the words, which otherwise might have been given as additional features to the network. As such, dos Santos and Zadrozny demonstrated that one could achieve state-of-the-art performance in POS tagging without selecting relevant word features by hand.

3.3 Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation

Ling et al. (2015) used a Bi-directional Long Short-Term Memory (bi-LSTM) network on characters to make word embeddings. They used it for POS tagging and language modeling. For the POS tagging they ran a second bi-LSTM on the word embeddings and used the softmax function for output labels. They tested their models on five languages: English, Portuguese, Catalan, German and Turkish (Marcus et al., 1993; Afonso et al., 2002; Martí et al., 2007; Brants et al., 2002; Atalay et al., 2003). Ling et al. had good results on all of them, with 97.36% accuracy on the English Penn Treebank. They also tried pre-trained embeddings and added word features and achieved improvement.

A notable thing about the model of Ling et al. was that the character-level embeddings required less space than word-level embeddings, as there are fewer characters than words. This came at the cost of more computation at runtime, however, as getting the word embeddings from character-level required running a bi-LSTM rather than a simple look-up. The authors suggested caching as a measure against this. Like dos Santos and Zadrozny (2014), the model Ling et al. used was able to learn linguistic features at a character level. However, Ling et al. used a bi-LSTM rather than a convolutional layer. The bi-LSTM might have worked better for long sequences as it preserves previous information rather than only finding information in a given range. When using a convolutional layer, there was the danger that important information could fall outside the chosen window size.

3.4 Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss

Plank et al. (2016) studied the effectiveness of bi-LSTMs for POS tagging for 22 different languages and also measured their sensitivity to data amount and noise. They used version 1.2 of the Universal Dependencies data set (Nivre et al., 2015) for their experiments. Plank et al. also used an auxiliary loss function to improve performance for words that were rare or not encountered in training, and used both word- and character-level embeddings. The character-level embeddings where achieved with a bi-LSTM like Ling et al. (2015) and the auxiliary loss function was meant to force the network to estimate word frequency along with the tags.

The combined word- and character-level embeddings worked well and did better than either alone. Plank et al. also tried to combine embeddings from characters and bytes, but this did not work as well as the embeddings from the characters alone. While the combined character- and word-level embeddings performed best, the character-level embeddings did better than the word-level embeddings alone. The auxiliary loss function did not seem to have any major effects on the results. As for their experiments with data amounts, their model performed consistently better than Conditional Random Fields (CRF) even for low amounts of data. The TnT tagger (Brants, 2000) performed better than their model on low amounts of data, but for most languages bi-LSTM seemed to catch up within a thousand sentences. The performance drop with increased noise was mostly the same for bi-LSTM and TnT up till 30%, at which point TnT did better.

An interesting part of the work of Plank et al. was that they used character-level information along with word-level information like dos Santos and Zadrozny (2014), but used bi-LSTM like Ling et al. (2015). Plank et al. also compared the performance of their model in more languages than the others. Their results on the robustness of bi-LSTM with respect to lack of data and noise in data were also interesting, as one might have expected the bi-LSTM to be more sensitive to such difficulties.

3.5 Character-Aware Neural Language Models

Kim et al. (2016) used a convolutional layer for character-level information followed by a max-over-time pooling layer, a highway layer and a Long Short-Term Memory (LSTM) layer. The highway layer passed on some dimensions of its input directly to its output, in this way similar to residual connections. The task they applied their model to was language modeling. They tested it on English, Czech, German, Spanish, French, Russian and Arabic,[1] with one smaller and one bigger data set for all except Arabic, for which they only had one. Kim et al. achieved good results and found that character-level embeddings worked better than word-level embeddings and that the highway layer helped performance. They also achieved better results with more data. They tried

[1]The Arabic data came from the News-Commentary corpus: http://opus.nlpl.eu/News-Commentary.php and the Non-Arabic data came from the 2013 ACL Workshop on Machine Translation and is available at: http://www.statmt.org/wmt13/translation-task.html

combining character-level embeddings and word-level embeddings like dos Santos and Zadrozny (2014) and Plank et al. (2016), but unlike them, Kim et al. found that it led to worse performance.

This work was interesting because Kim et al. used a convolutional network for character-level information combined with a highway layer and LSTM for word-level information. Like Ling et al. (2015), Kim et al. used only character-level embeddings and found that their model had less parameters than one using word-level embeddings, but at the cost of more computations. The highway layer seemed to work well in combination with the convolutional layer.

3.6 An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling

Bai et al. (2018) wished to explore the use of convolutional networks in sequence processing and decided to compare a farily simple CNN architecture with Recurrent Neural Network (RNN), LSTM and Gated Recurrent Unit (GRU), which are standard base algorithms for sequence processing today. In other words, they wished to compare a generic CNN architecture to generic recurrent network architectures. To do so they chose a number of tasks commonly used to evaluate recurrent networks.

The chosen tasks were concerned with sequence modeling, which is a kind of sequence-to-sequence prediction where the output can only depend on input received at current or previous timesteps. With these considerations they used causal convolutions to avoid using future inputs, as well as appropriate zero-padding and hidden layers of the same size as the input to ensure the output sequence would have the same length.

Bai et al. note that their network is similar to previous time-delay networks, but uses padding for consistent sizes as well as some more recent improvements on the CNN architecture that allows for longer effective history. The network of Bai et al. uses residual blocks and dilation that increases exponentially with network depth. When training the network, they used weight normalization and spatial dropout . They called their network a Temporal Convolutional Network (TCN).

The article lists some benefits and drawbacks of using TCNs. The structure of the network allows for parallel training and processing and gives stable gradients, unlike recurrent network structures. The receptive field of the network is flexible and can be changed by changing the network depth, the dilation factor, and the filter size. The network has relatively low memory requirements for training and takes variable length input. Some drawbacks are that the network can have higher memory requirements during evaluation and that domain transfer requires changing the parameters to fit the new domain.

The article describes eight sequence modeling tasks that the TCN, RNN, LSTM and GRU were tested on. It is noted that these architectures are not necessarily state-of-the-art for these tasks. The number of hidden units were chosen to keep the models about the same size as the LSTM model. The GRU network performed well on the adding

problem, but slightly worse than the TCN, and the LSTM network did best on the Penn Treebank, but in all the other tasks the TCN clearly outperformed the other networks.

3.7 Language Modeling with Gated Convolutional Networks

Dauphin et al. (2017) introduced Gated Linear Unit (GLU) for convolutional networks. The authors noted that CNNs do not require forget gates like LSTMs, but could benefit from output gates, which was implemented in the form of GLUs. The GLU multiplied half of the output of the convolutional layer with the other half, thus gating one half of the output with the other. The GLU seemed to perform better than ReLU in most cases measured in their experiments.

For their experiments on the language modelling task, Dauphin et al. used convolutional layers and GLUs stacked into blocks, some implemented with bottleneck layers for greater computational efficiency, and residual connections adding the input to the output of a block. They call their model a Gated Convolutional Network (GCNN). Their model outperformed most of the LSTM-models they compared with on the language modelling task (on Google Billion words and WikiText-103 primarily, also Gigaword and Penn Treebank), and could process sentences faster as well as train and converge faster.

