The University of Edinburgh

MSc Artificial Intelligence

MSc Thesis

Exploring Word Embeddings for Unsupervised Part-of-Speech tagging

Author:

Felicia Liu

A thesis submitted in fulfilment of the requirements

for the degree of Master of Science

School of Informatics

August 19, 2016

Declaration of Authorship

I, Felicia Liu, declare that this thesis titled, "Exploring Word Embeddings for Unsupervised Part-of-Speech Tagging" was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Signed:

Date:

Abstract

by Felicia Liu

The purpose of this study is to investigate different kinds of word embeddings to use for unsupervised part-of-speech tagging, which is tagging words with labels that denote their syntactical function given their context. Unsupervised models for NLP are needed to make use of the abundance of raw online textual data and to create systems that work with low-resource languages. This thesis will compare and contrast various neural network embedding models, in order to identify word embeddings that are well-suited for extracting relevant features on local word order, context, meaning, and morphology. Word-level embeddings have been trained with a context window of size 1 with different dimensions. The character-level embeddings have been extracted from a language model with a convolutional layer over characters. A Gaussian HMM is used as POS induction system. The findings were that models where a word is predicted from its context works better than embeddings where the context is predicted from a word. This means that CBOW and cwindow outperform the (structured) skipgram, and since cwindow is sensitive to word-order, it outperforms CBOW. On average the character-level embeddings performed around the level of GloVe and skipgram, which was good but not exceptional. In order to investigate the quality of the word embeddings trained for this project, we inspect the nearest neighbours and a 2D visualisation of the embedding space. This showed that word-level prediction-based embeddings have clusters of semantic similarity, whereas character-level embeddings have clusters of morphological similarity. Combining word- and character-level embeddings did very well in POS tagging. Further experiments included accounting for out-of-vocabulary words and performing tokenisation similar to the GHMM training data, where especially the latter showed significant improvement in POS tagging. On the basis of the results of this research it can be concluded that proper tokenisation and taking local word order into account can definitely improve embeddings for POS tagging. Furthermore, character-level embeddings are able to represent words in a completely different manner than regular embeddings that encode semantic information, which can be useful if combined with semantic features. It is worth researching other ways of creating character-level embeddings, either for part-of-speech tagging or for better word representation.

Acknowledgements

I would like to thank dr. Frank Keller for supervising my project and for having weekly meetings with me, which greatly helped me with the project. Furthermore, I would like to thank family and friends for continued support during the research and writing of this dissertation. Special thanks goes out to everyone who has helped me with proofreading this work.

Contents

Declaration of Authorship i

Abstract ii

Acknowledgements iii

Contents iv

List of Figures vi

List of Tables vii

1 Introduction 1
1.1 Structure of thesis . . . 3

2 Background 4
2.1 Part-of-speech tagging . . . 4
2.1.1 Supervised learning for POS tagging . . . 4
2.1.2 Unsupervised learning for POS tagging . . . 4
2.1.3 State of the art . . . 6
2.2 Word Embeddings . . . 7
2.2.1 Brief overview on the origin of word embeddings . . . 8
2.2.2 Neural network language models . . . 8
2.2.3 Neural network based word embeddings . . . 10
2.2.3.1 SENNA . . . 11
2.2.3.2 word2vec . . . 12
2.2.3.3 Structured word2vec (wang2vec) . . . 14
2.2.4 Corpus statistics-based word embeddings . . . 15
2.2.4.1 Global Vectors . . . 15
2.2.5 Character-level embeddings from neural language model . . . 16
2.2.6 Combined character-level and word-level embeddings . . . 19
2.3 Evaluation . . . 19
2.3.1 V-measure . . . 19

3 Methodology 21
3.1 Baseline model . . . 21

3.2 Datasets . . . 22
3.2.1 English Wikipedia 2006 dump . . . 22
3.2.2 Penn Treebank - Wall Street Journal . . . 24
3.3 The model pipeline . . . 24
3.3.1 Training word embeddings . . . 25

4 Experiments and Results **28**
4.1 Results on part-of-speech tagging . . . 28
4.1.1 HMM with 100 components . . . 28
4.1.2 HMM with 45 components . . . 31
4.1.3 A comparison between 45 and 100 components . . . 32
4.2 Quality of word embeddings . . . 34
4.2.1 Nearest neighbours of embeddings . . . 34
4.2.1.1 Semantically interesting words . . . 35
4.2.1.2 Syntactically interesting words . . . 35
4.2.1.3 Morphologically interesting words . . . 36
4.2.2 Visualising embeddings . . . 40
4.3 A solution for out-of-vocabulary words . . . 40
4.4 Leveraging more information by keeping capitalisation and punctuation . . . 41
4.5 Further experiments with word-level and character-level combined . . . 46
4.6 Test set performance . . . 47

5 Discussion and Conclusion **48**
5.1 Discussion . . . 48
5.1.1 Word-level embeddings . . . 48
5.1.2 Relation between embedding size and performance . . . 49
5.1.3 A comparison of 45 versus 100 components . . . 50
5.1.4 A look at character-level embeddings . . . 50
5.1.5 Qualitative differences between word- and character-level . . . 51
5.1.6 Analysis of further experiments . . . 51
5.2 Conclusion . . . 52
5.3 Future work . . . 54

Bibliography **56**

List of Figures

1 Illustration of Hidden Markov models . . . 7
2 Different types of (recurrent) neural networks . . . 9
3 A long short-term memory cell. . . . 10
4 Window approach of SENNA embeddings. . . . 11
5 Sentence approach of SENNA embeddings. . . . 13
6 CBOW and skipgram . . . 14
7 Cwindow and structured skipgram . . . 15
8 Structure of Kim et al. (2016)'s character-aware neural language model. . . . 17
9 Zipf's law visualised with Wikipedia data . . . 23
10 Cluster quality in V-measure for the different embeddings on POS tagging with 100 components. . . . 29
11 V-measure for different word embeddings and dimensions with HMM with 45 components, including character-level embeddings. . . . 31
12 Influence of perplexity of the language model on the (average) v-measure for character-level embeddings. . . . 33
13 Visualisation of CharCNN-Small embeddings projected down to 2D space with t-SNE. . . . 41
14 CharCNN-Small projected down to 2D space with visible cluster for words ending in -ed. . . . 42
15 CharCNN-Small projected down to 2D space with visible clusters for words ending in -ion(s) and -ship. . . . 42
16 CharCNN-Small projected down to 2D space with visible clusters for words with -ought and ending in -ish. . . . 43
17 Visualisation of cwindow-50 embeddings projected down to 2D space with t-SNE. 44
18 cwindow-50 projected down to 2D space with visible clusters for US locations and European locations. . . . 45
19 cwindow-50 projected down to 2D space with visible clusters for plays/literature and military. . . . 45
20 cwindow-50 projected down to 2D space showing word pairs such as boy and girl, and crew and staff. . . . 46

List of Tables

3.1 Overview of different LSTM-CharCNN models and their parameters. 27

4.1 V-measure for different word embeddings and dimensions with HMM with 100 components. 29
4.2 Different context window sizes for GloVe with dimension 20 and HMM with 100 components. 30
4.3 V-measure for CharCNN word embeddings with HMM with 100 components. . . 30
4.4 V-measure for different word embeddings and dimensions with HMM with 45 components. 31
4.5 V-measure for CharCNN word embeddings with HMM with 45 components, where the dimensionality of Large and Small has been reduced with PCA. 32
4.6 V-measure for CharCNN word embeddings with the two GHMM models, no dimensionality reduction. 32
4.7 A comparison of the average V-measure of all embeddings for different dimensions and GHMM settings. 33
4.8 A comparison of V-measure, homogeneity, and completeness for cwindow-50 for GHMM-45 and GHMM-100. 33
4.9 Nearest neighbours for semantically interesting words. 37
4.10 Nearest neighbours for syntactically interesting words. 38
4.11 Nearest neighbours for morphologically interesting words. 39
4.12 V-measures of skipgram and CBOW on Wiki-1 (completely processed), Wiki-2 (preserving punctuation), and Wiki-3 (punctuation and capitalisation) with 45 components. 43
4.13 V-measures for combinations of word-level and character-level embeddings on the GHMM with 45 components. 46
4.14 Test set performance as V-measures for the best of the embeddings on the GHMM with 45 components. 47

1 Part-of-speech tags used in the Penn Treebank dataset. 55

Chapter 1

Introduction

The purpose of this project is to experiment with different types of word embeddings in order to identify those that are particularly well-suited for part-of-speech (POS) tagging, which requires encoding syntactical features in the embeddings. Part-of-speech tagging is the task of assigning labels that denote the syntactical function of words in a given corpus based on each word's definition and context. Examples of such labels are NN (*noun*), JJ (*adjective*), and DT (*determinant).* For POS induction it is important to leverage both meaning and context for disambiguation, as there are many words where more than one label is possible. The word "like" demonstrates this; it can be used as a preposition, a conjunction, a noun, a verb, and an adverb. In this project, we are specifically interested in improving unsupervised POS tagging, as being able to leverage the excessive amount of unlabelled text available on the web and creating better unsupervised models are both interesting and worthwhile tasks. Part-of-speech tagging itself is a very helpful way of gaining more information about words in a corpus. The POS tags act as useful features to other NLP tasks such as syntactic parsing and named-entity recognition. Furthermore, these tags are used in finding the right stem of a word for information retrieval; helping summarisation tasks by identifying important nouns or verbs in a document; and determining the right pronunciation in speech synthesis and recognition tasks (Jurafsky and Martin, 2000).

A shortage of labelled data is especially a problem in languages other than English, where creating hand-labelled data can be very expensive. Besides that, they can pose other complications. An example of such a complication is if the language itself is morphologically rich, such as Turkish and Finnish. In Turkish, verbs have many different endings for all the different word forms. They can be recognised as verbs more easily if their morphology is inspected in POS tagging. If successful, unsupervised methods for POS tagging can be applied cross-domain and solve problems in other languages. Moreover, supervised POS tagging of English text has reached great performance already. State-of-the-art models can reach up to 97% accuracy on

English data (Manning, 2012). The last 3% is lost due to a variety of reasons, which Manning (2012) believes are not going to be solved by better machine learning methods or more advanced feature engineering.

Due to the recent increase in popularity of deep neural networks and their proven success in computer vision and speech recognition, deep learning has reached Natural Language Processing (NLP) as well. There are two ways in which deep learning has improved NLP: firstly, deep neural networks have been used to replace existing models for tasks such as document summarisation, language modelling, and machine translation; and, secondly, deep neural networks have been used to create word embeddings: vector representations of words which encode automatically extracted features through one or several layers of intermediate representation learning. Word embeddings are trained in an unsupervised manner and have been proven to outperform hand-engineered features (Collobert et al., 2011). They can be produced from simple one-layer networks to composite networks consisting of several sophisticated transformations, such as a convolutional layer as in Kim et al. (2016). We are interested in using these word embeddings as input to our POS tagging system in lieu of vocabulary indices or hand-engineered feature vectors. In order for word embeddings to improve POS tagging, they need to encode syntactical and morphological information.

This project will identify how syntax-informed word embeddings can be created in an unsupervised setting and used as input to the POS tagging task. We have some hypotheses on how we can improve encoding of syntactical information in word embeddings. Before discussing these hypotheses, we would like to emphasise the difference between prediction-based models, corpus-statistics models, and character-level embeddings. For the purpose of the introduction, each model will be explained briefly: prediction-based models usually create embeddings as part of an objective that models language in a limited context; statistics-based models use global occurrence counts when creating vector representations; and character-level models produce a word-level embedding by inspecting individual characters. These different models will be explained in more detail in Section 2.2. First of all, in the case of prediction-based models and statistics-based models, we believe that a smaller context window around the word of interest will work better than a larger context window. There is evidence that supports the hypothesis that smaller context windows result in better syntactical representations (Lin et al., 2015); we would like to reiterate this here to make sure that the embeddings we use take this into account. Secondly, the word embeddings created need to take word-order around the target word into account, whenever this is a possibility for the embedding of interest. For the POS induction system it is helpful to know what kind of word preceded the target word, and what kind of word followed the target word. The next example can illustrate this: being able to encode that the previous word is a determinant will increase the likelihood that the target word is an adjective or a noun. Finally, we would like to encode the morphology of target words, which can be done through encoding morphemes in a one-hot representation, or through character-level

embeddings. In the scope of this project we are not interested in the first type of models, as these would need to have access to a database of morphemes, which will make the system not completely unsupervised anymore. We are more interested in a model that learns character-level embeddings where it is able to pick up on important prefixes and suffixes automatically. Through recognising suffixes, for example, the word embedding can encode that both *giving* and *taking* have the suffix "-ing", and therefore could both be VBG (verb, gerund or present participle), whereas the words *eventful* and *successful* both end with "-ful", and therefore should both be tagged with JJ (adjective). In short, it is hypothesised that POS tagging can be improved by embeddings that encode information from local word order through small context windows or by encoding features on the morphology of words.

The main findings can be summarised as follows. Word embedding models such as CBOW and cwindow that use context to predict the target word do better in POS tagging than their counterparts skipgram and structured skipgram, which use the target word to predict the context. On top of this, cwindow, which has extra parameters to encode word order information of context words, outperforms CBOW, which treats each word in the context the same. Now turning to the experimental evidence on the usage of character-level embeddings the results do not indicate higher performance for these type of embeddings, even though they are very capable of encoding morphology and subword structure of target words. A clear divide between word-level and character-level embeddings has been found by finding nearest neighbours and by projecting the embeddings to two-dimensional space. Word-level embeddings show clusters for semantic similarity, whereas character-level embeddings show clusters for morphological similarity. Additional experiments show that representing low-frequency and/or out-of-vocabulary words with an averaged vector does not increase performance; tokenising the embedding training data to be similar to the POS training data results in significant improvements; and combining word-level and character-level embedding lead to good input features for POS tagging: they improve a lot upon the scores for using the embeddings individually.

1.1 Structure of thesis

In Chapter 2, we set out to discuss the related work that has led to the development of the research objective of this thesis. It will include an overview of part-of-speech tagging and relevant word embeddings. After discussing the background a discussion of the baseline model will follow in Chapter 3 as part of the methodology. This will also contain an overview of the data sets used, and the model pipeline. An overview of results and their interpretation will follow in Chapter 4. Finally, the thesis will conclude with a discussion of the results and suggestions for further research in Chapter 5.