3.8 Part-of-Speech tagging for Norwegian

There has been some work on POS tagging for Norwegian that is relevant to mention. The Oslo-Bergen tagger (Hagen et al., 2000), developed at the University of Oslo and Uni Computing in Bergen, is a commonly used tagger based on Constraint Grammar. Its development was part of the early work on NLP for Norwegian, which also involved the creation of various digital language resources. A tagset was also created, with 358 tags. The earliest versions gave out still ambiguous output, but the most recent version (Johannessen et al., 2012) has an additional statistical module that removes the remaining ambiguities. The Oslo-Bergen tagger can tag both the Norwegian written standards, Bokmål and Nynorsk, but the latest statistical module is only for Bokmål. Using the statistical module, the Oslo-Bergen tagger has an accuracy of 96.5% for Bokmål on the evaluation text used.

Some other works include Marco (2014), Johansen (2016), and Kåsen et al. (2019). Marco used the existing FreeLing (Padró and Stanilovsky, 2012) to create a statistical tagger for Norwegian. They used a simplified version of the Oslo-Bergen tagset and achieved an accuracy of 97.3% for POS tagging. Johansen trained Google's SyntaxNet (Andor et al., 2016) on Norwegian Bokmål and Nynorsk using the Norwegian Dependency Treebank. SyntaxNet is Feed-Forward Neural Network (FFN) based and uses global normalization. They achieved an F-score of 97.54% for Bokmål. Kåsen et al. tests five different taggers on transcriptions of Norwegian dialects (Øvrelid et al., 2018). The best performing tagger was the bi-LSTM model of Plank et al. (2016) with 97.33% accuracy.

The Norwegian Universal Dependencies treebanks have been used in some other experiments. Øvrelid and Hohle (2016) converted the Norwegian Dependency Treebank to Universal Dependencies and reported the first results of testing on it. They used SVMTool (Giménez and Màrquez, 2004) for the testing and achieved 96.82% on the POS tagging task.

Straka et al. (2016) developed a pipeline for processing of text in CoNLL-U format that performs multiple NLP tasks, POS tagging among them. It was tested on many of the UD treebanks, and it had 97.2% accuracy on Norwegian and 94.5% accuracy on English. The tagger part was based on MorphoDiTa (Straková et al., 2014), while GRUs and LSTMs were used for lower level processing. They called their pipeline "UDPipe".

Velldal et al. (2017) worked on joint dependecy parsing for Bokmål and Nynorsk on the Universal Dependencies treebanks, with POS tagging as part of the process. Their best results were achieved using UDPipe, separately rather than joint, with 97.07% accuracy on Bokmål.

Bohnet et al. (2018) tested a Meta-BiLSTM, which used character and word based encodings with separate loss functions, on the language specific tags of the UD treebanks. They report an accuracy of 99.75% on Norwegian Bokmål.

It should be noted that in nearly all these experiments the data, the tagset, or both were different. This means the results cannot readily be compared.

4 Architecture

The architecture that was tested was a Gated Convolutional Network (GCNN) (see section 3.7) implemented in Pytorch and the data used was from the Universal Dependencies project (see section 2.3.1). The architecture used only character-level embeddings.

4.1 Data and preprocessing

The data used in the experiments came from the Universal Dependencies project and was already in a standardized format (see section 2.3.1). The only preprocessing required was to separate the words and the part-of-speech tags from the other data and transform them into a suitable input format. The data was also already divided into a develpment set, a training set, and a test set. As the data split seemed reasonable and contained similar variation the existing partioning was used.

Rather than using sentences as input to the network, the sentences were concatenated and divided into sections of equal length for less variation in sample sizes. On the other hand, the words remained variable length, which meant the samples could not be stored as tensors and had to be passed into the network one by one.

The data of the UD treebanks were divided into sentences. These were concatenated into one text, then divided into sequences of same length. The sequences were then shuffled before training, using a fixed seed so all tests with the same sequence length had the same order of sequences.

4.2 Network architecture

The basic architecture was a one-dimensional Convolutional Neural Network (CNN). For nonlinearity, Gated Linear Units (GLUs) (see section 3.7) were used after the convolutional layers. The layers were organized into residual blocks with one or more gated convolutional layer between the residual connections. The input to each convolutional layer was padded with zero-padding at both ends to ensure the sequence length was preserved.

The architecture (see fig. 4.1) was composed of two parts in sequence: a character-level network and a word-level network. The output of the character-level part was a single tensor per word. The output of the word-level network was a vector of log-probabilities of part-of-speech tags. The first layer of the character-level network was an embedding layer which mapped from integer character representations to real-valued character vectors. The vectors were then passed through one or more residual blocks

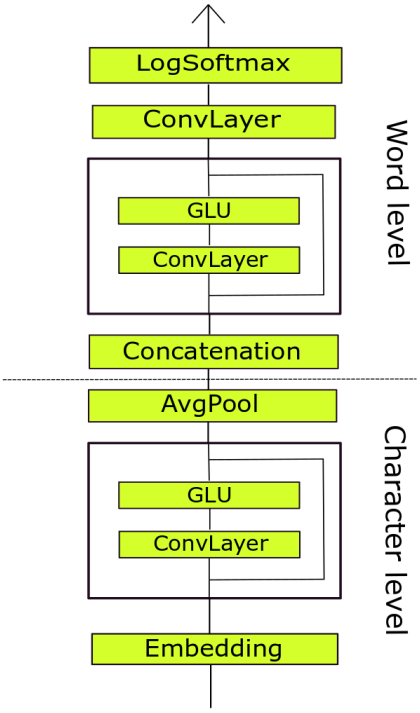


Figure 4.1: The basic network architecture

before the output was passed through a layer of average pooling to reduce the word length dimension to one.

The input to the network was a list of variable length words with integer representations of the characters. All the characters of the word were passed through the character-level part of the network together. The first part of the character-level network was an embedding layer mapping the integer representations of the characters to vectors.

Since the character-level part of the network gave out tensors of the same size, the tensors were concatenated for input to the word-level part of the network. The word-level part consisted of one or more residual blocks, followed by a convolutional layer to reduce the output to the right number of classes. The output was then passed through a LogSoftmax layer to obtain log-probabilities.

4.3 Input parameters

The program took many variables relating to the architecture and training setup as input. For both the character-level part and the word-level part the program took lists of kernel size (see section 2.2.2), dilation, and number of hidden units, where each item

in the lists corresponded to one residual block and adding items meant adding residual blocks. Naturally, the length of the lists for one part of the network had to correspond with the length of the other lists meant for that part of the network. If left undefined, each list would default to a single value. The size of the character embeddings was also given as input.

The files with the data to be used were non-optional input parameters. A file with training data, a file with test data, and a file containing the part-of-speech tags to be used were all required to run the tests. In addition, a file with a predefined character vocabulary could be given. If a character vocabulary was not given, it was constructed from the training data.

The training setup required a number of values to be specified. The sequence length for text-sequences was given this way, as was the learning algorithm, with the alternatives being Adagrad, Adam, and SGD (see section 2.2.1). The learning rate for the algorithm, as well as momentum if applicable, could also be given. Dropout could be used by giving a non-zero value for it.