Chapter 2

Background

2.1 Part-of-speech tagging

This section will briefly discuss supervised POS tagging, followed by a more lengthy discussion of relevant unsupervised models.

2.1.1 Supervised learning for POS tagging

Jurafsky and Martin (2000) introduce a couple of traditional ways of doing POS tagging, amongst others the Hidden Markov model (HMM) and maximum entropy Markov model (MEMM). State-of-the-art machine learning models for supervised part-of-speech tagging are maximum entropy classifiers, SVMs, and guided learning (Jurafsky and Martin, 2000). All three of these classifiers process the input text in a bidirectional fashion and result in accuracies between 97.16 and 97.33%. With a gold standard, supervised learning is almost a solved problem in English. However, for low-resource languages where it is hard to obtain a gold standard, unsupervised tagging is necessary.

2.1.2 Unsupervised learning for POS tagging

Many different models have been designed to perform unsupervised POS induction. The oldest model is Brown clustering by Brown et al. (1992). In their work they extend n-gram language models to class-based n-gram models. A language model defines the probability of a string of words w_1^k. This probability can be expressed as the multiplication of the following terms

$$P(w_1^k) = P(w_1)P(w_2|w_1)\cdots P(w_k|w_1^{k-1}), \tag{2.1}$$

where w_1^{k-1} is the string $w_1 w_2 \cdots w_{k-1}$. Normally we want to predict w_k given its history w_1^{k-1}, however, a Markov assumption is usually made to restrict the history to the previous n words. This is what defines an n-gram language model. Furthermore, because of existing semantic and syntactic similarity between words, Brown et al. (1992) state that it is possible to group different words together in classes based on their similarity. This forms the basis of what underlies Brown clustering. With this information, we define an n-gram class model as a language model which is based on n-grams and word classes, so that the following holds

$$P(w_k|w_1^{k-1}) = P(w_k|c_k)P(c_k|c_1^{k-1}). \tag{2.2}$$

In order to cluster words, the probability of the data under this model needs to be optimised. Instead of using generic classes for clustering, we use part-of-speech tags as an alternative. By making this change the model can be readily extended to POS induction. This system is trained by greedy agglomerative hierarchical clustering, where afterwards individual words are potentially moved to a different class if it increases the probability of the corpus (Brown et al., 1992).

Between this initial model from 1992 to Lin et al. (2015), many other models for POS induction have been designed. Christodoulopoulos et al. (2010) performed a research survey where they compare seven different unsupervised POS induction systems using the same data and evaluation measures. One of the models is the aforementioned Brown clustering model (Brown et al., 1992). This model has been extended by Clark (2003) through adding morphological information, implemented by a single-order letter Hidden Markov model. It also uses a different search procedure that raises the probability that the model clusters words together with similar morphological features. Another clustering approach induces the number of clusters during training (Biemann, 2006). It employs a graph clustering algorithm named *Chinese Whispers*. The system makes two partitionings of the word graphs from the contextual similarity of high frequency words (target words) and log-likelihood statistics of non-target words (Biemann, 2006). The last four models all use variations on (bigram) HMMs. Griffiths and Goldwater (2007) use a Bayesian approach by placing Dirichlet priors on the multinomial distribution used to define state-state and state-emission probabilities. A collapsed Gibbs sampler is used to deduce the tags. Johnson (2007) has investigated why HMMs trained with Expectation-Maximisation (EM) result in bad models. Since EM assigns approximately the same number of word tokens to each hidden state, the performance is bad because the actual distribution of tokens to part-of-speech tags is skewed. Johnson (2007) therefore uses a Bayesian approach as well, but with Variational Bayes (VB) rather than Gibbs sampling. Another way of encouraging sparsity is through directly constraining the posterior probability distributions (Graça et al., 2009). They use the same bigram HMM, but apply the posterior regularisation framework for this goal. Lastly, Berg-kirkpatrick et al. (2010) created a system that is structurally similar to an HMM, with the logistic function instead of multinomial state-state and state-emission distributions.

Because of this, Berg-kirkpatrick et al. (2010) are able to use local features, such as character trigrams and capitalisation, and to modify the M-step of EM with a gradient-based search algorithm. Christodoulopoulos et al. (2010) find that Brown clustering, Clark (2003), and Biemann (2006) perform very well compared to the other four models, running with a fraction of the time needed for the newer HMM models. The two best performing models are Clark (2003) and Berg-kirkpatrick et al. (2010), which also happen to be the only two models to use morphological information. Moreover, since Brown et al. (1992) and Clark (2003) are very similar, apart from the added morphological features, Christodoulopoulos et al. (2010) conclude that the morphological features are what makes the difference between the two models.

2.1.3 State of the art

We now move on to what we regard as the state-of-the-art model, which will function as a baseline model to the experiments in this project. The baseline is provided by Lin et al. (2015), who use a classic first-order Hidden Markov model (HMM) and a conditional random field autoencoder (CRF) for the POS tagging task, where the input to the models is replaced with word embeddings. As the HMM functions as the baseline, we will also go into a bit more depth on what an Hidden Markov model is and how it works.

A Hidden Markov model is defined as the following

$$p(\boldsymbol{w}, \boldsymbol{t}) = \prod_{i=1}^{\ell} p(t_i|t_{i-1}) \times p(w_i|t_i), \tag{2.3}$$

where $p(t_i|t_{i-1})$ represents the transition probability and $p(w_i|t_i)$ represents the emission probability. The transition probability is also known as the state-state probability. The emission probability is the probability of generating word w_i given tag t_i. Generally, an HMM's state emission probability $p(w_i|t_i)$ is defined with a multinomial distribution over the possible output tags, although a logistic function can be used as well (Berg-kirkpatrick et al., 2010). One of the models compared in Lin et al. (2015) is the standard multinomial HMM. Instead of the multinomial distribution, the other HMM in Lin et al. (2015) is used with a Gaussian distribution. Since Lin et al. (2015) replace the input of the models with word embeddings, the Gaussian HMM (GHMM) defines the probability $p(w_i|t_i)$ differently. Instead of generating a word type, a word embedding $\boldsymbol{v}_w$ is generated from a Gaussian distribution with mean $\boldsymbol{\mu}_t$ and covariance matrix $\boldsymbol{\Sigma}_t$ for tag t. $\boldsymbol{v}_w|t$ is defined as the Gaussian distribution as follows

$$p(\boldsymbol{v}_w; \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) = \frac{\exp(-\frac{1}{2}(\boldsymbol{v}_w - \boldsymbol{\mu}_t)^T \boldsymbol{\Sigma}_t^{-1}(\boldsymbol{v}_w - \boldsymbol{\mu}_t))}{\sqrt{(2\pi)^d |\boldsymbol{\Sigma}_t|}}. \tag{2.4}$$

It is based on the intuition that words from a specific tag t are generated from the cluster with centre $\boldsymbol{\mu}_t$ and covariance matrix $\boldsymbol{\Sigma}_t$. Figure 1 shows two examples of HMMs.

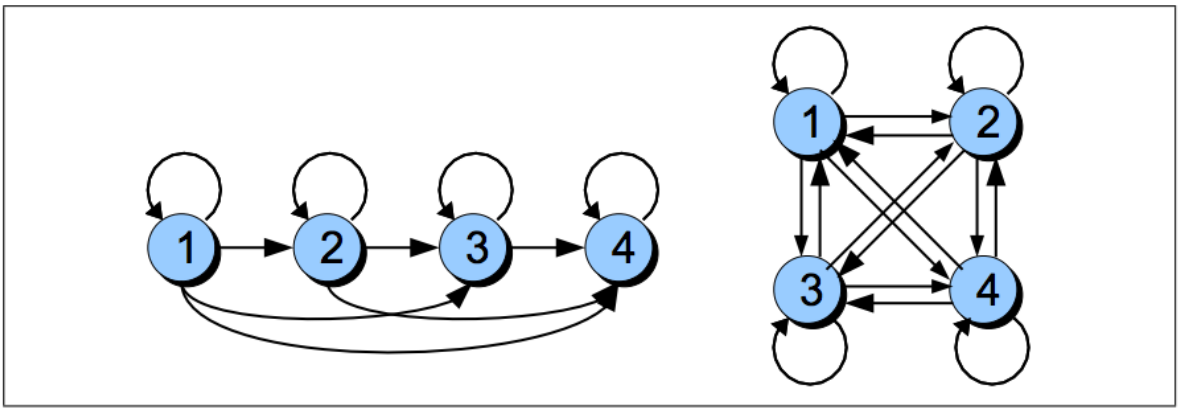

FIGURE 1: Two examples of HMMs: a Bakis HMM on the left side and a fully connected HMM on the right side (Jurafsky and Martin, 2000)

In the CRF autoencoder model, the input $\hat{\boldsymbol{w}}$ is reconstructed through latent variables, on the condition that a copy of the input $\boldsymbol{w}$ is given (Ammar et al., 2014). Therefore, $p(\boldsymbol{t}|\boldsymbol{w})$ is the encoding model and $p(\hat{\boldsymbol{w}}|\boldsymbol{t})$ is the reconstruction model. With feature vector $\boldsymbol{\lambda}$ and local feature function $\mathbf{f}$ the linear-chain CRF is defined as follows

$$p(\hat{\boldsymbol{w}}, \boldsymbol{t}|\boldsymbol{w}) = p(\boldsymbol{t}|\boldsymbol{w}) \times p(\hat{\boldsymbol{w}}|\boldsymbol{t}) \tag{2.5}$$

$$\propto p(\hat{\boldsymbol{w}}|\boldsymbol{t}) \times \exp \boldsymbol{\lambda} \cdot \sum_{i=1}^{|\boldsymbol{w}|} \mathbf{f}(t_i, t_{i-1}, \boldsymbol{w}) \tag{2.6}$$

For both methods the Baum-Welch algorithm is used to estimate $\boldsymbol{\mu}_{t^*}$ and $\boldsymbol{\Sigma}_{t^*}$ (Baum et al., 1970).

As input to both models Lin et al. (2015) the skipgram and structured skipgram embeddings are used (Ling et al., 2015a; Mikolov et al., 2013b). Through experimentation, they find that small context windows produces better embeddings for POS induction. The reason for this may be that small context windows produce better representations that take into account close words and their word order compared to larger context windows.

2.2 Word Embeddings

The traditional approach to solving NLP tasks has been to extract a rich set of hand-engineered features and use it with a standard classification algorithm (Jurafsky and Martin, 2000). Creating and selecting these features is an empirical process, where most features are made based on linguistic ideas, selected through a trial and error process, and completely dependent on which specific NLP task at hand is solved. Instead of hand-designing features, there are word embeddings trained in unsupervised fashion which can greatly improve performance on NLP

tasks (Al-Rfou et al., 2013; Collobert et al., 2011; Pennington et al., 2014). This section will give an overview of work on word embeddings, outline interesting approaches, and highlight the manner in which each of the embeddings can produce relevant representations for POS tagging.

2.2.1 Brief overview on the origin of word embeddings

Using continuous vectors to represent words is not a recently discovered method for NLP. Work in this field started around the 1980s and more recently has focused on neural network language models that learn word representations jointly or through internal projection (Mikolov et al., 2013b). The simplest representation of a word are co-occurrence vectors, where a word w is represented by a vector where each entry denotes the association between a word w and a context word c. This association can be pure co-occurrence counts, or they can be created through (positive) pointwise mutual information (PPMI) (Levy et al., 2015). When such a vector represents a document, one may use the bag-of-words model. This is a vector with length equal to the vocabulary size containing the number of times each word occurs in the document. A transformation that can be applied to this type of representation can be term-frequency inverse document-frequency (tf-idf).

Usually, these type of representations can produce very large and sparse matrices. To reduce dimensionality in order to be able to generalise better and for computational efficiency, singular value decomposition (SVD) may be applied to these matrices (Levy et al., 2015). Because of the recent increase in popularity of deep learning models, this project will mainly focus on distributed representations learnt through neural networks as opposed to through Latent Semantic Analysis (LSA), as the latter does not match the performance of neural network-based embeddings.

2.2.2 Neural network language models

A language model is defined as a probability distribution over a sequence of words. As discussed in previous sections, it is traditionally modelled as an n-gram model by making a Markov assumption. These count-based models are easy to train but they suffer from problems associated with data sparsity Kim et al. (2016). Neural network-based language models started out as feed-forward networks, where the history of the previous N words is presented to the network with 1-of-V encoding (Bengio et al., 2003). The input is projected to a projection layer, followed by hidden layers before predicting the probability distribution over all V words in the vocabulary. The projection layer is a matrix that transforms the vocabulary index to a word embedding. These neural network-based models represent words as vectors, which makes it possible to use them as input to a neural network. The parameters of this model are learned through training

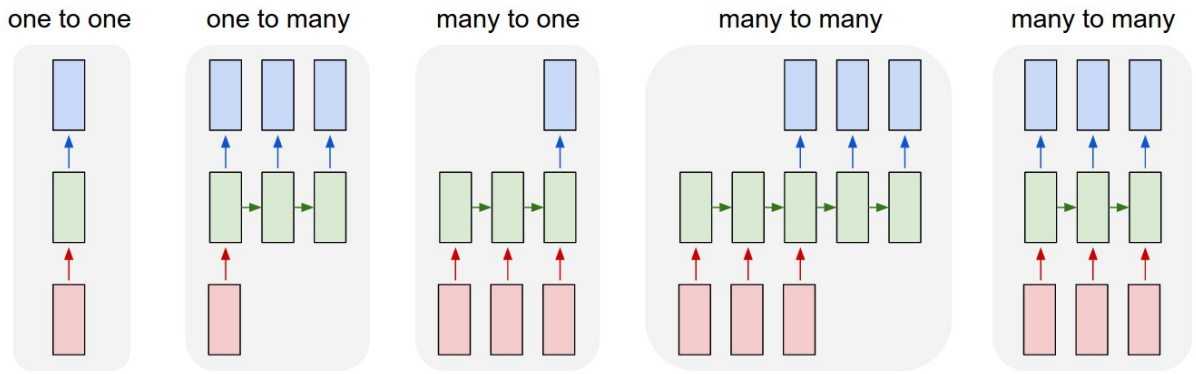

FIGURE 2: Different types of (recurrent) neural networks. From left to right: (1) A general feedforward network. (2) Sequence output (example: image captioning task). (3) Sequence input (example: sentiment analysis). (4) Sequence input and sequence output (example: RNN encoder-decoder for machine translation). (5) Synced sequence input and output (example: language modelling). Image taken from Andrej Karpathy's blog[1].

on a language modelling corpus by minimising the perplexity of the resulting output. The embeddings produced by these type of models show that words with similar semantics are close in vector space (Mikolov et al., 2013a).

One limitation of the feed-forward network is the fixed size of the input. Recurrent neural networks (RNN) were proposed to overcome this limitation by accepting a stream of input, as well as producing a stream of output. Moreover, an RNN connects the hidden layer of the previous time step to the current time step's hidden layer allowing it to remember information about its history. See Figure 2 for a graphical depiction of the different possible structures of an RNN, where example 5 shows a language modelling architecture. Whereas in feedforward networks during the propagation of input to a hidden layer the input is multiplied by a matrix, after which a nonlinear transformation may be applied, in an RNN the hidden layer gets both the input at current time $\boldsymbol{x}_t \in \mathbb{R}^n$ as well as the hidden layer from the previous time step $\boldsymbol{h}_t \in \mathbb{R}^m$. The following formula shows how the hidden state vector is computed

$$\boldsymbol{h}_t = f(\boldsymbol{W}\boldsymbol{x}_t + \boldsymbol{U}\boldsymbol{h}_{t-1} + \boldsymbol{b}), \tag{2.7}$$

where $\boldsymbol{W} \in \mathbb{R}^{m \times n}$, $\boldsymbol{U} \in \mathbb{R}^{m \times m}$, $\boldsymbol{b} \in \mathbb{R}^m$ are the parameters of the affine transformation in an RNN, and f is an element-wise non-linearity.

Regular RNNs, also known as vanilla RNNs, suffer from vanishing and exploding gradients, and can have a hard time remembering previous information (Pascanu et al., 2012). Gradients vanish when they become close to zero, and explode when they become very large. Because of these issues, it can be hard to properly train an RNN. This led to the invention of Long Short-Term Memory cells, which solve these problems and are able to remember long distance histories (Hochreiter and Schmidhuber, 1997). They extend regular RNNs through the addition

[1] http://karpathy.github.io/2015/05/21/rnn-effectiveness/

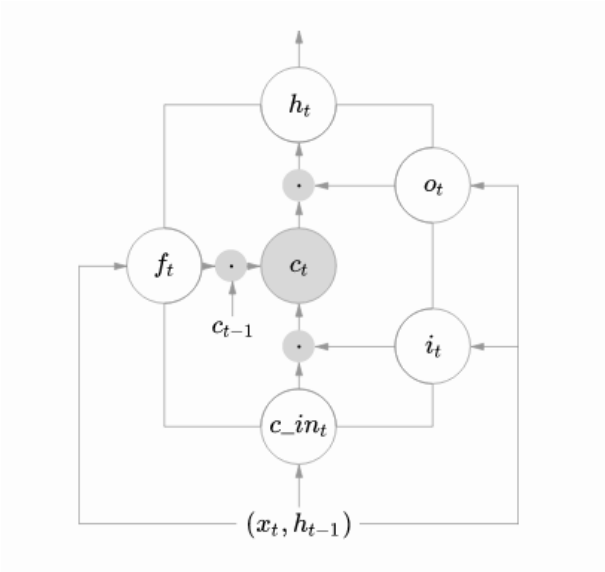

FIGURE 3: Visualisation of a long short-term memory cell with input, forget, and output gates. Image taken from Adam Paszke's blog[2].

of a memory cell vector $\boldsymbol{c}_t \in \mathbb{R}^n$ at each time step. Therefore, in addition to the input $\boldsymbol{x}$ at time t and the previous hidden state $\boldsymbol{h}_{t-1}$, it also takes in the previous cell state in the following way

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}^i \boldsymbol{x}_t + \boldsymbol{U}^i \boldsymbol{h}_{t-1} + \boldsymbol{b}^i) \tag{2.8}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}^f \boldsymbol{x}_t + \boldsymbol{U}^f \boldsymbol{h}_{t-1} + \boldsymbol{b}^f) \tag{2.9}$$

$$\boldsymbol{o}_t = \boldsymbol{W}^o \boldsymbol{x}_t + \boldsymbol{U}^o \boldsymbol{h}_{t-1} + \boldsymbol{b}^o) \tag{2.10}$$

$$\boldsymbol{g}_t = \tanh(\boldsymbol{W}^g \boldsymbol{x}_t + \boldsymbol{U}^g \boldsymbol{h}_{t-1} + \boldsymbol{b}^g) \tag{2.11}$$

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \boldsymbol{g}_t \tag{2.12}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t), \tag{2.13}$$

where $\boldsymbol{f}_t$ is the forget gate, $\boldsymbol{i}_t$ is the input gate, and $\boldsymbol{o}_t$ is the output gate. The parameters of the LSTM are $\boldsymbol{W}^j, \boldsymbol{U}^j, \boldsymbol{b}^j$ for each of the input, forget, output, and "intermediate" gate $\boldsymbol{g}_t$. Because memory cells are added to each other, they minimise the vanishing gradient problem. However, they do not help with exploding gradients, for which clipping the gradient is a reasonable solution. LSTMs have been shown to perform much better than regular RNNs (Hochreiter and Schmidhuber, 1997). Both networks can be extended to multiple layers, which is often needed for good performance.

2.2.3 Neural network based word embeddings

The following section will describe a few word embeddings from the group of prediction-based models.

[2] http://apaszke.github.io/lstm-explained.html

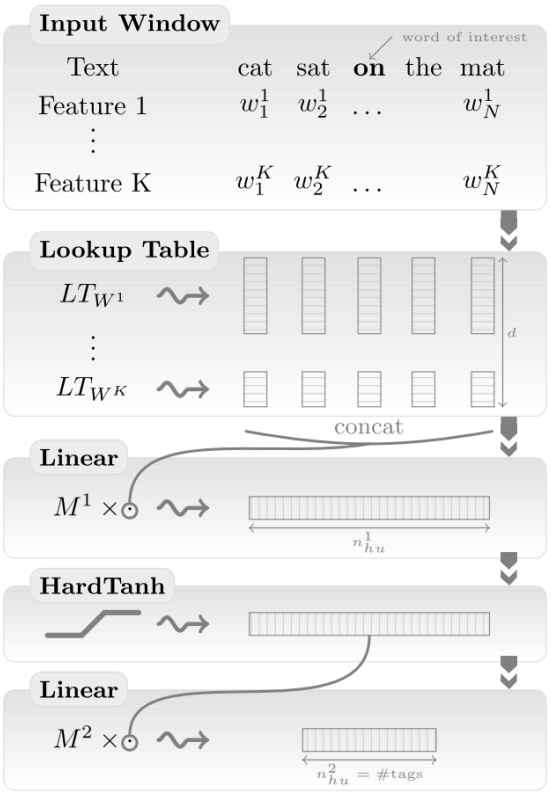

FIGURE 4: Transforming words into feature vectors. Words from a limited dictionary are fed as indices to the network. Each of the words is mapped to a feature vector through a look-up table. This table is essentially a matrix which applies the same transformation to each layer. The result is fed to a regular feedforward network. In the figure this is depicted as a linear layer, a hard tanh transformation, and another linear layer. Everything is trained with backpropagation from an initial random initialisation. Image taken from Collobert et al. (2011).

2.2.3.1 SENNA

Collobert et al. (2011) criticise developing hand-engineered features to optimise performance of a model for a specific NLP benchmark. Instead, they propose using a unified learning system for finding internal representation of features and optimise this on *multiple* benchmarks. Because they want to avoid task-specific features, the learned representations are more general than the tasks they need to perform well on. Collobert et al. (2011) benchmark their system on part-of-speech tagging (POS), chunking (CHUNK), Named-Entity Recognition (NER), and Semantic Role Labelling (SRL). They call their system SENNA for Semantic/Syntactic Extraction using a Neural Network Architecture. As can be seen in Figure 4, each word of interest is represented by an index from a finite dictionary. This index is mapped to a feature vector through a lookup table, which are randomly initialised, but can be replaced with pre-trained representations. It is then passed on to further layers of the neural network.

In the window approach, a fixed size window of k words around the word of interest is used to produce a matrix of size $D \times k$. This matrix can be unrolled to a Dk-dimensional vector and

fed as input through a combination of linear and hardtanh layers. The hardtanh is defined as follows

$$\text{HardTanh}(x) = \begin{cases} -1 & \text{if } x < -1 \\ x & \text{if } -1 \leq x \leq 1 \\ 1 & \text{if } x > 1 \end{cases}. \tag{2.14}$$

This process is depicted in Figure 4. Features are extracted on two levels: the first layer extracts features from the word and the second layer extracts them from a window around the word of interest. This model treats the window of words as a sequence which has both local and global structure. This approach works well for most of the NLP tasks, except for semantic role labelling.

In the sentence approach the problem with needing to look at the whole sentence is solved by using a convolutional approach, also known as Time Delay Neural Networks (TDNNs) (Collobert et al., 2011). The convolutional layer takes in a complete sentence produced through a lookup table with local features around each word. During the convolutional operation a window slides over the zero-padded input and calculates the Frobenius inner product as output. The max-layer takes the result and performs a max operation over time in order to produce a fixed-size global feature vector. The produced feature vector is then fed to other layers producing affine transformations. A more mathematical explanation of TDNNs can be found in Section 2.2.5.

Both networks are trained using stochastic gradient descent to maximise the likelihood over training data. As a criterion, the cross-entropy loss is used. The resulting word embeddings do well on the NLP tasks, but not when looking for nearest neighbours in the vector space by calculating cosine similarity. To improve these embeddings, Collobert et al. (2011) leverage two large data sets (English Wikipedia and the Reuters RCCV1) and use the window approach to train a language model. Rather than using perplexity, the language model is trained in a pairwise ranking approach. Collobert et al. (2011) train a language model with window size 11 and 100 hidden units. This language model is then trained on the complete Wikipedia and Reuters corpus, with a total training time of nearly 2 months in 2011. These embeddings show much better nearest neighbours than the ones just trained to perform well on the NLP benchmark tasks. The SENNA embeddings are chosen for the POS tagging task because it takes word order of the context window into account when training the embeddings. They are also used to supervised tasks where syntactical information is needed, such as chunking.

2.2.3.2 word2vec

Perhaps the most well-known and widely-used embeddings are from the word2vec model of Mikolov et al. (2013b), which has shown to do very well on semantic and word analogy tasks,

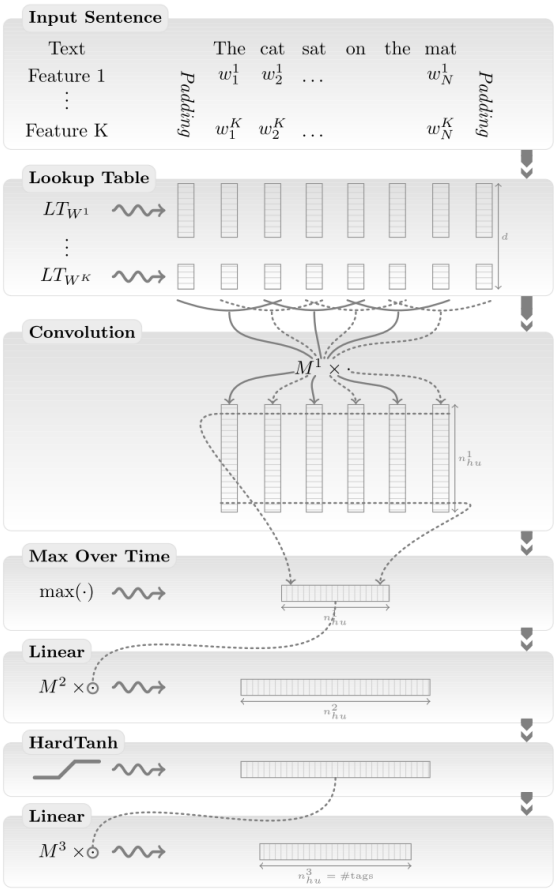

FIGURE 5: Transforming sentences into global feature vectors. A matrix is produced through a lookup table which represents a sentence. The sentence is fed through a time delay neural network layer and max-pooling layer to produce a fixed-size global feature vector. This vector is then passed through further affine network layers. Image taken from Collobert et al. (2011).

are fast to train, and are thus widely used.

Continuous-bag-of-words Figure 6 shows the *continuous bag-of-words* (CBOW) and *continuous skipgram* models. The CBOW model is based on Bengio et al. (2003)'s feedforward NNLM, although without the non-linear hidden layer. It takes as input a context window around the word of interest and averages them to get the projection layer. The projection layer is then used to predict the word of interest. Since the context vectors are averaged and order does not matter, this model has been named bag-of-words, analogous to the bag-of-words vectors mentioned in the beginning of this section that provide information about terms occuring together, but not about their order.

Continuous skipgram The skipgram model is the "inverse" of the CBOW model. In this model, the word of interest acts as the input to predict words within a certain window size of the input word. Since words farther away should play less of a role in being predicted by the middle word, these words are given a lesser weight when sampled.

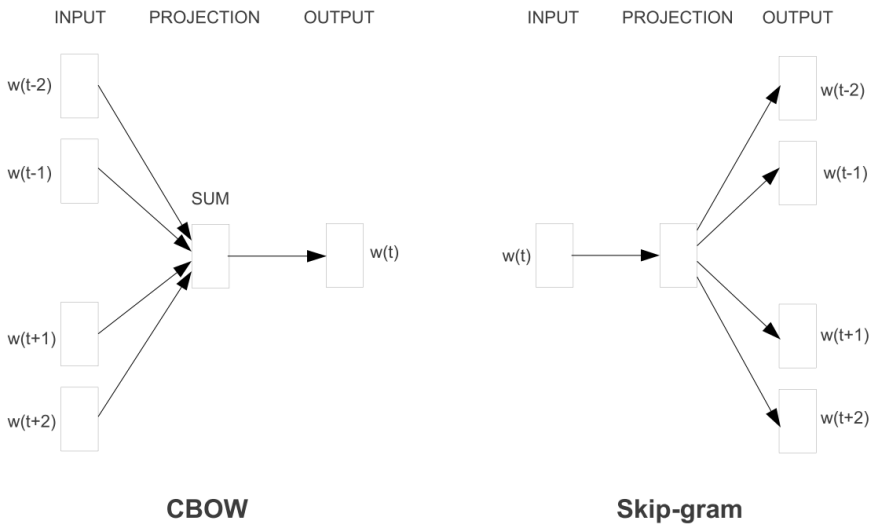

FIGURE 6: The structure of the CBOW and skipgram models are shown here. The CBOW method uses context words to predict the target word by projecting it to a lower dimensional space. The skipgram model is the "inverse" of the CBOW model. It predicts the context words using the middle word. Image taken from Mikolov et al. (2013b).