5 Experiments and Results

5.1 Experimental Plan

The goal of the experiments was to see how well a Convolutional Neural Network (CNN) performed on the Part-of-Speech tagging (POS tagging) task. The first thing to do was to develop an architecture to use. This was accomplished by implementing and testing different variations on the development part of the Bokmål part of the Norwegian UD treebank. The second experiment was to train and test the developed architecture on the training and testing part of the same treebank. Different values of hyperparameters were tested to find out which gave the best result and to see what results might be expected of the architecture. The third experiment was to train and test the architecture on an English UD treebank with the hyperparameters determined in the former experiment to test its performance on another language. Finally, a simpler CNN architecture was tested for comparison.

5.1.1 Initial experiments

The goal of the initial experiments was to develop a good network architecture and find good hyperparameters. There were a lot of adjustable parameters and parts, so random search (see section 2.2.1) was attempted using the development set of a Norwegian UD treebank to determine the basic experimental setup for the following experiments. No fixed seed was used, and the search was constrained to run only 51 iterations due to the fact that it was time consuming to run it on the PC. All results were saved for later comparison, and the best and worst results were used to guide choice of hyperparameters for the later experiments.

There was also an attempt to compare the architecture using GLU and the architecture using ReLU using 5-fold crossvalidation. The data used was the predefined development set of the Bokmål part of the Norwegian UD treebank. A fixed seed was not used, and 5 folds were used instead of the more common 10 because it took a long time to run on the PC.

The initial experiments were not performed rigorously as they were only meant to guide development of the architecture for the later experiments.

5.1.2 Testing on one of the Norwegian UD databanks

The results of the initial experimentation encouraged a more systematic investigation of the effects of different hyperparameters, so this was done on the training and test parts of the Bokmål part of the Norwegian UD treebank. The goals were to gain better

understanding of the effects of the different hyperparameters as well as measure the performance of the architecture on the Norwegian dataset. The treebank used was the Bokmål version, based on the Bokmål section of the Norwegian Dependency Treebank, with 310k tokens.

5.1.3 Testing on one of the English UD databanks

Some tests were run on one of the English UD treebannks to test how the architecture performed on a different language. There were four English UD treebanks. Of them, the GUM treebank (Zeldes, 2017) was chosen as it was the second largest of the listed English treebanks with 101k tokens, and had the most variation in text sources. The best performing hyperparameters from the previous experiment were used, as well as base parameters.

5.1.4 Testing ReLU architecture

During the initial development and experiments a version of the architecture using ReLU activation instead of Gated Linear Unit (GLU) was considered. The initial experiments indicated that the Gated Convolutional Network (GCNN) performed better, but at a steep rise in parameters. It was decided to run a larger test to see clearer how the CNN model with ReLU units performed in comparison to the GCNN.

5.2 Experimental Setup

The architecture in all the experiments except the initial experimentation were trained with the Adagrad algorithm (see section 2.2.1) for 15 epochs with an initial learning rate of 0.01. Only the training part of the treebanks were used for training while the accuracy given in the results was measured on the test part of the treebanks. The experiments were run on a PC with an Intel Core i5 8th Gen. This led to the experiments generally taking 5-8 hours per complete run, which ended up reducing the scope of the experiments somewhat. All experiments, excepting some of the initial experiments, were run with fixed seed due to the stochastic nature of training neural networks. Negative log-likelihood was used as loss function.

Where dilation was used it was doubled for every residual block, so with 2 residual blocks the dilation would be 1 for the first and 2 for the second. The character part and the word-level part were considered separately, so both started with 1 at the first block. It should be noted that for the GCNN the effective number of units is half the given number due to the use of GLUs, since GLUs use one half of the output to gate the other half (see section 3.7).

The performance measures reported here are accuracy and macro-F1. Accuracy was chosen for ease of understanding and comparison with results reported by others. The F1-score was computed by averaging the F1-scores of the individual word-classes and was chosen for an alternative measurement and comparison between results.

The UDPOS tagset (see section 2.3.1) was used in all experiments.

Fold	GLU	ReLU
1	0.921	0.884
2	0.905	0.913
3	0.917	0.905
4	0.917	0.901
5	0.930	0.888
Average	0.918	0.898

Table 5.1: Accuracy in 5-fold crossvalidation for GLU and ReLU

Sq.L	Emb.	Hidden Units	Dilation	Kernel Size	Accuracy
80	400	300 | 300	2 | 1	4 | 3	0.934
40	100	300 400 400 | 100	4 4 2 | 1	2 2 2 | 3	0.934
50	100	100 200 | 100 200	8 1 | 1 2	3 4 | 3 2	0.927
60	200	200 400 100 | 200	1 2 4 | 4	2 2 2 | 3	0.925
30	400	100 400 | 200 100 100	4 2 | 1 2 1	2 2 | 4 2 4	0.925
50	400	100 | 200 300	8 | 8 8	3 | 4 3	0.833
80	100	100 | 200 300	8 | 8 8	2 | 4 3	0.830
30	200	200 | 300 400 200	8 | 1 2 8	2 | 3 4 4	0.817
80	400	400 | 200 300 100	8 | 2 2 8	4 | 4 2 3	0.813
80	100	200 400 200 | 100 400 400	8 8 1 | 2 4 4	2 2 2 | 4 4 3	0.692

Table 5.2: Top five and bottom five results from random search

5.3 Experimental Results

The results of the experiments are summarised here. For discussion of the results, see chapter 6.

5.3.1 Initial experiments

As the results of the 5-fold crossvalidation in table 5.1 shows, the architecture using GLUs scored better on average on the development set. It was therefore decided to use this architecture for further experiments.

The results of the random search were largely inconclusive, but some patterns could be seen. The best and worst results are given in table 5.2, and they suggest that the network performed better with more units in the character-level part than the word-level part, that small dilations might be beneficial, and that differences in kernel size had little effect. The base hyperparameters in table 5.3 were chosen based on the initial experimentation.

Hyperparameter	**Value**
Character embedding size	100
Character net size	2x200
POST net size	1x100
Training algorithm	Adagrad
Initial learning rate	0.1
Batch length	80
Dilation	1 (no dilation)
Kernel size	3
Training rounds	15

Table 5.3: The base hyperparameters for the first experiment

Variation	**Accuracy**	**F1**
No variation (base)	95.63%	89.8%
With dilation	95.69%	89.9%
Character embedding 200 units	95.78%	89.7%
Character net 1x400 units	95.37%	89.7%
Character net 4x100 units	95.52%	89.7%
Character net 3x200 units	**95.88%**	**90.8%**
POST net 1x200 units	95.56%	90.3%
POST net 2x200 units	95.76%	90.1%
Batch length 30	95.51%	89.9%
With two convolutional layers in residual block:		
No other changes	96.08%	90.2%
With dilation	96.12%	89.9%
Character embedding 200 units	95.95%	88.1%
Character net 1x400 units	96.06%	89.2%
Character net 2x100 units	95.55%	88.6%
Character net 4x100 units	95.28%	86.4%
POST net 1x200 units	96.02%	89.9%
POST net 2x200 units	95.54%	89.5%
With dilation and dropout 0.5	95.59%	90.7%
With dilation and dropout 0.2	**96.65%**	92.1%
With fixed character vocabulary and dilation	96.19%	88.6%
With fixed character vocabulary, dilation, and dropout 0.2	96.52%	**92.2%**