Mikolov et al. (2013b) have found that their word2vec model performs well on more complex similarity tasks and finding semantic relations. In Mikolov et al. (2013a) several extensions of the skipgram model have been presented. These include subsampling of frequent words, using of noise contrastive estimation (NCE) (Mnih and Kavukcuoglu, 2013), inclusion of phrase-based modelling, and negative sampling as alternative to the hierarchical softmax. The skipgram model is part of the baseline experiments and since Lin et al. (2015) do not report performance of CBOW in the POS tagging task, we include it here for completeness.

2.2.3.3 Structured word2vec (wang2vec)

The word2vec models CBOW and skipgram both ignore word order information. Ling et al. (2015a) argue that because of this reason, these models are less suitable to tasks involving syntax. Syntax gives us information about what words go in which order in a sentence; if the order of the sentence is changed, the words will also have changed syntactical roles. Evidence has been found that ignoring word order has been proven to result in less than optimal syntactic representations (Andreas and Klein, 2014). In that paper, researchers have compared word2vec, which can be trained relatively quickly, to the embeddings by Collobert et al. (2011), which are computationally more expensive. The latter incorporates word order information, and produced far better results than the word2vec model.

Ling et al. (2015a) aim to model word order as well by introducing the *structured skipgram* and *continuous window* (cwindow) models. Whereas in the skipgram model only one output matrix

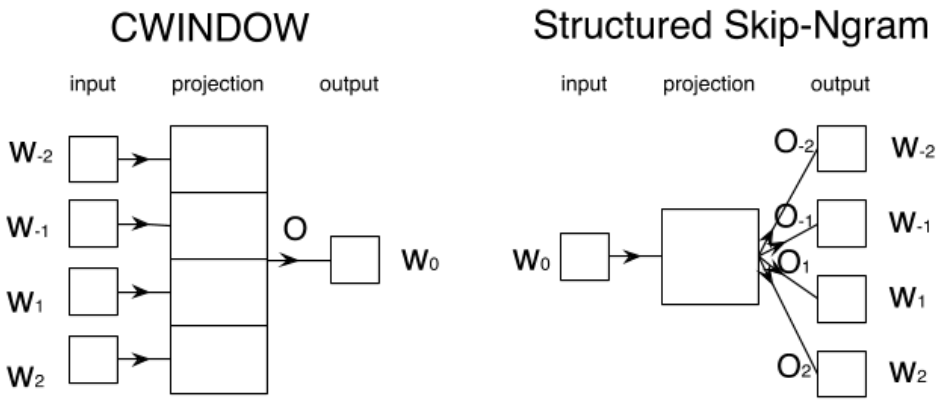

FIGURE 7: The structure of the cwindow and structured skipgram models are shown here. The models are adaptations of word2vec's CBOW and skipgram. Instead of one output matrix O there is a different one for each of the words within the context window. Image taken from Ling et al. (2015a).

O is used to predict the context word of a word of interest, the structured skipgram model is adapted to incorporate word positioning. Instead of only one output matrix O, each word in the context window of c on each side will have their own output matrix. There will thus be $c \times 2$ output matrices O, compared to only one output matrix for word2vec. Ling et al. (2015a) make a similar adaptation to CBOW. For predicting each context word given the middle word there will be a specific, different output matrix, resulting in an increase of parameters by $c \times 2$, similar to the structured skipgram. This model is called cwindow, and is similar to the embeddings by Collobert et al. (2011), except that there is no projection of the vector of word embeddings to a window embedding. There are also $c \times 2$ more parameters in this model, in order to account for word position information. Both models are shown in Figure 7. Ling et al. (2015a) test their embeddings against word2vec and SENNA on part-of-speech tagging and dependency parsing and show improved performance on both tasks.

2.2.4 Corpus statistics-based word embeddings

2.2.4.1 Global Vectors

There are two main branches for learning vector representations of words: global matrix factorisation methods and local context window methods (Pennington et al., 2014, p. 1532). Latent semantic analysis (LSA) is an example of the former. It efficiently uses statistical information about the corpus it is trained on, but it does poorly on Mikolov et al. (2013b)'s word analogy task. On the other hand, models such as word2vec do well on this task, but they do not use corpus statistics very well when creating word vectors; they do not depend on global co-occurrence counts at all. Both models have their advantages and disadvantages.

Pennington et al. (2014) set out to create a new type of word embedding that leverages the advantages of count-based methods and prediction-based methods to create GloVe, which stands for Global Vectors. It is a log-bilinear regression model that produces word representations in an unsupervised manner. The intuition behind GloVe is that instead of pure probabilities, ratios of co-occurrence should be used as a starting point (Pennington et al., 2014, p. 1535). To preserve vector linearity, the training objective of GloVe vectors is to make sure the dot product of two word vectors is equal to the logarithm of the probability of co-occurrence between the words. With co-occurrences, one naturally runs into sparsity issues, which a weighting function in the form of a least squares regressor will help combat. As for the performance of the GloVe model, Pennington et al. (2014) find that GloVe on overall beats regular SVD techniques and word2vec on word analogy, word similarity, and NER tasks. Specifically for syntactic subtasks, they found that GloVe does well with small and asymmetric context windows, again following the intuition that syntactic information is encoded in the immediate context of a word, rather than in large context windows.

2.2.5 Character-level embeddings from neural language model

Normally, neural language models (NLMs) work on the word-level and ignore any subword information that may be helpful in predicting a low perplexity next word. For example, if the history is "have been ...", words with suffix "-ing" make a good prediction. Another problem of standard NLMs is that embeddings for rare words are not estimated very well. A character-level embedding can always be created from a rare word, because it is not word-level based. Especially in morphologically rich languages such as Turkish and Arabic, there are many word forms and therefore many more different tokens, but there exists structural similarity between words with the same function.

To combat this problem Kim et al. (2016) propose a language model that uses subword information for better prediction in a language model. To leverage character information they use a character-level convolutional neural network (CNN) to extract features from the input word. The output of the CNN is reduced to a fixed-size embedding by way of a max-pooling layer, which is then fed to a RNN language model with LSTM cells.

The next section goes into a bit more depth regarding the network structure.

Character-level CNN In order to go from input word to output embedding, a representation of the input word needs to be constructed. The model will collect the vocabulary when looping through the data set, constructing a matrix of character embeddings $\boldsymbol{Q} \in \mathbb{R}^{d \times |\mathcal{C}|}$, where $\mathcal{C}$ is the vocabulary of characters and d is the character embedding dimensionality. The character-level representation for a word k consisting of the l characters $[c_1, \ldots, c_l]$

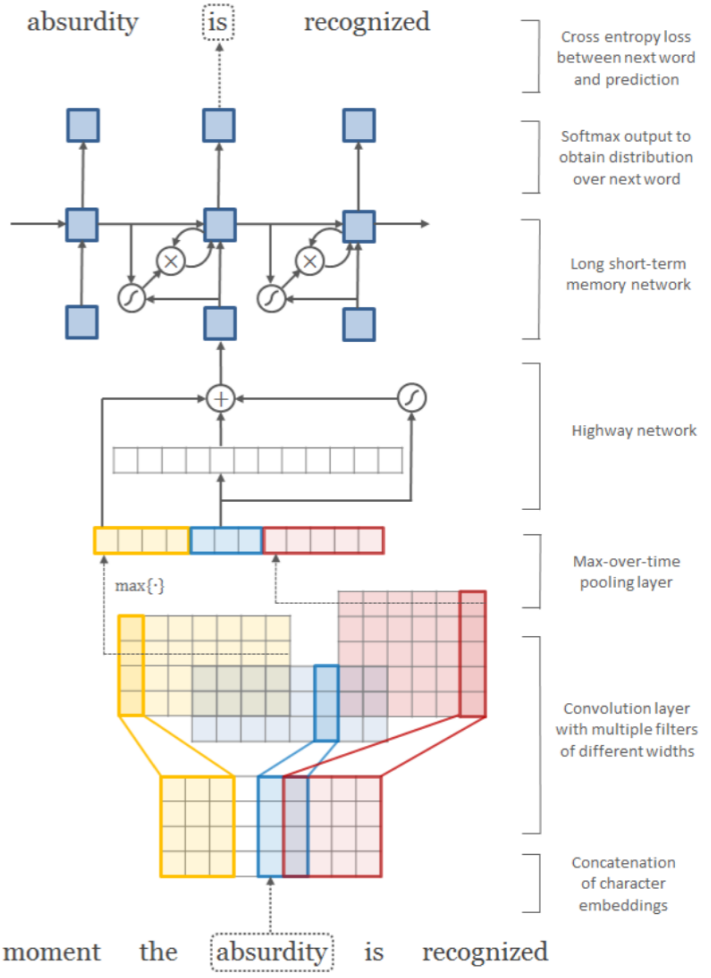

FIGURE 8: The structure of Kim et al. (2016)'s character-aware neural language model. The word "absurdity" is taken as the input. Using a look-up table it is transformed to a matrix C_k. A convolutional layer is used to obtain different feature maps. After this, a max-over-time pooling layer is used to get a fixed-dimensional representation of the word, which is then passed to the highway network. The output is further propagated to an LSTM language model network, which produces the output using softmax. Error is calculated using cross-entropy loss. Image taken from Kim et al. (2016).

is the concatenation of column vectors for each of the c_j characters where $j = 1, \ldots, l$, obtaining a character-level representation of k, namely $\boldsymbol{C}^k \in \mathbb{R}^{d \times l}$. After constructing the representation of the input, for technicalities it is padded with zeros in order for all representations to have the same size in the mini-batch. It is then fed to the convolutional layer. A narrow convolution between the matrix $\boldsymbol{C}^k$ and a kernel $\boldsymbol{H} \in \mathbb{R}^{d \times w}$ of width w is applied, a bias is added, and a nonlinearity is used to get a feature map $\boldsymbol{f}^k \in \mathbb{R}^{l-w+1}$. As stated in Kim et al. (2016), the following formula explains how the i-th element of a feature map is produced:

$$\boldsymbol{f}^k[i] = \tanh(\langle \boldsymbol{C}^k[*, i : i + w - 1], \boldsymbol{H} \rangle + b). \tag{2.15}$$

The Frobenius inner product is taken between the i to $(i + w - 1)$-th column and the kernel $\boldsymbol{H}$. In order to produce a fixed-size embedding, the *max-over-time* is taken:

$$y^k = \max_i \boldsymbol{f}^k[i]. \tag{2.16}$$

As stated in (Kim et al., 2016), this will take the most important feature for each kernel, assuming that a kernel is scanning for character n-grams.

Highway network Kim et al. (2016) show improvements in performance by running the fixed-size embedding $\boldsymbol{y}^k$ through a *highway network*. A highway network has a transform gate $\boldsymbol{t}$ and carry gate $(1 - \boldsymbol{t})$ which perform the following computations

$$\boldsymbol{z} = \boldsymbol{t} \odot g(\boldsymbol{W}_H \boldsymbol{y} + \boldsymbol{b}_H) + (1 - \boldsymbol{t}) \odot \boldsymbol{y}), \tag{2.17}$$

where $\boldsymbol{z}$ is the output of the highway layer, $\boldsymbol{W}_H, \boldsymbol{b}_H$ are the weights and bias of the network, and g is a non-linearity function. A highway network allows deep neural networks to *carry* some of the values of the input vector $\boldsymbol{y}$ directly to the output. Kim et al. (2016) find that a highway network improves the NLM's performance over just directly inputting the fixed-size embedding, and compared to no transformation of $\boldsymbol{y}$, a regular multilayer perceptron (MLP) actually worsens performance.

RNN language model The probability distribution over the next word w_{t+1} given the history $w_{1:t} = [w_1, \ldots, w_t]$ with vocabulary $\mathcal{V}$ is given by the RNN language model (RNNLM) by calculating the softmax over the affine transformation to the hidden layer $\boldsymbol{h}_t$ as follows

$$P(w_{t+1} = j | w_{1:t}) = \frac{\exp(\boldsymbol{h}_t \cdot \boldsymbol{p}^j + q^j)}{\sum_{j' \in \mathcal{V}} \exp(\boldsymbol{h}_t \cdot \boldsymbol{p}' + q^{j'})}. \tag{2.18}$$

Here $\boldsymbol{p}^j$, the output embedding, is the j-th column of the matrix $\boldsymbol{P} \in \mathbb{R}^{m \times |\mathcal{V}|}$ and q^j is the bias weight. Generally, an RNNLM gets as input the embedding $\boldsymbol{x}^k$, but in this model it is replaced by the output of the character-level CNN.

The following will describe the training settings used by Kim et al. (2016). The model is trained with truncated backpropagation while minimising the negative log-likelihood (NLL)

$$NLL = -\sum_{t=1}^{T} \log P(w_t | w_{1:t-1}). \tag{2.19}$$

The output of the LM is evaluated with perplexity. The optimisation algorithm used is stochastic gradient descent with a learning rate which halves each time the perplexity does not change. Furthermore, as regularisation dropout is applied to the LSTM layers and softmax layer. Instead of calculating the softmax, the hierarchical softmax is calculated. The latter is used when the

number of output classes is very large. The complete network structure can be found in Figure 8.

2.2.6 Combined character-level and word-level embeddings

Santos and Zadrozny (2014) propose a deep neural network architecture that combines character-level with word-level embeddings in an end-to-end system that directly performs POS tagging. The network is fed a sentence, and its output is for each word in the sentence a score for a specific tag $t \in T$. The Viterbi algorithm is used to do structured prediction from the output. In Santos and Zadrozny (2014)'s work, a word is represented by the concatenation of a character-level and word-level embedding. The word-level embedding is created in a similar way to other word-level models. The character-level embedding is created through a convolutional layer that uses a max operation in order to obtain a fixed-size embedding, in a broad sense similar to Kim et al. (2016). The POS tagging system follows the window approach of Collobert et al. (2011). In this work, we are not replicating Santos and Zadrozny (2014) neural network, rather, we are interested in the approach of combining character-level with a word-level embedding. In Chapter 4 we will do experiments with the best character-level and word-level embeddings combined.

2.3 Evaluation

Christodoulopoulos et al. (2010) note that there are difficulties evaluating the output of an unsupervised model like described above. This is because the POS induction model outputs clusters, and these clusters are not easily mapped onto their gold standard equivalents. In order to be able to evaluate the performance of the model, we will use V-measure (Rosenberg and Hirschberg, 2007), as described in the following section. Other cluster evaluation methods exist, such as one-to-one mapping and many-to-one mapping, however, Christodoulopoulos et al. (2010) suggest that V-measure is less sensitive to the number of induced clusters than those two methods, and should therefore be used as an alternative to the standard measures.

2.3.1 V-measure

V-measure is an entropy-based way of external cluster evaluation, first proposed by Rosenberg and Hirschberg (2007). It compares a target clustering with the output clustering produced by a model and provides a measure of how comparable the two outputs are. It does so by measuring the extent to which homogeneity and completeness in the found clusters have been satisfied. It

calculates "accuracy" corresponding to F-measure. Where F-measure combines precision and recall, V-measure combines homogeneity and completeness.

F-measure is calculated as follows

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}. \tag{2.20}$$

F-measure is usually used in classification and information retrieval tasks. Precision here is the proportion of correctly classified instances of all instances, whereas recall is the proportion of correctly classified as 'A' instances from the total number of instances that have label 'A'. In V-measure, we seek analogues to precision and recall: namely homogeneity and completeness. Clustering satisfies homogeneity when each cluster contains only data points that are originally from the same single class. Completeness is satisfied when data with the same class label are clustered together in the same group.

V-measure is "the weighted harmonic mean of two values, homogeneity (h, the precision analogue) and completeness (c, the recall analogue)" (Christodoulopoulos et al., 2010, p. 587).

$$h = 1 - \frac{H(T|C)}{H(T)} \tag{2.21}$$

$$c = 1 - \frac{H(C|T)}{H(C)} \tag{2.22}$$

$$VM = \frac{(1 + \beta)hc}{(\beta h) + c} \tag{2.23}$$

As can be seen, homogeneity is calculated by looking at the conditional entropy of the distribution over class given the to-be-evaluated cluster (Rosenberg and Hirschberg, 2007). This value is zero in the perfect case, but to get a value close to 1 to be good and a value close to 0 to be bad, we invert this value. The formula for completeness is obtained due to symmetry between homogeneity and completeness.

Chapter 3

Methodology

The purpose of the project is to improve on the results achieved by Lin et al. (2015). In their paper, they compare using a Hidden Markov Model (HMMs) to using Conditional Random Field (CRF) autoencoders, both with multinomial output and Gaussian emissions. Since Gaussian emissions outperformed multinomial output and the difference in performance between using an HMM and using a CRF is small, this project will take the HMM as the model of interest. The aim is to improve the results of using an HMM with Gaussian emissions by experimenting with different word embeddings, with specific interest in word embeddings that encode syntactical information, and character-level embeddings.

3.1 Baseline model

The model by Lin et al. (2015) has been discussed in the previous chapter. The Hidden Markov model is implemented using the library `hmmlearn`[1], which offers an implementation of the Gaussian HMM, trained with the Baum-Welch algorithm. Calculating the V-measure given the clusters and gold standard tags has been made possible with `v_measure_score` from `sklearn.metrics`[2].

[1] https://github.com/hmmlearn/hmmlearn
[2] http://scikit-learn.org/stable/modules/generated/sklearn.metrics.v_measure_score.html

3.2 Datasets

3.2.1 English Wikipedia 2006 dump

The data used to train the embeddings is from Wikipedia. Specifically, it is the first 10^9 bytes of the English Wikipedia dump on 3rd of March, 2006[3]. The data is preprocessed using a perl script[4] which reduces the XML dump to clean text that consists of lowercase letters (a-z) and spaces, where all other characters are converted to spaces. Digits are spelled out, and all tables, tags, and their content are removed, except for image captions. Figure 9 shows that the Wikipedia data follows Zipf's law.

The original data looks like the following:

```
'''Anarchism''' originated as a term of abuse first used against early [[working
 class]] [[radical]]s including the [[Diggers]] of the [[English Revolution]] an
d the [[sans-culotte|''sans-culottes'']] of the [[French Revolution]].[http://uk
.encarta.msn.com/encyclopedia_761568770/Anarchism.html] Whilst the term is still
 used in a pejorative way to describe ''&quot;any act that used violent means to
 destroy the organization of society&quot;''&lt;ref&gt;[http://www.cas.sc.edu/so
cy/faculty/deflem/zhistorintpolency.html History of International Police Coopera
tion], from the final protocols of the &quot;International Conference of Rome fo
r the Social Defense Against Anarchists&quot;, 1898&lt;/ref&gt;, it has also bee
n taken up as a positive label by self-defined anarchists.
```

Here, the series of xml tags that precedes the start of the Wikipedia have been removed for clarity.

After preprocessing, which includes remove all tags, URL encoded characters, lowercasing everything, and replacing numbers by their words, the data looks like the following:

```
anarchism originated as a term of abuse first used against early working class
radicals including the diggers of the english revolution and the sans culottes
of the french revolution whilst the term is still used in a pejorative way to
describe any act that used violent means to destroy the organization of society
it has also been taken up as a positive label by self defined anarchists
```

For further experiments in the results section, we have produced different versions of the data by preprocessing it slightly differently. The following data sample is from leaving all punctuation by adding spaces around them so they will be recognised as tokens.

[3] http://download.wikipedia.org/enwiki/20060303/enwiki-20060303-pages-articles.xml.bz2
[4] http://mattmahoney.net/dc/textdata

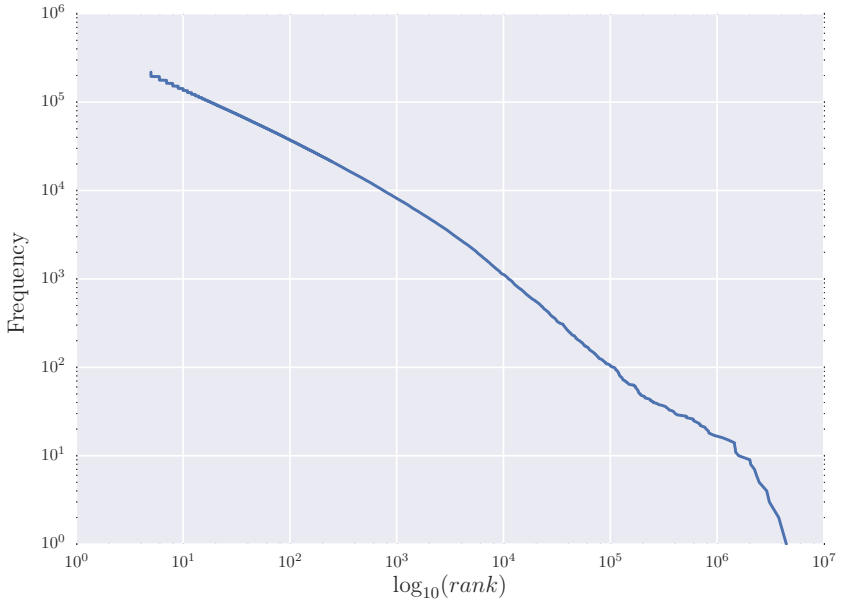

FIGURE 9: Plot the $\log_{10}$(rank) of a word against its frequency. Distribution of frequencies over words in English Wikipedia dataset follows Zipf's law.

```
'' 'anarchism '' ' originated as a term of abuse first used against early
working class radicals including the diggers of the english revolution
and the '' sans culottes '' of the french revolution . whilst the term is
still used in a pejorative way to describe '' any act that used violent means
to destroy the organization of society '' , it has also been taken up as a
positive label by self defined anarchists .
```

The third version of the data keeps punctuation and, on top of that, preserves capitalisation.

```
'' 'Anarchism '' ' originated as a term of abuse first used against early
working class radicals including the Diggers of the English Revolution
and the '' sans culottes '' of the French Revolution . Whilst the term is
still used in a pejorative way to describe '' any act that used violent means
to destroy the organization of society '' , it has also been taken up as a
positive label by self defined anarchists .
```

To refer to these differently preprocessed corpora, they will be denoted as Wiki-1, Wiki-2, and Wiki-3, in order of their degree of preprocessing. Wiki-1 is the fully preprocessed corpus, only keeping alphabetical characters and spaces; Wiki-2 preserves punctuation on top of Wiki-1, and Wiki-3 does not lowercase capital letters.

3.2.2 Penn Treebank - Wall Street Journal

Traditionally, the Wall Street Journal data of the Penn Treebank[5] is used for training and testing POS tagging systems. This project will also use the WSJ data, specifically release 3. The data is split into training, development, and testing data. The training data will use sections 00-18, the development data will use 19-21, and the test data will use the last sections 22-24.

All the tokens are tagged with their part-of speech. The content of the first sentence from section 00 is:

```
[ Pierre/NNP Vinken/NNP ]
,/,
[ 61/CD years/NNS ]
old/JJ ,/, will/MD join/VB
[ the/DT board/NN ]
as/IN
[ a/DT nonexecutive/JJ director/NN Nov./NNP 29/CD ]
./.

[ Mr./NNP Vinken/NNP ]
is/VBZ
[ chairman/NN ]
of/IN
[ Elsevier/NNP N.V./NNP ]
,/,
[ the/DT Dutch/NNP publishing/VBG group/NN ]
./.
```

3.3 The model pipeline

Firstly, the word embeddings are trained using the English Wikipedia data. If needed, they are converted to "word2vec" format, either binary or as text file. If the embeddings are saved as a text file, the first line needs to contain the number of embeddings and dimensions of the embeddings. Each line thereafter is the word, followed by the floating point entries of the vector representations, separated by spaces. A script to convert GloVe format to word2vec format has been provided by Manas Ranjan Kar[6]. The `gensim` library is used to load vectors into the

[5] https://catalog.ldc.upenn.edu/LDC99T42
[6] https://github.com/manasRK/glove-gensim

system[7] (Řehůřek and Sojka, 2010). As the next step an HMM is built with diagonal covariance, and to be trained in 10 iterations. Experiments have been run with 100 components and 45 components. The latter number equals the number of distinct tags in Penn Treebank. The training data for the HMM comes from the English Penn Treebank, sections 00-18. Each token in the Penn Treebank is replaced by its corresponding word embedding. The concatenation of all tokens and sentence lengths is then fed to the model to fit parameters. After training the model, it is used for evaluation, with the same embeddings it was trained with, on the development set. The development set consists of sections 19-21 of the English Penn Treebank. To evaluate the model, we are using V-measure as defined by Rosenberg and Hirschberg (2007).

3.3.1 Training word embeddings

This section will describe how the word embeddings are trained with respect to parameter settings, number of iterations, and training methods used. Unless otherwise specified, all embeddings have been trained on the Wiki-1 corpus, as the Wiki-2 and Wiki-3 corpora are reserved for further experiments after the main experiments.

word2vec Both skipgram and CBOW embeddings are trained, with context window of 1, for dimensions 20, 50, 100, and 200. A negative sampling rate is used with a value of 25, with a threshold of 1×10^{-4} for downsampling of frequent words. The vectors are trained with 15 iterations. The code for producing these word embeddings has been made public by the researchers[8].

wang2vec The cwindow and structured skipgram embeddings are trained with a context window of 1, for dimensions 20, 50, 100, and 200. As an objective, the noise contrastive estimation is used to approximate the softmax objective function. Similarly to word2vec, a threshold of 1×10^{-4} is used for downsampling of frequent words. The vectors are trained with 15 iterations. In order to prevent algebraic overflows, which can happen with wang2vec embeddings, the parameter cap is set to 1. An implementation by Ling et al. (2015a) has been provided on GitHub[9].

GloVe The GloVe embeddings are trained with a window size of 1 and with a maximum of 15 iterations. All other parameters and training methods are fixed to the standard ones used in Pennington et al. (2014). The code has been released by Pennington et al. (2014) on GitHub[10].

[7] https://pypi.python.org/pypi/gensim
[8] https://code.google.com/archive/p/word2vec/
[9] https://github.com/wlin12/wang2vec
[10] https://github.com/stanfordnlp/GloVe

SENNA The pretrained SENNA embeddings are taken from Collobert et al. (2011)'s website[11]. They are 50-dimensional. The data used to train them are from Wikipedia and Reuters. For more details see Collobert et al. (2011).

CharCNN The LSTM-CharCNN model by Kim et al. (2016) has been run with two different settings, namely CharCNN-Small and CharCNN-Large. Table 3.1 will show which models are trained and what their test set perplexity is on the language modelling task. We are interested in investigating the influence on POS tagging the embeddings made by the two well-performing LSTM models will have. The resulting embeddings will be quite large, 525 for CharCNN-Small, and 1100 for CharCNN-Large, so dimensionality reduction is necessary in order to create embeddings with similar size to the standard used (20, 50, 100, and 200). This will be done with Principal Component Analysis (PCA) using `PCA` from `sklearn.decomposition`. PCA is a linear dimensionality reduction method that uses Singular Value Decomposition to project the data in a lower dimensional space. Moreover, we are also interested in the embeddings the LSTM-CharCNN model makes if the parameters are set in such a way that it will directly create embeddings of appropriate sizes. To realise this the sum of the sizes of the feature maps need to be 20, 50, 100, and 200. Table 3.1 outlines the kernel widths and resulting feature map sizes chosen in order to achieve embeddings of those sizes, keeping in line how the feature map sizes are chosen for the two well-performing language models (CharCNN-Small, CharCNN-Large). We keep all other hyperparameters the same as the settings of CharCNN-Small. The model has been implemented in Torch[12]. The complete language model has been provided by the researchers[13]. An own implementation was necessary for extracting the embeddings from the neural language model. The language model has been trained on Penn Treebank, which is conventional for language modelling. The embeddings are created using a vocabulary of words sorted by frequency obtained from scanning the Wiki-1 training corpus. Each word is fed through the CharCNN and word plus embedding are saved to disk.

Santos and Zadrozny (2014) In Santos and Zadrozny (2014) a combination of a word-level and character-level embedding are used to improve supervised part-of-speech tagging. As the final step we will therefore concatenate the best performing word-level embedding with the best performing character-level embedding, in hope that the POS induction system will do very well.