Table 5.4: Results of running the GCNN on the Norwegian data with different hyperparameters

Class	Precision	Recall	F1
ADJ	0.949	0.946	0.947
ADP	0.967	0.970	0.968
ADV	0.968	0.953	0.960
AUX	0.926	0.964	0.945
CCONJ	0.985	0.996	0.990
DET	0.964	0.975	0.969
INTJ	1.000	0.636	0.778
NOUN	0.952	0.971	0.961
NUM	0.988	0.988	0.988
PART	0.994	1.000	0.997
PRON	0.970	0.980	0.975
PROPN	0.965	0.920	0.942
PUNCT	1.000	1.000	1.000
SCONJ	0.920	0.911	0.915
SYM	1.000	0.727	0.842
VERB	0.939	0.930	0.934
X	0.737	0.133	0.225

Table 5.5: Per class measures for base setup with two layers on Norwegian data

5.3.2 Testing on one of the Norwegian UD databanks

The results of running some different variations of parameters compared to those given in table 5.3 are shown in table 5.4. The best result was 96.65% accuracy achieved by using dilations and dropout with two convolutional layers in the residual blocks. The F1-score of this result was the second best, but only by 0.1%. The variant with two layers in the residual blocks generally performed better in terms of accuracy, but the F1 scores were often somewhat lower.

There was not a lot of variation in the results, and the differences were not very large. Using dilation improved accuracy somewhat for both the one-layer and two-layer variant, but the F1-score of the two-layer variant was slightly lower. Increasing the character embedding size from 100 to 200 only gave better accuracy for the model with one layer in the residual blocks, and had a negative effect on the F1-score. Using a single block with 400 hidden units instead of two blocks with 200 hidden units gave a little worse results for all measures, but the accuracy of the one-layer block model suffered worst. Using 100 hidden units in four blocks also gave worse results in all cases. Adding another block with 200 hidden units with the one-layer block model improved both accuracy and F1-score.

Increasing the word-level part of the model to 200 hidden units lowered the accuracy for both variants, though the F1-score of the one-layer variant went up a little. Adding another block with 200 hidden units improved accuracy for the model with one-layer blocks, but gave worse results for the model with two-layer blocks.

Using two blocks of 100 units for the character-level part of the two-layer block model gave better results than using four blocks of 100 units. Using a sequence length of 30 rather than 80 for the one-layer block model reduced the accuracy.

Building on the best accuracy achieved so far, the two-layer block model with dilation was tested with dropout and fixed character vocabulary. Using a dropout probability of 0.5 reduced the accuracy and increased the F1-score, but a dropout probability of 0.2 improved both accuracy and F1 score. The fixed character vocabulary gave a higher accuracy and lower F1 score, but in conjunction with a dropout probability of 0.2 the accuracy came out a bit worse than the test using only dropout while the F1-score was slightly higher. The test results of the two-layer block with dilation and dropout probablity of 0.2 appear to be the best of the set.

Table 5.5 shows the results per class for the two layer block model with the hyperparameters of table 5.3. The X class, used for words that cannot be classified, has the lowest score by all given measures. The X class, the interjection class (INTJ), and the symbol class (SYM) were the classes with the fewest occurences and the lowest F1-scores.

Hyperparameters	**Accuracy**	**F1**
Experiment 1 base	92.55%	81.5%
Experiment 1 base with dilation and dropout 0.2	**92.99%**	**83.6%**
Experiment 1 base with dilation, dropout 0.2, and POST net 1x200 units	92.70%	83.4%

Table 5.6: Results of running the GCNN on the English data

5.3.3 Testing on one of the English UD databanks

The results of running the GCNN architecture on the English UD treebank with some different hyperparameters are shown in table 5.6. The result using the best hyperparameters from the former experiment was 92.99%. A test was run with the base parameters of table 5.3 for comparison, and a test with 200 hidden units for the word-level part of the model was run to see if the effect on results was the same. Both tests came out worse than the one using the best hyperparameters of section 5.3.2.

Table 5.7 shows that, as with the Norwegian databank, the lowest scores were those of the X class, the interjection class, and the symbol class. Those classes were the ones with fewest occurrences in this case as well.

5.3.4 Testing ReLU architecture

Running a ReLU architecture on the Norwegian data yielded a better result than the GCNN with one layer in the ResBlock of comparable size (see table 5.8) and similar accuracy to the GCNN with two layers in ResBlock of twice the size. It performed slightly worse on the English treebank, with 92.19% accuracy compared to 92.99% for the GCNN.

Class	Precision	Recall	F1
ADJ	0.844	0.768	0.804
ADP	0.962	0.962	0.962
ADV	0.849	0.848	0.848
AUX	0.949	0.966	0.957
CCONJ	0.993	0.991	0.992
DET	0.990	0.979	0.984
INTJ	0.000	0.000	0.000
NOUN	0.903	0.905	0.904
NUM	0.935	0.971	0.953
PART	0.927	0.991	0.958
PRON	0.954	0.970	0.962
PROPN	0.887	0.911	0.899
PUNCT	0.994	0.996	0.995
SCONJ	0.813	0.831	0.822
SYM	0.762	0.516	0.615
VERB	0.880	0.894	0.887
X	0.583	0.212	0.311

Table 5.7: Per class measures for base setup with two layers on English data

Hyperparameters	Data	Accuracy	F1
Experiment 1 base parameters with dilation	Norwegian	96.09%	90.3%
Experiment 1 base parameters with dilation	English	92.19%	82.6%

Table 5.8: Results of running the ReLU architecture

Architecture	Parameter count
Base GCNN double layer	1078917
Base GCNN single layer	537917
Base ReLU double layer	567917

Table 5.9: Parameter counts on the Norwegian data

5.3.5 Parameter count

The parameter count of the network rose very quickly when using GLUs, especially when using the version with two layers in the residual block. As shown in table 5.9, the GCNNs had far more parameters compared to the architecture using ReLUs when the same number of units was chosen for the layers. As mentioned in section 5.2, the GCNN also has effectively half the number of units per layer compared to the ReLU architecture. Due to the high parameter count the GCNN with two layers in the residual block took longer to train than the other architectures.

6 Discussion

In this final chapter the experiments and the results that were presented in chapter 5 and the architecture that was presented in chapter 4 will be discussed and evaluated. How well the research questions were answered and whether the goal of the research was achieved will also be considered. Finally, the work will be concluded and ideas for future work presented.

6.1 Discussion of Results

The results of the first experiments, described in section 5.3.1, were meant to guide development of the architecture and choice of hyperparameters rather than later evaluation. However, some trends were observed, some of which were further explored in later experiments. The architecture using Gated Linear Units (GLUs) appeared to perform better than the one using ReLUs, which was why GLUs were chosen for the main experiments. There were no major differences between different hyperparameter values, but there were some. Small dilations seemed to have a positive effect, which makes sense, since using larger values might cause information to be lost. On the other hand, no trends were observed with kernel sizes. Having a larger character-level part than word-level part appeared to work better than the opposite, and similar results were also observed in the next experiment.