[11] http://ronan.collobert.com/senna/
[12] http://torch.ch/
[13] https://github.com/yoonkim/lstm-char-cnn

	20-D	50-D	100-D	200-D	CharCNN-Small	CharCNN-Large
Number of kernels	2	3	4	5	6	7
Kernel widths	{1,2}	{1, 2, 3}	{1, 2, 3, 4}	{1, 2, 3, 4, 5}	{1, 2, 3, 4, 5, 6}	{1, 2, 3, 4, 5, 6, 7}
Feature maps	10, 10	10, 15, 25	10, 15, 25, 50	10, 15, 25, 50, 100	25, 50, 75, 100, 125, 150	50, 100, 150, 200, 200, 200, 200
Embedding size	20	50	100	200	525	1100
RNN size	300	300	300	300	300	650
Char-vec size	15	15	15	15	15	15
Highway layers	1	1	1	1	1	2
Test perplexity	181.26	112.15	105.79	98.58	92.52	79.18

TABLE 3.1: Overview of different LSTM-CharCNN models and their parameters.

Chapter 4

Experiments and Results

4.1 Results on part-of-speech tagging

This section will go over the results from the experiments and report the performance on the development set. First, it will outline the V-measure scores obtained by the two GHMM models with 45 and 100 components and compare these to the scores in the Lin et al. (2015) paper. It will then compare the GHMM with 45 and 100 components with each other, to determine the differences that both models create when run with different embeddings. So far, the results section has focused on the Gaussian HMM model, but hereafter it will focus more on the qualitative aspects of the created word embeddings. These will be illustrated in two manners: we will find the nearest neighbours for a couple of interesting words for each of the best performing embeddings; and we will also use t-Distributed Stochastic Neighbour Embedding (t-SNE) (Van Der Maaten and Hinton, 2008) to project the embeddings to a lower dimensional space and inspect the clusters found. The next section will discuss the results obtained by using an embedding for out-of-vocabulary words; see if results can be improved if the data is tokenised in a similar way as PTB; and take Santos and Zadrozny (2014)'s approach of concatenating word-level and character-level embeddings on the POS induction task. After these further experiments, we report the final results on the test set.

4.1.1 HMM with 100 components

While Lin et al. (2015) mainly experimented with skipgram and structured skipgram embeddings, we have been able to try out five more embeddings in addition to replicating their work. Apart from skipgram and structured skipgram, we explore the performance of continuous bag-of-words (CBOW), Global Vectors (GloVe), continuous window (cwindow), SENNA (Semantic/syntactic Extraction using a Neural Network Architecture), and character-level embeddings.

dim	skipgram	CBOW	GloVe	cwindow	structured
20	0.5120	0.5261	0.5248	**0.5879**	0.5624
50	0.5283	0.5438	0.5234	0.5822	0.5711
100	0.5400	0.5438	0.5407	0.5780	0.5510
200	0.5319	0.5418	0.5329	0.5663	0.5364

TABLE 4.1: V-measure for different word embeddings and dimensions with HMM with 100 components.

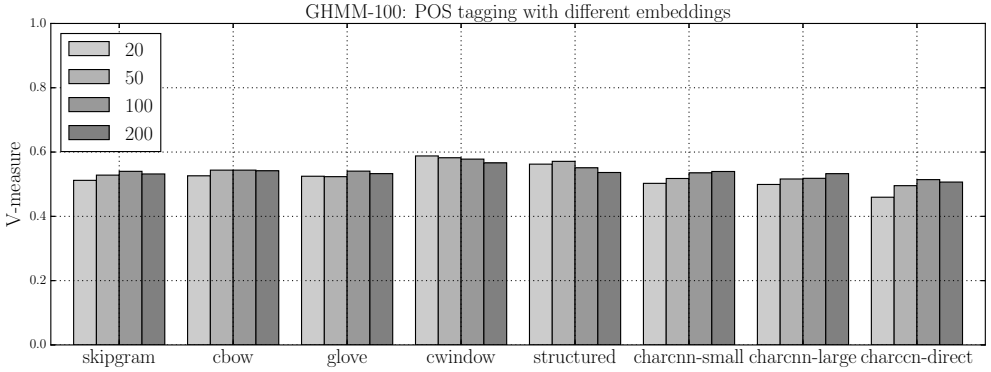

FIGURE 10: Cluster quality in V-measure for the different embeddings on POS tagging with 100 components.

Table 4.1 shows the V-measures obtained for each of the different embeddings. An underlined value means that it is the highest V-measure for that specific embedding, whereas a bold-type value means it is the best value overall in the table. We have been able to reproduce the results of Lin et al. (2015) using the skipgram and structured skipgram models in the Gaussian HMM and are able to confirm that structured skipgram outperforms the regular skipgram for almost every dimension. The V-measures obtained are also in line with Lin et al. (2015)'s work, being around 0.5 and slightly higher.

The table also shows that CBOW embeddings are better than skipgram embeddings for all dimensions, and similarly, cwindow embeddings outperform structured skipgram embeddings for each dimension. In general, the wang2vec vectors, cwindow and structured skipgram, do better than their word2vec counterparts. If we now take a look at GloVe, we see that their performance is not necessarily bad, but it does not stand out compared to the other embeddings in Table 4.1. Its performance is on par with skipgram, with minute differences making it slightly better.

The table does not present the perforance of the pretrained SENNA embeddings, which has been the best up until now. Their dimensionality is 50 and the resulting V-measure obtained is 0.6395, which is better than any other type of embedding so far. Unfortunately, there have been some problems with training SENNA embeddings with Attardi (2015)'s toolkit and the

source code of Al-Rfou et al. (2013), so we have not been able to train these embeddings on the same data set as the others, and in other dimensions than 50.

Regarding embedding size, Lin et al. (2015) found that a dimension of 100 is the worst size for the POS task. We do not necessarily agree with this, as word2vec does well with a dimension of 100. Figure 10 shows a visual comparison of the V-measure scores which we presented in Table 4.1 and Table 4.3. Overall, a larger dimension size is slightly better for word2vec and GloVe embeddings. This differs for the wang2vec embeddings, which seem to perform better with a smaller dimension.

The previous section on prediction-based models purely focuses on embeddings with a context window of 1, which Lin et al. (2015) found to be the best window size for syntactic tasks. We have been able to replicate this finding with the GloVe embeddings. Indeed, the larger the window size grows, the worse the performance becomes. The V-measure for a context window of 15, as seen in Table 4.2, has become the lowest overall score for the Gaussian HMM with 100 components.

window size	2	5	10	15
V-measure	**0.5148**	0.4900	0.4838	0.4753

TABLE 4.2: Different context window sizes for GloVe with dimension 20 and HMM with 100 components.

Before moving on to examine the results with Gaussian HMMs where we employ 45 components, we take a look at the performance of the character-level embeddings as presented in Table 4.3 and depicted on the right in Figure 10. As discussed in Chapter 3, the two best settings for the language model by Kim et al. (2016) have been used to create CharCNN-Large, which has an embedding size of 1100, and CharCNN-Small, with an embedding size of 525. In order to make them comparable to the prediction-based and statistics-based models, we have applied Principal Component Analysis (PCA) to obtain embeddings of dimension 20, 50, 100, and 200. We have also adjusted the parameters of the language model to directly obtain embeddings of these sizes (see Table 3.1). We name these embeddings of the latter type CharCNN-Direct.

dim	CharCNN-Large	CharCNN-Small	CharCNN-Direct
20	0.5026	0.4992	0.4595
50	0.5178	0.5162	0.4954
100	0.5353	0.5183	0.5142
200	**0.5395**	0.5328	0.5067

TABLE 4.3: V-measure for CharCNN word embeddings with HMM with 100 components.

Table 4.3 shows the results of POS induction with these embeddings. Here it is more clear that larger embeddings do better, which is visible in Figure 10. We see that CharCNN-Direct

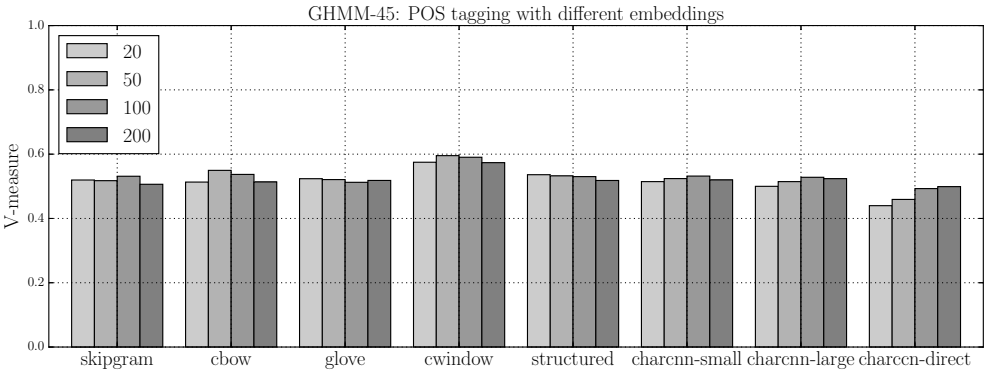

FIGURE 11: V-measure for different word embeddings and dimensions with HMM with 45 components, including character-level embeddings.

performs worse than the other two embeddings and we notice that CharCNN-Large seems to dominate in every dimensionality over the other two types of embeddings.

4.1.2 HMM with 45 components

All experiments in the previous section have been rerun with 45 components, which are the exact number of tags in the PTB POS tag set. Table 4.4 shows the scores obtained for the different embeddings ran on the GHMM model with 45 components. All V-measures seem to be slightly lower, except for cwindow, where a dimension of 50 gives the best result across the values in Table 4.4. On average, smaller embeddings do better on the GHMM with 45 components. We observe the same trends as with 100-component GHMM: CBOW is better than skipgram, which is in parallel with cwindow being better than structured skipgram. On average wang2vec does better than word2vec, with cwindow being best overall. GloVe has moderate performance compared to the other embeddings. Again, the pretrained embeddings by Collobert et al. (2011) are the best overall with a V-measure of 0.6490. Section 4.1.3 will go into a bit more depth on the differences between these two settings.

dim	skipgram	CBOW	GloVe	cwindow	structured
20	0.5196	0.5132	0.5236	0.5749	0.5360
50	0.5175	0.5496	0.5209	**0.5822**	0.5325
100	0.5313	0.5370	0.5124	0.5780	0.5302
200	0.5063	0.5135	0.5182	0.5663	0.5181

TABLE 4.4: V-measure for different word embeddings and dimensions with HMM with 45 components.

Table 4.5 shows the results of the character-level embeddings, where dimensionality reduction was performed with PCA. Table 4.6 also shows the performance of the character-level embeddings as they get produced in the LSTM-CharCNN without being changed. While the values

in Table 4.6 for CharCNN-Large are around the average of values for their reduced counterpart in Table 4.5, the values for CharCNN-Small are slightly worse than their reduced counterpart. Both unchanged embeddings do better than CharCNN-Direct. The values in Table 4.4 and Table 4.5 are visualised in Figure 11 as well.

dim	CharCNN-Large	CharCNN-Small	CharCNN-Direct
20	0.4999	0.5146	0.4397
50	0.5146	0.5240	0.4592
100	0.5281	**0.5318**	0.4929
200	0.5238	0.5199	0.4989

TABLE 4.5: V-measure for CharCNN word embeddings with HMM with 45 components, where the dimensionality of Large and Small has been reduced with PCA.

model settings	CharCNN-Large (1100)	CharCNN-Small (525)
GHMM-45	0.5160	0.5039
GHMM-100	**0.5212**	0.5121

TABLE 4.6: V-measure for CharCNN word embeddings with the two GHMM models, no dimensionality reduction.

In Table 3.1 we show the hyperparameters with which we trained several LSTM-CharCNN language models. For each of the settings, we note the perplexity achieved on the test set of PTB. From each language model differently-sized embeddings have been extracted as input to the POS induction system. The perplexities for each language model have been plotted against the V-measures achieved from running the POS tagging system in Figure 12. A low perplexity means that the language model is doing well. We note two striking things: as the perplexity goes up, we see performance on the POS induction task go down and we observe that for the character-level embeddings, on average, the 100 component GHMM does better than the 45 component GHMM.

4.1.3 A comparison between 45 and 100 components

The average V-measure for all experiments with the GHMM with 100 components is 0.533, whereas the average for 45 components is 0.523. Furthermore, when the V-measures for embeddings are averaged across all dimensions, we note that for each embedding the V-measure for the 100-component GHMM is higher, except for cwindow, where it is the opposite. The results of taking the mean of the V-measures for each dimension across all different embeddings are presented in Table 4.7.

This table also shows that the V-measures for GHMM-100 are all slightly higher than for GHMM-45. It can also be observed that the embedding size with the overall best score is dimension 100. Although the number of tags used in the Penn Treebank tag set is 45, there is

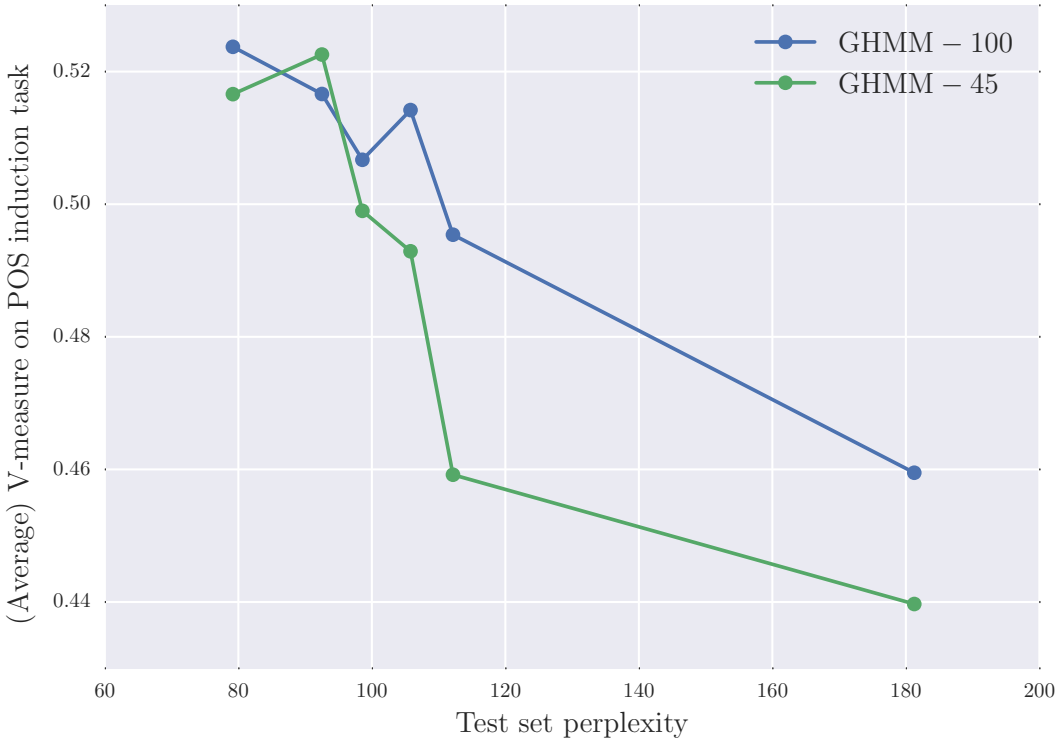

FIGURE 12: Influence of perplexity of the language model on the (average) v-measure for character-level embeddings.

dim	GHMM-45	GHMM-100
20	0.5152	0.5218
50	0.5267	0.5348
100	0.5318	**0.5402**
200	0.5216	0.5360

TABLE 4.7: A comparison of the average V-measure of all embeddings for different dimensions and GHMM settings.

no clear reason why more states in an HMM produces slightly better results. We remind the reader that V-measure is composed of the score for homogeneity and completeness as defined in Section 2.3.1. It would be interesting to inspect the homogeneity and completeness scores for GHMM-45 and GHMM-100 and see if a trade-off can be identified. Therefore, we calculate the homogeneity and completeness, along with V-measure for cwindow-50 and present this in Table 4.8.

model	V-measure	homogeneity	completeness
GHMM-45	0.5954	0.6288	0.5653
GHMM-100	0.5822	0.6667	0.5167

TABLE 4.8: A comparison of V-measure, homogeneity, and completeness for cwindow-50 for GHMM-45 and GHMM-100.