The largest experiment involved testing different values of various hyperparameters on a Norwegian dataset, and expanded on the trends seen in the first experiments in a more systematic way. Especially of note is that for Norwegian, and perhaps English too, the architecture benefited from having a larger character-level part and a smaller word-level part. This might be because word-internal information was more useful than information about the context of the words, or it might have been that information did not pass properly through to the word-level part of the network as it had to pass through the character-level part first. The best performance was achieved using dropout, which forced the architecture to use more of its learning capability. Using a fixed character vocabulary also had a positive effect on the performance when used on its own, but combining dropout and fixed character vocabulary resulted in worse accuracy. This might have been due to both methods doing the work of improving character representations. Using two convolutional layers in the residual blocks gave better results at the cost of higher parameter count, and seems to strike a nice balance between too few and too many layers between the residual connections. The range of the results achieved was not very large, so one might expect further testing of the architecture not to deviate too far from that range.

As described in section 5.3.3, the implemented architecture performed worse on the English dataset than the Norwegian dataset. There are many possible reasons for this. It might be due to differences between the English treebank and the Norwegian one. The Norwegian treebank was three times as large as the English treebank used, which might be why the architecture did better on the Norwegian treebank. The English treebank also had greater variation in text types and sources, and the architecture might have struggled to handle this. Another possibility might lie in the fact that the hyperparameters were chosen based on experimentation on Norwegian data and might not work as well on English data. The architecture itself might also be less effective on English than Norwegian due to differences between the languages.

In the last experiment, the network using ReLU performed about as well as the Gated Convolutional Network (GCNN) with a lot fewer parameters. The GLUs appeared to perform better than the ReLUs in the initial experiments, and the main difference between initial experiments and the later ones was that the first used the development set of the Norwegian treebank while the latter used the Norwegian and English training and test sets. The development set was a lot smaller, so the results might have been misleading. The testing was not performed as rigidly in the initial experiments either, which might also have been important. In any case, the results of the experiments are not enough to draw any definite conclusion about which architecture is better.

6.2 Comparison of Results

Plank et al. (2016) achieved respectively 95.87% and 91.62% accuracy on the Norwegian and English treebanks of UD v1.2 with their bi-LSTM model using character embeddings only. The GCNN architecture explored in this report was tested on UD v2.1, and its best results were 96.65% accuracy on the Norwegian UD treebank and 92.99% on the English treebank. The GCNN achieved better results on this measure, but it used far more parameters and probably more adjustments as well. It should be noted that Plank et al. might have used different treebanks as they used an earlier version of the UD treebanks, and that the different versions of UD might have affected results as well.

State-of-the-art results on the UD treebanks are 98.5% accuracy on the Norwegian treebank and 96.1% on the English treebank, achieved by Heinzerling and Strube (2019) on UD v1.2 using BERT (Devlin et al., 2019), BPemb (Heinzerling and Strube, 2018), and character embeddings. The GCNN architecture did not achieve state-of-the-art results.

Bohnet et al. (2018) reported higher accuracy on the UD treebanks, but they used the language specific tags and not the UD tags, so the results cannot truly be compared.

Both Plank et al. and Heinzerling and Strube have a notable lower accuracy on English than on Norwegian, as did the Convolutional Neural Network (CNN) based architectures described in this report.

6.3 Discussion of Architecture

The architecture differs in many ways from the Temporal Convolutional Network (TCN) of Bai et al. (2018) and GCNN of Dauphin et al. (2017). The net drew input from both future and past inputs in the sequence and used zero-padding on both sides to preserve sequence length. The input itself was taken in the form of fixed length sequences, like in Bai et al. (2018), but variable length was allowed for individual words. Other options exist for all of these architectural decisions, but the focus was on developing a functional tagger, so the effects of these options and other architectural decisions were not explored in this thesis.

The exploration of different hyperparameters for the architecture was limited, both in which were chosen to be explored and what values of them were explored. However, given the relatively small range of the results, it seems unlikely that other hyperparameters would yield major differences in the results.

The GCNN ended up suffering from a likely excessive and rapidly growing parameter count. All tested variants of the CNN-based architecture required fairly few training rounds, but the high parameter count of the GCNN also impacted training time. In Wu et al. (2019) they propose lightweight convolutions which greatly reduce the parameter count of a GCNN, but at the time of testing Pytorch lacked optimization for this, which meant the training time grew many orders larger when an implementation was attempted. If the ReLU architecture achieves similar performance as the GCNN, as the last experiment seems to indicate, it would, with its lower parameter count, be preferable.

A notable aspect is that the architecture uses only character embeddings. These were not pre-trained. If they were, the performance might have improved, as in Plank et al. (2016) and Collobert et al. (2011). While character embeddings worked pretty well for Norwegian and English as tested here, one would not expect them to work as well with all languages. Their effectiveness would depend on how much information can be extracted from within the word. The experiments of Plank et al. (2016) show that whether character embeddings or word embeddings work best depends on the language. For all the languages Plank et al. tested, a combination of word embeddings and character embeddings worked better than either alone. In a similar way, word embeddings might have been tested here too.

The architecture might be sensitive to data amount and variation, though further tests would be required to confirm or deny this.

6.4 Discussion of Experiments

Though multiple experiments were run, only the GCNN on Norwegian data was explored in any kind of depth. The architecture was only tested on two different languages, and those languages were related. While the experiments ended up being fairly limited considering the many possible languages that could be tested, the experiments were enough to establish what performance might be expected from the GCNN architecture.

Different values for some hyperparameters for the GCNN architecture were explored when running the architecture on the Norwegian dataset. This exploration resulted in finding a set of values for those hyperparameters which performed well, but there were also many other variables related to the architecture and its training which were left unexplored. There was not enough time to test all the variables, but in retrospect it is clear that the experiments could have benefited from more rigorous testing in the early phase. Had the adjustments been tested with a systematic approach from the beginning, the capabilities of the CNN-based architectures could have been explored further. More thorough testing early on might also have led to the use of ReLU rather than GLU, or at least a better comparison of the two.

One of the variables not explored was the choice of learning algorithm. Different learning algorithms and learning rates were not tested in a systematic way beyond finding some that achieved acceptable results, because it was considered more interesting to explore other variables. The Adagrad algorithm was used for training in all the experiments. This was not necessarily the optimal choice, and notably most of the articles reviewed in chapter 3 used different algorithms.

The experiments were conducted to see what results might be expected from the architecture, and this goal was achieved. However, more extensive testing would be required to ensure statistical significance. Having statistically significant results would of course be preferable, but the fact that the experiments were run on a PC limited how many and how long experiments could be run. Still, it would have been useful to run at least one set of hyperparameters for every experiment with different seeds to get more statistically significant data.

6.5 Final Evaluation

The goal of the research, as stated in chapter 1, was to investigate the usefulness of CNNs for Part-of-Speech tagging (POS tagging). Judging by the results, the performance of a CNN-based architecture is comparable to that of a LSTM-based architecture on the POS tagging task, but one cannot say CNNs perform better than more commonly used sequence-based algorithms on POS tagging in general. A more rigorous comparison would be required to make any claims of this, or the contrary. Following on that, the CNN-based taggers did not achieve state-of-the-art results on the Norwegian dataset, so even though CNNs perform well enough to be useful, they cannot be said to be more useful than more common sequence-processing algorithms. Additionally, the tests only used the UD tagset, and tests with the language specific tagset were not performed, though the architecture allows for it. This was to better compare results with other experiments on the UD treebanks and to compare performance accross languages. As for how CNNs perform on different languages, the only other language tested was English. The architecture performed worse on the English treebank than the Norwegian treebank, but a similar drop in performance was seen in the experiments of others who used different algorithms.

Although the CNN-based architectures tested in the experiments did not impress

with their performance, they were nonetheless up to the task set to them. They did not measure up to state-of-the art results, but the results were comparable to the results of a bi-LSTM. The use of CNN for sequence processing is yet limited, but it is likely that better performance can be achieved if it continues. There are also many different mechanisms that can be used in conjunction with CNN, as well as different variants of CNN that can be further explored and which are often used in other contexts.

6.6 Future Work

As for the work detailed here, there are many interesting directions that might be explored further. The work here only used character embeddings, and as Plank et al. (2016) have shown, the combination of word embeddings and character embeddings tend to work better than either alone. This would therefore be useful to incorporate, and it would also be interesting to examine how the architecture performed with only word embeddings. It would also be interesting to try different ways of creating the embeddings, as well as pre-training them and the network as a whole. There are also many other variations on CNN and CNN-architectures that might be tested, for example lightweight convolutions. Additionally, it would be of interest to test what effect differences in the data, such as data amount or variation, would have on the training and final results. Finally, testing with more languages, and using tagsets specific to the chosen languages might be interesting.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

Susana Afonso, Eckhard Bick, Renato Haber, and Diana Santos. Floresta sintá(c)tica: a treebank for Portuguese. In *Proceedings of the Third International Conference on Language Resources and Evaluation (LREC 2002)*, pages 1698–1703, Las Palmas de Gran Canaria, Spain, 2002.

Sandra Aluísio, Jorge Pelizzoni, Ana Raquel Marchi, Lucélia de Oliveira, Regiana Manenti, and Vanessa Marquiafável. An account of the challenge of tagging a reference corpus for Brazilian Portuguese. In *Proceedings of the 6th International Conference on Computational Processing of the Portuguese Language*, PROPOR'03, pages 110–117, Faro, Portugal, 2003. URL `http://dl.acm.org/citation.cfm?id=1758748.1758769`.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 2442–2452, Berlin, Germany, 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1231. URL `https://www.aclweb.org/anthology/P16-1231`.

Nart B. Atalay, Kemal Oflazer, and Bilge Say. The annotation process in the Turkish treebank. In *Proceedings of 4th International Workshop on Linguistically Interpreted Corpora (LINC-03) at EACL 2003*, pages 33–38, Budapest, Hungary, 2003. URL `https://www.aclweb.org/anthology/W03-2405`.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *Computer Research Repository (CoRR)*, abs/1803.01271, 2018. URL `http://arxiv.org/abs/1803.01271`.

Bernd Bohnet, Ryan McDonald, Gonçalo Simões, Daniel Andor, Emily Pitler, and Joshua Maynez. Morphosyntactic tagging with a meta-BiLSTM model over context sensitive token encodings. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 2642–2652, Melbourne, Australia, 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1246. URL https://www.aclweb.org/anthology/P18-1246.

Sabine Brants, Stefanie Dipper, Silvia Hansen, Wolfgang Lezius, and George Smith. The TIGER treebank. In *Proceedings of the workshop on treebanks and linguistic theories*, Sozopol, Bulgaria, 2002.

Thorsten Brants. TnT — a statistical part-of-speech tagger. In *Proceedings of the Sixth Applied Natural Language Processing Conference ANLP-2000*, pages 224–231, Seattle, Washington, USA, 2000.

Sabine Buchholz and Erwin Marsi. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City, USA, June 2006. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W06-2920.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 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)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://www.aclweb.org/anthology/D14-1179.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *Computer Research Repository (CoRR)*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2461–2505, 2011.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *Proceedings of the 34th International Conference on Machine Learning*, ICML'17, pages 933–941, Sydney, New South Wales, Australia, 2017. JMLR.org. URL http://dl.acm.org/citation.cfm?id=3305381.3305478.

Marie-Catherine de Marneffe and Christopher D. Manning. The Stanford typed dependencies representation. In *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*, pages 1–8, Manchester, UK, August 2008. URL https://www.aclweb.org/anthology/W08-1301.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, Minneapolis, Minnesota, USA, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL `https://www.aclweb.org/anthology/N19-1423`.

Cícero Nogueira dos Santos and Bianca Zadrozny. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31 st International Conference on Machine Learning*, pages II–1818–II–1826, Beijing, China, 2014.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research (JMLR)*, 12:2121–2159, July 2011. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1953048.2021068`.

Eraldo Luis Rezende Fernandes. *Entropy guided feature generation for structure learning*. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brasil, Sept 2012.

Jesús Giménez and Lluís Màrquez. SVMTool: A general POS tagger generator based on Support Vector Machines. In *Proceedings of the 4th LREC*, Lisbon, Portugal, 2004.

Kristin Hagen, Janne Bondi Johannessen, and Anders Nøklestad. A constraint-based tagger for Norwegian. In *17th Scandinavian Conference of Linguistics. Odense Working Papers in Language and Communication 19*, pages 31–48, University of Southern Denmark, Odense, Denmark, 2000.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, Las Vegas, NV, USA, June 2016.

Benjamin Heinzerling and Michael Strube. BPEmb: Tokenization-free pre-trained subword embeddings in 275 languages. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, pages 2989–2993, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL `https://www.aclweb.org/anthology/L18-1473`.

Benjamin Heinzerling and Michael Strube. Sequence tagging with contextual and non-contextual subword representations: A multilingual evaluation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 273–291, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1027.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Janne Bondi Johannessen, Kristin Hagen, André Lynum, and Anders Nøklestad. Obt+stat. A combined rule-based and statistical tagger. In Gisle Andersen, editor, *Exploring Newspaper Language. Corpus compilation and research based on the Norwegian Newspaper Corpus*, pages 51–65. John Benjamins Publishing Company, 2012.

Bjarte Johansen. Training googles syntaxnet to understand norwegian bokmål and nynorsk. In *NIK: Norsk Informatikkonferanse*, pages 13–18, Bergen, Norway, November 2016.

Andre Kåsen, Anders Nøklestad, Kristin Hagen, and Joel Priestley. Tagging a Norwegian dialect corpus. In *Proceedings of the 22nd Nordic Conference on Computational Linguistics*, pages 350–355, Turku, Finland, 2019. Linköping University Electronic Press. URL `https://www.aclweb.org/anthology/W19-6140`.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush. Character-aware neural language models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)*, pages 2741–2749, Phoenix, Arizona, USA, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *Computer Research Repository (CoRR)*, 2014.

Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time-series. In *The handbook of brain theory and neural networks*, pages 276–279. The MIT Press, 1995.

Wang Ling, Tiago Luís, Luís Marujo, Ramón Fernandez Astudillo, Silvio Amir, Chris Dyer, Alan W Black, and Isabel Trancoso. 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*, page 1520–1530, Lisbon, Portugal, 2015.