To repeat Section 2.3.1, homogeneity is satisfied when each cluster only contains data points from the same class. Completeness is satisfied when data points with the same class label are found clustered together in the same group. The V-measures for GHMM-45 and GHMM-100 are very close as they only differ by 0.132 from each other. However, the homogeneity and completeness scores diverge much more. With 45 states in the Hidden Markov model, which is equal to the number of tags (and therefore number of clusters), the completeness score is higher relative to GHMM-100, whereas the homogeneity score is lower. This is because we have the same number of clusters in the predicted labels as in the gold standard, which means that clusters in the predicted labels have less chance to be spread out. With GHMM-100, the system deals with some sort of "many-to-less" cluster system, which means that multiple differently labelled groups may be assigned to the same cluster in the gold standard. Each of these groups can therefore be homogeneous, but if they are truly mapped to one cluster in the gold standard, completeness will suffer. In short, using more clusters than there are tags will be a trade-off between completeness for homogeneity.

4.2 Quality of word embeddings

The previous section has shown the results from experimenting with all the different word embeddings. To gain more information on the embeddings we have trained, this section will move away from the quantitative evaluation of the embeddings, but will rather look at the embeddings from a qualitative viewpoint. We will do this in two ways: firstly, we will perform exhaustive search on the top 5 nearest neighbours of specific interesting words by using cosine similarity as a distance metric; and secondly, we will view the clusters obtained by projecting the embeddings down to a lower dimensional space using t-distributed Stochastic Neighbour Embedding (t-SNE) (Van Der Maaten and Hinton, 2008).

4.2.1 Nearest neighbours of embeddings

Following the trend of several papers to list nearest neighbours of interesting words in order to look at the quality of embeddings, this section will take the best setting for each type of word embedding and compare them with each other. The following list of words has been compiled from examining the words used in similarity tasks in Collobert et al. (2011); Kim et al. (2016); Ling et al. (2015a,b); Luong et al. (2013); Santos and Zadrozny (2014). The chosen embeddings are skipgram-100, CBOW-50, GloVe-20, cwindow-50, structured-skipgram-20, SENNA, CharCNN-Large-100, and CharCNN-Small-100. The distance metric we used is cosine similarity. To find the nearest neighbours the `distance` tool from word2vec is used, which performs exhaustive search. The 20.000 most frequent words are taken from each word embedding. In Table 4.9 the semantically interesting words *person, king, france, reddish*, and *richard* are shown

with their top 5 nearest neighbours. Table 4.10 shows "syntactically" interesting words *while, his, you, their* and *about*, which in POS tagging have one of the less ambiguous tags. Table 4.11 shows words that are built up from prefixes and suffixes, such as *inconsiderable* and *unsteadiness*, as well as word-pairs that are similar, but differ in prefixes or suffixes, such as *commenting, comment, unaffected*, and *affect* We hope to find words with similar meaning; words with similar POS tag; and words with similar morphology for these sets of words respectively.

4.2.1.1 Semantically interesting words

In Table 4.9 there is a divide between the word-level embeddings on one side and the character-level embeddings on the other side. The word-level embeddings mostly show words with a similar meaning, and if not, they show words from the same part-of-speech. The character-level embeddings move completely away from this and mainly show words with a similar structure as the word of interest. For the word *person*, its nearest neighbours for the word2vec and wang2vec families are mostly people. GloVe and SENNA here show some noisy nearest neighbours. On the other hand, the character embeddings show words that look like variations of spelling the word *person* or that have the same subword per- or -son. Analysing the other nearest neighbours of the character embeddings we see that they generally contain subwords of the target worfds. Where all of the other word vectors show other countries for the query *france*, the character vectors return words that contain *fran(c)(e)*, such as *franc, franco*. For the name *richard* all other embeddings return English names for men, where the CharCNN embeddings return words structurally close to *richard*, such as other names (*richardson*) or words (*orchard*). Comparing the word-level embeddings we see good nearest neighbours for the word *reddish* for skipgram and CBOW that are adjectives describing colour ending on *-ish*. The wang2vec embeddings return adjectives that describe appearance. GloVe and SENNA differ from these embeddings as they return a mix of adjectives, nouns, and verbs. In short, the word2vec embeddings result in the most "correct" nearest neighbours whlie wang2vec returns mostly semantically similar words, if not words with similar POS tag. SENNA and GloVe show noisy nearest neighbours; and the character-level embeddings diverge completely by showing similarly spelled words.

4.2.1.2 Syntactically interesting words

Table 4.10 shows the nearest neighbours of five words with fairly unambiguous part-of-speech tags. The nearest neighbours for the character embeddings are again structurally similar words. This task was designed to hopefully reveal nearest neighbours that are of the same part-of-speech tags. For the word *while* we indeed get other coordinating conjunctions (CC). It can be argued that the CCs shown for word2vec, wang2vec, and GloVe are of slightly higher quality than the ones obtained for SENNA. Ignoring the word "clothing", the other words for SENNA

do not express the same meaning as much as the first four embeddings do. For the word "his", word2vec, wang2vec, and SENNA return good results, whereas GloVe seems to return both similar words to "his", as well as words that are very likely to succeed the target word (like *career* or *life*). This is also true for the word *you* and GloVe's nearest neighbours. The nearest neighbours for the other embeddings are very noisy, which is most likely due to *you* being a high frequency word in the vocabulary. Regarding the word *their*, we see it roughly returns the same group of words as for the word *his*, since they are both possessive pronouns (PRP$). Surprisingly, when looking for nearest neighbours of the word *about*, the structured wang2vec embeddings return better results than word2vec. Since the main difference between those two embeddings are that wang2vec has specific parameters for the order of words, it may be that this is the explanation for this discrepancy. Finally, the word *about* returns rather noisy results, compared to wang2vec and SENNA, which is most likely due to the focus on word order.

4.2.1.3 Morphologically interesting words

With the morphologically interesting words, we finally reap the benefits of character-level embeddings. For the words *inconsiderable, unsteadiness, commenting, comment*, and *unaffected* in Table 4.11, the character-level embeddings are able to produce nearest neighbours with the same morphemes as the target words. In the case of *inconsiderable*, it is close in vector space to other words with the subwords "consider-", "in-', "-con-", "-er-", and "-able". This holds for the other words as well. The other embeddings have a harder time of returning the correct part-of-speech words for the target words. Here with "correct part-of-speech word" is meant that the returned word is of the same part-of-speech as the target word. Furthermore, for a lower-frequency word such as *inconsiderable* the other embeddings have a hard time to even find semantically related words. The SENNA embeddings unfortunately do not include *inconsiderable* or *unsteadiness* in its vocabulary. Another interesting result is for the word *commenting*: apart from SENNA, all embeddings exclusively return verbs, but not all of them return the same type of verb needed here, which is a VBG (verb, gerund, or present participle). The character embeddings return the correct part-of-speech words for *commenting*. The word *comment* can be a noun or verb both. The word2vec/wang2vec embeddings are able to mostly return words that can be both noun and verb, whereas the character-embeddings return mostly noun words. We further see almost all embeddings returning a VBD (verb, past tense) or VBN (verb, past participle) for *unaffected*, again except for SENNA. Finally, for *affect* we see the word-level embeddings returning verbs that are used in similar context. Note that they are not necessarily very close semantically. The character-level vectors again return words with similar subwords.

word	skip-100	CBOW-50	GloVe-20	cwin-50	struct-20	SENNA	Large-100	Small-100
person	woman	woman	true	woman	woman	letter	peterson	peterson
	persons	child	man	child	firm	case	persons	pension
	anyone	settlor	result	man	child	honor	pearson	persons
	patient	spouse	real	nation	hobby	ages	emerson	pearson
	victim	persons	woman	girl	demon	problem	patterson	poisson
king	kings	hussa	prince	prince	queen	lord	ping	ping
	queen	theodric	alexander	queen	lord	queen	ming	qing
	harthacanute	kings	charles	captain	lady	emperor	qing	kind
	mordha	sillok	edward	bishop	prince	titles	kind	bing
	monarch	culen	henry	lord	grace	fighting	ring	kong
france	spain	belgium	rome	spain	cuba	santa	franc	trance
	italy	italy	germany	belgium	luxembourg	composition	franco	franc
	belgium	spain	spain	italy	guatemala	germany	frances	franco
	germany	bordeaux	portugal	austria	portugal	italy	franca	ordnance
	luxembourg	luxembourg	japan	poland	peru	sweden	francs	franca
reddish	grayish	grayish	lime	yellowish	whitish	violet	resisted	yiddish
	greyish	greenish	honey	grayish	wavy	territorial	yiddish	swedish
	yellowish	blackish	trout	bluish	feathered	academia	revised	rush
	greenish	purplish	yellow	mottled	coppery	aggregate	registered	dish
	bluish	irides	fat	red	bulbous	tore	swedish	irish
richard	robert	robert	robert	william	william	robert	richards	orchard
	walter	philip	george	harold	harold	peter	richardson	richards
	francis	william	francis	robert	stephen	reportedly	orchard	richardson
	hugh	ralph	james	charles	albert	david	richland	richland
	arthur	john	thomas	hugh	john	william	archaic	richmond

TABLE 4.9: Nearest neighbours for semantically interesting words.

word	skip-100	CBOW-50	GloVe-20	cwin-50	struct-20	SENNA	Large-100	Small-100
	whilst	whilst	but	whilst	and	clothing	chile	whale
	furthermore	whereas	when	whereas	or	and	whale	whole
while	whereas	besides	however	furthermore	that	when	whole	chile
	although	alongside	although	moreover	where	but	white	whistle
	besides	and	making	additionally	although	although	ile	meanwhile
	her	her	her	their	their	their	hiss	hiss
	their	their	life	its	its	its	tis	is
his	its	its	their	her	our	her	hiv	hs
	the	my	career	our	whose	your	sis	ois
	my	hagbarthus	own	my	her	the	vis	this
	we	we	let	we	we	me	yu	ou
	jiveman	somebody	me	they	ye	diabetes	ou	your
you	copyedit	hucklebuck	we	ye	yours	sandwich	your	od
	somebody	me	know	anybody	yourself	palestinian	wu	young
	ohhh	yourself	say	bidu	bugs	we	yo	oo
	its	its	positions	its	its	his	heir	heir
	his	his	own	his	his	its	either	the
their	her	her	its	our	our	sandwich	tor	tie
	our	our	his	my	whose	your	esther	either
	the	your	life	your	your	my	ir	ether
	regarding	over	how	around	in	over	abbott	trout
	concerning	microcosmos	besides	over	between	in	abbot	bout
about	reword	regarding	quotes	in	around	that	bout	abbott
	nonvanity	indepth	notes	between	since	around	trout	layout
	basepairs	concerning	what	from	reaching	outside	boot	abbot

TABLE 4.10: Nearest neighbours for syntactically interesting words.

word	skip-100	CBOW-50	GloVe-20	cwin-50	struct-20	SENNA	Large-100	Small-100
inconsiderable	denominate	sphenic	endear	obligatory	destabilizing		considerable	considerable
	policyholders	discounting	materialise	signifigant	satisfactory		inconquerable	indecomposable
	signifigant	surprising	disappoint	unimpeachable	foolproof		inconsolable	inconquerable
	extortionate	denormal	congenial	unaffordable	scientifically		insufferable	incommensuable
	substantial	staggering	distasteful	explicit	realigning		imponderable	unconscionable
unsteadiness	neuralgic	equilibrioception	zelotes	decompensated	thymoma		unreadiness	unreadiness
	schizotypy	neuralgic	christopher	dyserythropoietic	mechnical		steadiness	uneasiness
	insensibility	persecutory	wck	microcornea	hemiplegic		untidiness	steadiness
	vestibulo	schizotypy	ranko	expressivity	hyperglycemic		uneasiness	untidiness
	unnaturalness	diffractive	hypernatremia	morphea	indirectness		steadfastness	sturdiness
commenting	commented	commented	insisting	insisting	insisted	hackers	committing	committing
	gloating	criticizing	sells	insists	insisting	possessed	competing	commemorating
	raved	insisting	binds	focusing	commented	corvette	commanding	commanding
	remarked	remarking	insists	concentrating	speculating	cyborg	coming	coming
	joked	discussing	enabled	insisted	insists	jtdirl	connecting	competing
comment	comments	comments	request	remark	consensus	lyon	commencement	commitment
	remarks	reply	notice	report	slump	marco	commitment	commencement
	reply	remark	finding	notice	shame	nan	competent	component
	remark	remarks	account	statement	ban	thebes	clement	competent
	quip	quip	permission	commentary	commentary	orchestras	comments	clement
unaffected	affected	obscured	disruped	affected	affected	micronesia	affected	infected
	obscured	disturbed	outdated	disrupted	disrupted	baptist	unwanted	affected
	disturbed	hindered	motivated	regulated	obscured	brett	unidentified	inflicted
	untouched	affected	unreliable	damaged	enforced	apparatus	uninhabited	effected
	evident	compromised	unstable	overwhelmed	regulated	trenton	unfinished	unidentified
affect	impair	contribute	reject	reflect	utilize	ngo	effect	effect
	contribute	impair	observe	disrupt	involve	paula	affects	affects
	depend	reflect	recognize	eclude	activate	cabin	affected	affected
	affecting	alter	deny	involve	utilise	collier	affecting	affecting
	affects	relate	arise	satisfy	overwhelm	mentally	affection	defect

TABLE 4.11: Nearest neighbours for morphologically interesting words.