Cristina Sánchez Marco. An open source part-of-speech tagger for Norwegian: Building on existing language resources. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 4111–4117, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL `http://www.lrec-conf.org/proceedings/lrec2014/pdf/801_Paper.pdf`.

Mitchell P. Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of English: The Penn treebank. *Computational linguistics*, 19(2): 313–330, 1993.

M Antonia Martí, Mariona Taulé, Lluis Márquez, and Manuel Bertran. Anotación semiautomática con papeles temáticos de los corpus CESS-ECE. *Procesamiento del Lenguaje Natural*, 38:67–76, 2007.

Joakim Nivre, Željko Agić, Maria Jesus Aranzabe, Masayuki Asahara, Aitziber Atutxa, Miguel Ballesteros, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Cristina

Bosco, Sam Bowman, Giuseppe G. A. Celano, Miriam Connor, Marie-Catherine de Marneffe, Arantza Diaz de Ilarraza, Kaja Dobrovoljc, Timothy Dozat, Tomaž Erjavec, Richárd Farkas, Jennifer Foster, Daniel Galbraith, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Yoav Goldberg, Berta Gonzales, Bruno Guillaume, Jan Hajič, Dag Haug, Radu Ion, Elena Irimia, Anders Johannsen, Hiroshi Kanayama, Jenna Kanerva, Simon Krek, Veronika Laippala, Alessandro Lenci, Nikola Ljubešić, Teresa Lynn, Christopher Manning, Cătălina Mărănduc, David Mareček, Héctor Martínez Alonso, Jan Mašek, Yuji Matsumoto, Ryan McDonald, Anna Missilä, Verginica Mititelu, Yusuke Miyao, Simonetta Montemagni, Shunsuke Mori, Hanna Nurmi, Petya Osenova, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Slav Petrov, Jussi Piitulainen, Barbara Plank, Martin Popel, Prokopis Prokopidis, Sampo Pyysalo, Loganathan Ramasamy, Rudolf Rosa, Shadi Saleh, Sebastian Schuster, Wolfgang Seeker, Mojgan Seraji, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Kiril Simov, Aaron Smith, Jan Štěpánek, Alane Suhr, Zsolt Szántó, Takaaki Tanaka, Reut Tsarfaty, Sumire Uematsu, Larraitz Uria, Viktor Varga, Veronika Vincze, Zdeněk Žabokrtský, Daniel Zeman, and Hanzhi Zhu. Universal Dependencies 1.2, 2015. URL `http://hdl.handle.net/11234/1-1548`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic.

Joakim Nivre, Željko Agić, Lars Ahrenberg, Lene Antonsen, Maria Jesus Aranzabe, Masayuki Asahara, Luma Ateyah, Mohammed Attia, Aitziber Atutxa, Liesbeth Augustinus, Elena Badmaeva, Miguel Ballesteros, Esha Banerjee, Sebastian Bank, Verginica Barbu Mititelu, John Bauer, Kepa Bengoetxea, Riyaz Ahmad Bhat, Eckhard Bick, Victoria Bobicev, Carl Börstell, Cristina Bosco, Gosse Bouma, Sam Bowman, Aljoscha Burchardt, Marie Candito, Gauthier Caron, Gülşen Cebiroğlu Eryiğit, Giuseppe G. A. Celano, Savas Cetin, Fabricio Chalub, Jinho Choi, Silvie Cinková, Çağrı Çöltekin, Miriam Connor, Elizabeth Davidson, Marie-Catherine de Marneffe, Valeria de Paiva, Arantza Diaz de Ilarraza, Peter Dirix, Kaja Dobrovoljc, Timothy Dozat, Kira Droganova, Puneet Dwivedi, Marhaba Eli, Ali Elkahky, Tomaž Erjavec, Richárd Farkas, Hector Fernandez Alcalde, Jennifer Foster, Cláudia Freitas, Katarína Gajdošová, Daniel Galbraith, Marcos Garcia, Moa Gärdenfors, Kim Gerdes, Filip Ginter, Iakes Goenaga, Koldo Gojenola, Memduh Gökırmak, Yoav Goldberg, Xavier Gómez Guinovart, Berta Gonzáles Saavedra, Matias Grioni, Normunds Grūzītis, Bruno Guillaume, Nizar Habash, Jan Hajič, Jan Hajič jr., Linh Hà Mỹ, Kim Harris, Dag Haug, Barbora Hladká, Jaroslava Hlaváčová, Florinel Hociung, Petter Hohle, Radu Ion, Elena Irimia, Tomáš Jelínek, Anders Johannsen, Fredrik Jørgensen, Hüner Kaşıkara, Hiroshi Kanayama, Jenna Kanerva, Tolga Kayadelen, Václava Kettnerová, Jesse Kirchner, Natalia Kotsyba, Simon Krek, Veronika Laippala, Lorenzo Lambertino, Tatiana Lando, John Lee, Phuong Lê Hồng, Alessandro Lenci, Saran Lertpradit, Herman Leung, Cheuk Ying Li, Josie Li, Keying Li, Nikola Ljubešić, Olga Loginova, Olga Lyashevskaya, Teresa Lynn, Vivien Macketanz, Aibek Makazhanov, Michael Mandl, Christopher Manning, Cătălina Mărănduc, David Mareček, Katrin Marheinecke, Héctor Martínez Alonso, André Martins, Jan Mašek, Yuji Matsumoto, Ryan

McDonald, Gustavo Mendonça, Niko Miekka, Anna Missilä, Cătălin Mititelu, Yusuke Miyao, Simonetta Montemagni, Amir More, Laura Moreno Romero, Shinsuke Mori, Bohdan Moskalevskyi, Kadri Muischnek, Kaili Müürisep, Pinkey Nainwani, Anna Nedoluzhko, Gunta Nešpore-Bērzkalne, Luong Nguyễn Thị, Huyền Nguyễn Thị Minh, Vitaly Nikolaev, Hanna Nurmi, Stina Ojala, Petya Osenova, Robert Östling, Lilja Øvrelid, Elena Pascual, Marco Passarotti, Cenel-Augusto Perez, Guy Perrier, Slav Petrov, Jussi Piitulainen, Emily Pitler, Barbara Plank, Martin Popel, Lauma Pretkalniņa, Prokopis Prokopidis, Tiina Puolakainen, Sampo Pyysalo, Alexandre Rademaker, Loganathan Ramasamy, Taraka Rama, Vinit Ravishankar, Livy Real, Siva Reddy, Georg Rehm, Larissa Rinaldi, Laura Rituma, Mykhailo Romanenko, Rudolf Rosa, Davide Rovati, Benoît Sagot, Shadi Saleh, Tanja Samardžić, Manuela Sanguinetti, Baiba Saulīte, Sebastian Schuster, Djamé Seddah, Wolfgang Seeker, Mojgan Seraji, Mo Shen, Atsuko Shimada, Dmitry Sichinava, Natalia Silveira, Maria Simi, Radu Simionescu, Katalin Simkó, Mária Šimková, Kiril Simov, Aaron Smith, Antonio Stella, Milan Straka, Jana Strnadová, Alane Suhr, Umut Sulubacak, Zsolt Szántó, Dima Taji, Takaaki Tanaka, Trond Trosterud, Anna Trukhina, Reut Tsarfaty, Francis Tyers, Sumire Uematsu, Zdeňka Urešová, Larraitz Uria, Hans Uszkoreit, Sowmya Vajjala, Daniel van Niekerk, Gertjan van Noord, Viktor Varga, Eric Villemonte de la Clergerie, Veronika Vincze, Lars Wallin, Jonathan North Washington, Mats Wirén, Tak-sum Wong, Zhuoran Yu, Zdeněk Žabokrtský, Amir Zeldes, Daniel Zeman, and Hanzhi Zhu. Universal Dependencies 2.1, 2017. URL `http://hdl.handle.net/11234/1-2515`. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics (ÚFAL), Faculty of Mathematics and Physics, Charles University, Prague, Czech Republic.

Lilja Øvrelid and Petter Hohle. Universal dependencies for Norwegian. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1579–1585, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA). URL `https://www.aclweb.org/anthology/L16-1250`.

Lilja Øvrelid, Andre Kåsen, Kristin Hagen, Anders Nøklestad, Per Erik Solberg, and Janne Bondi Johannessen. The LIA treebank of spoken Norwegian dialects. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan, May 2018. European Language Resources Association (ELRA). URL `https://www.aclweb.org/anthology/L18-1710`.

Lluís Padró and Evgeny Stanilovsky. FreeLing 3.0: Towards wider multilinguality. In *Proceedings of the Language Resources and Evaluation Conference (LREC 2012)*, Istanbul, Turkey, May 2012. ELRA.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang,

Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., Vancouver, Canada, 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal Part-of-Speech tagset. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 2089–2096, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA). URL `http://www.lrec-conf.org/proceedings/lrec2012/pdf/274_Paper.pdf`.

Barbara Plank, Anders Søgaard, and Yoav Goldberg. Multilingual part-of-speech tagging with bidirectional long short-term memory models and auxiliary loss. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, page 412–418, Berlin, Germany, 2016.

Anders Søgaard. Semisupervised condensed nearest neighbor for part-of-speech tagging. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Short Papers*, pages 48–52, Portland, Oregon, USA, 2011. Association for Computational Linguistics. URL `http://dl.acm.org/citation.cfm?id=2002736.2002748`.

Milan Straka, Jan Hajič, and Jana Straková. 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)*, pages 4290–4297, Portorož, Slovenia, May 2016. European Language Resources Association (ELRA). URL `https://www.aclweb.org/anthology/L16-1680`.

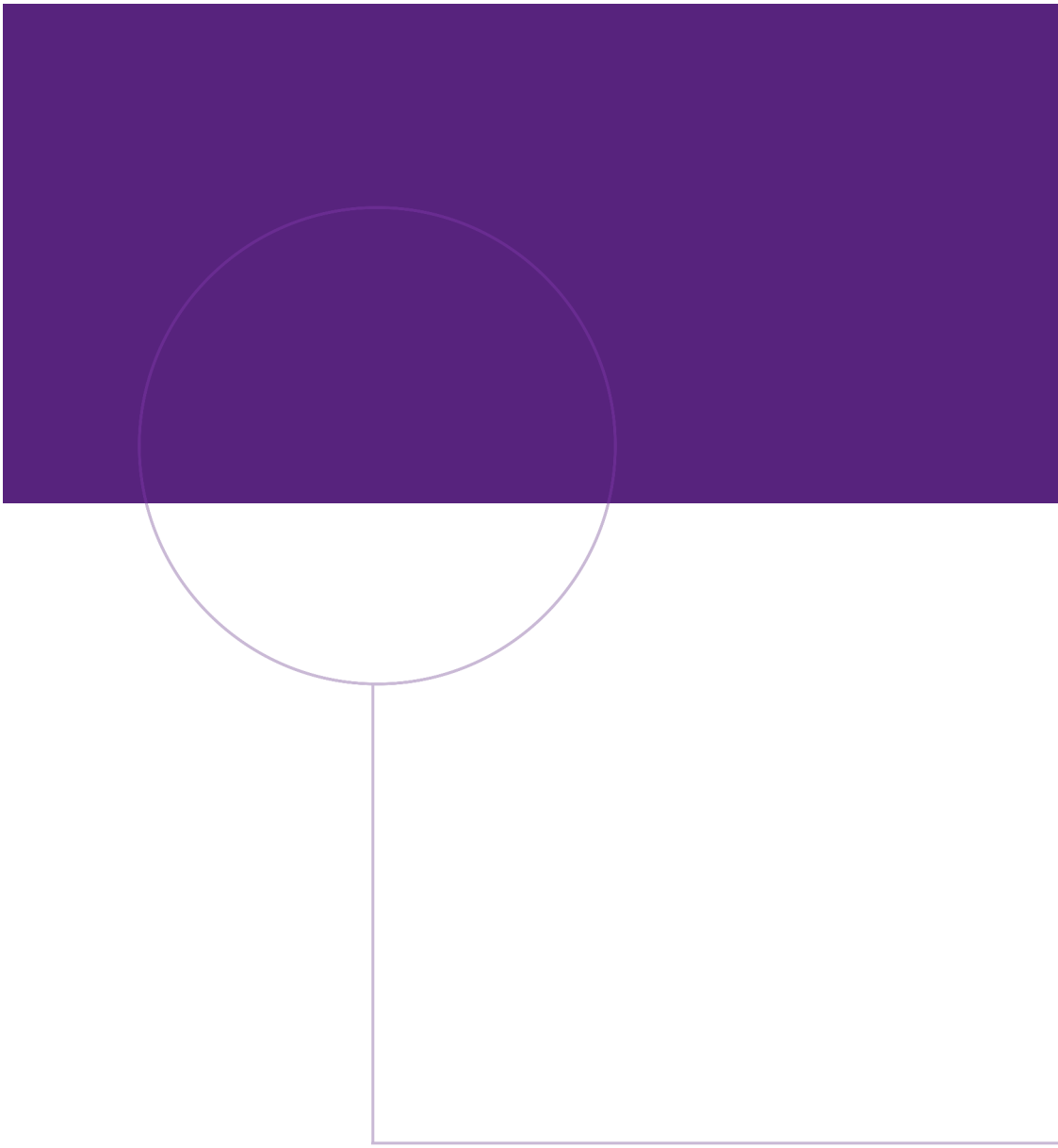
Jana Straková, Milan Straka, and Jan Hajič. Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 13–18, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P14/P14-5003.pdf`.

Erik Velldal, Lilja Øvrelid, and Petter Hohle. Joint UD parsing of Norwegian Bokmål and Nynorsk. In *Proceedings of the 21st Nordic Conference on Computational Linguistics*, pages 1–10, Gothenburg, Sweden, May 2017. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/W17-0201`.

Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. *Computer Research Repository (CoRR)*, 2019.

F. Yu and V. Koltun. Multi-Scale Context Aggregation by Dilated Convolutions. *Computer Research Repository (CoRR)*, November 2015.

Amir Zeldes. The GUM corpus: Creating multilayer resources in the classroom. *Language Resources and Evaluation*, 51(3):581–612, 2017. doi: http://dx.doi.org/10.1007/s10579-016-9343-x.



NTNU

Kunnskap for en bedre verden