4.2.2 Visualising embeddings

The previous section searched for nearest neighbours of specifically chosen words, which is one way of assessing embedding quality. Another often used method is visualising them in vector space. Since the embeddings are high dimensional and therefore hard to visualise directly, t-distributed stochastic neighbour embedding (t-SNE) will be applied on the vectors in order to create easily visualisable 2D embeddings (Van Der Maaten and Hinton, 2008). This is an algorithm which transforms similarities between data points to joint probabilities. It then aims to minimise the Kullback-Leibler divergence between the joint probabilities of the high dimensional data and the low-dimensional embedding (Van Der Maaten and Hinton, 2008). This specific method creates 2D or 3D plots where points close to each other in the low-dimensional space are also close and/or similar to each other in the high dimensional representation.

In order to reduce clutter, only the first 2000 words have been taken from the vocabulary, sorted by frequency. Figure 13 shows the results of applying t-SNE on the CharCNN-Small embeddings of size 100 after running the algorithm for 1000 iterations, where Figures 14, 15, and 16 show zoomed-in clusters. These clusters show groups of words with similar morphemes, for example verbs ending in -ed, as displayed by Figure 14. Figure 17 shows the cwindow embeddings with clusters in Figures 18, 19, and 20.

4.3 A solution for out-of-vocabulary words

The general approach to solving for out-of-vocabulary (OOV) words is to use an averaged vector, rather than training for a lot of low frequency or OOV words. As part of the section of further experiments, the CharCNN-Small embeddings of size 100 have been taken, and all words with a frequency of 5 or less are removed. The embeddings from these words have been averaged and added to the other embeddings as the "OOV" embedding. In the previous experiments, if a word is OOV, the zero vector will be used as its embedding.

When using the zero vector for OOV words, the CharCNN-Small embedding of dimension 100 result in a V-measure of 0.5318. Using the approach outlined above, we obtain a V-measure of 0.5247, which is a little bit lower than without an OOV embedding. Although this experiment only compares one of several types of embeddings, it seems to show that a special OOV embedding may not be able to increase performance by much.

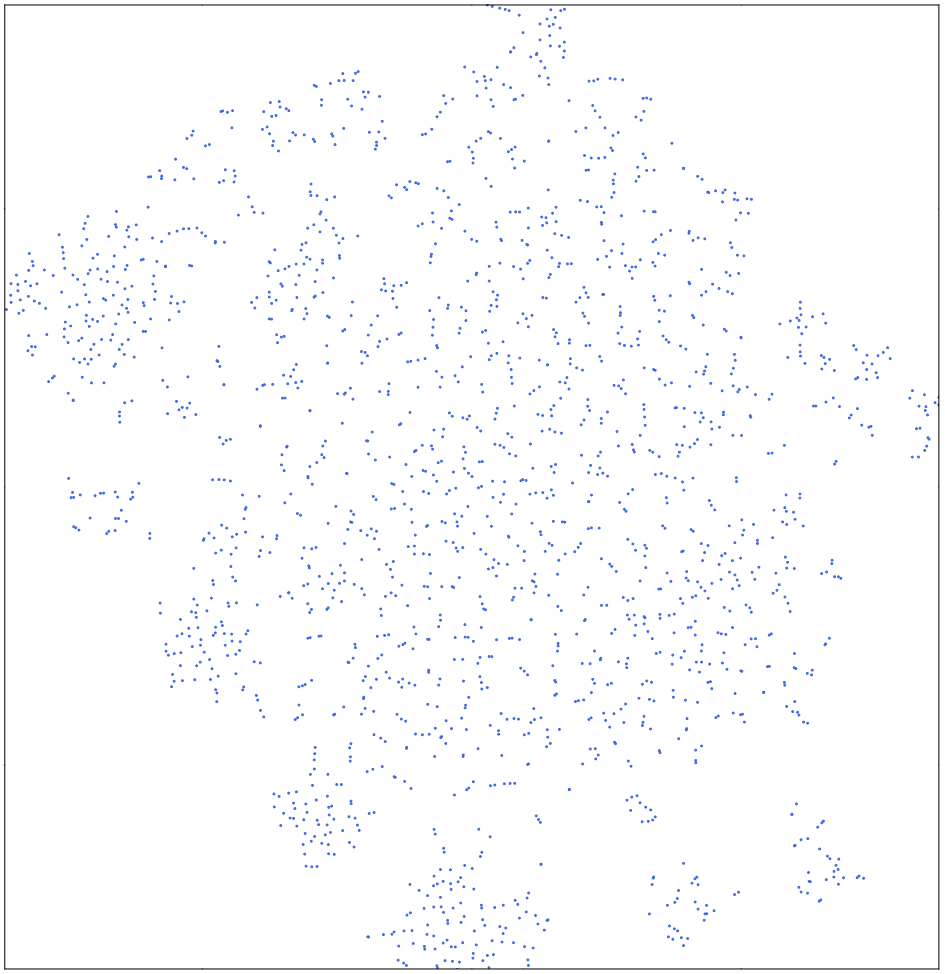

FIGURE 13: Visualisation of CharCNN-Small embeddings projected down to 2D space with t-SNE.

4.4 Leveraging more information by keeping capitalisation and punctuation

A point of criticism for the manner in which the embeddings have been trained is that all punctuation and capitalisation have been removed, which as can be seen in Chapter 3, are part of the Penn Treebank WSJ data. By removing punctuation, we are removing the option of having a representation for it, and therefore we are assigning the zero vector to punctuation marks. Punctuation have their own tag in the PTB data set, and knowing where a comma is placed provides additional information to a sequence tagger. In a similar manner, having embeddings for capitalised words can help in a number of ways. Firstly, the first word of a

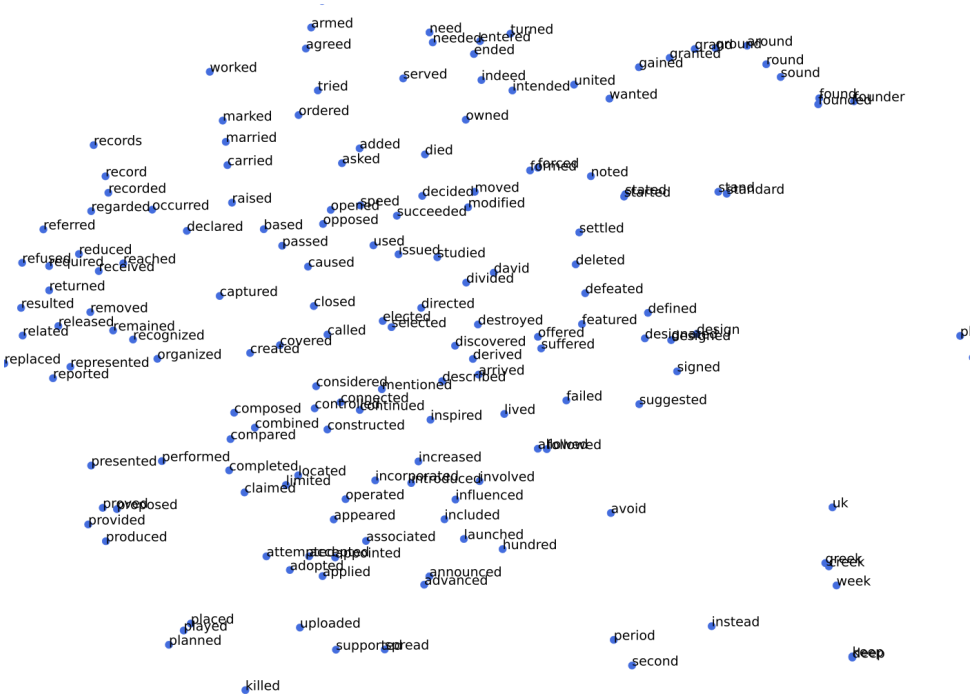

FIGURE 14: CharCNN-Small projected down to 2D space with visible cluster for words ending in -ed.

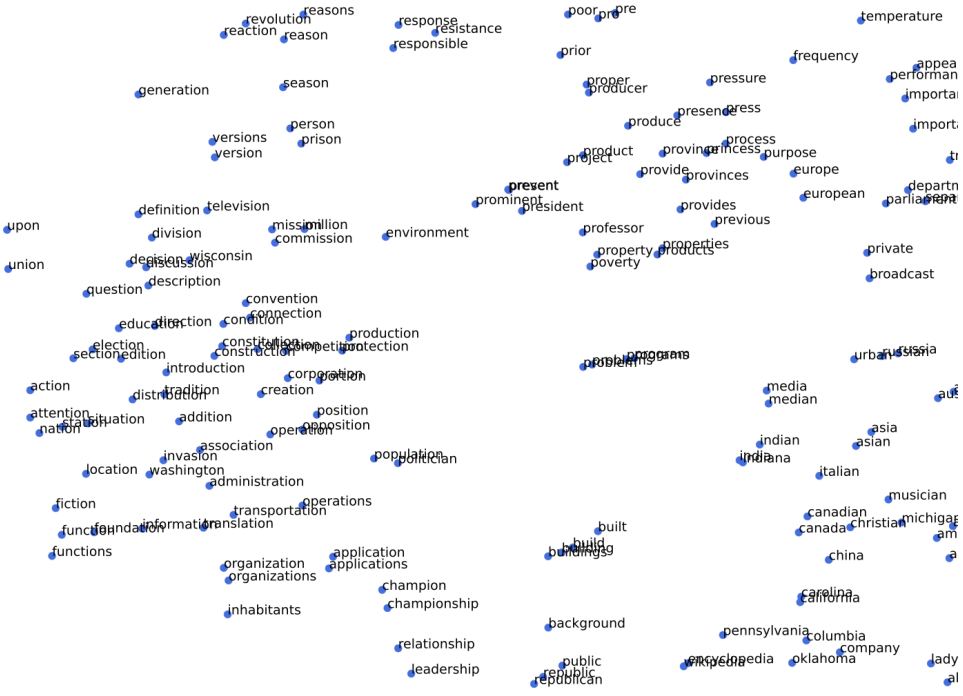

FIGURE 15: CharCNN-Small projected down to 2D space with visible clusters for words ending in -ion(s) and -ship.

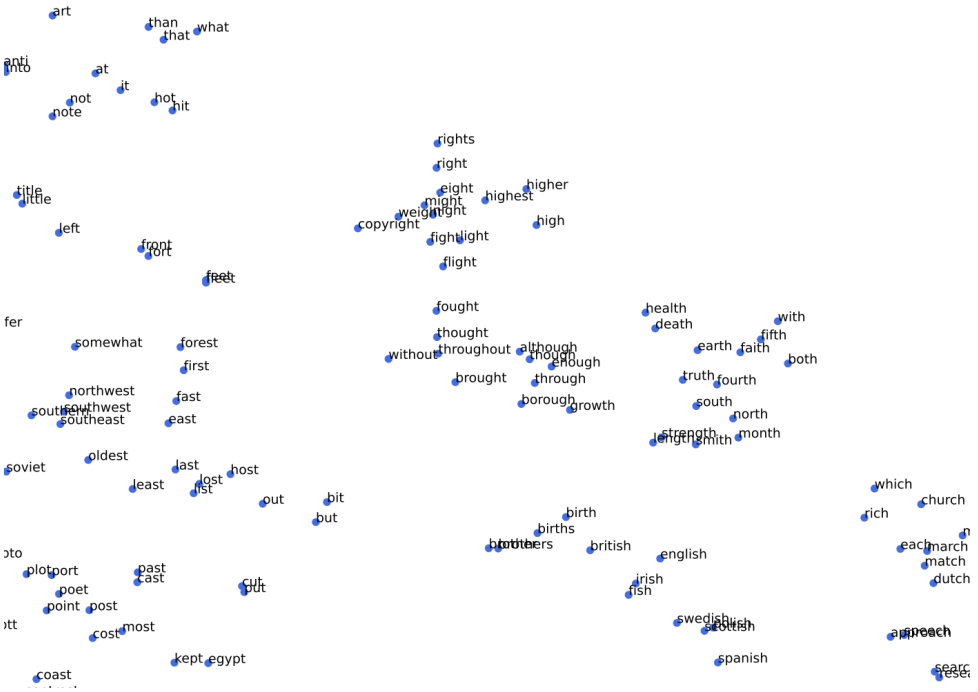

FIGURE 16: CharCNN-Small projected down to 2D space with visible clusters for words with -ought and ending in -ish.

sentence is capitalised and knowing that a word is the first in a sentence can provide information on what tag it is. Some part-of-speech are statistically more likely to occur at the start of the sentence than others. Secondly, if the capitalised word is not the first word of a sentence it is more likely to be a location or a geographical term, a noun or adjective derived from the former, or an abbreviation. This shows clearly that there is valuable information in capitalised words and/or punctuation versus uncapitalised words.

To evaluate this hypothesis, we take a subset of the embeddings evaluated in Section 4.1.2 and train them on the Wiki-2 and Wiki-3 datasets. A comparison of the results can be found in Table 4.12. The numbers in the first column are from Table 4.1 and Table 4.4 and they are repeated here for comparison.

embedding	dim	Wiki-1	Wiki-2	Wiki-3
skipgram	100	0.5313	0.5798	0.5990
CBOW	100	0.5370	0.6097	0.6152
GloVe	20	0.5236	0.5811	0.5786
cwindow	50	0.5822	0.6210	**0.6572**
structured	20	0.5360	0.5995	0.6105
CharCNN-Large	100	0.5281	0.5448	0.5086

TABLE 4.12: V-measures of skipgram and CBOW on Wiki-1 (completely processed), Wiki-2 (preserving punctuation), and Wiki-3 (punctuation and capitalisation) with 45 components.

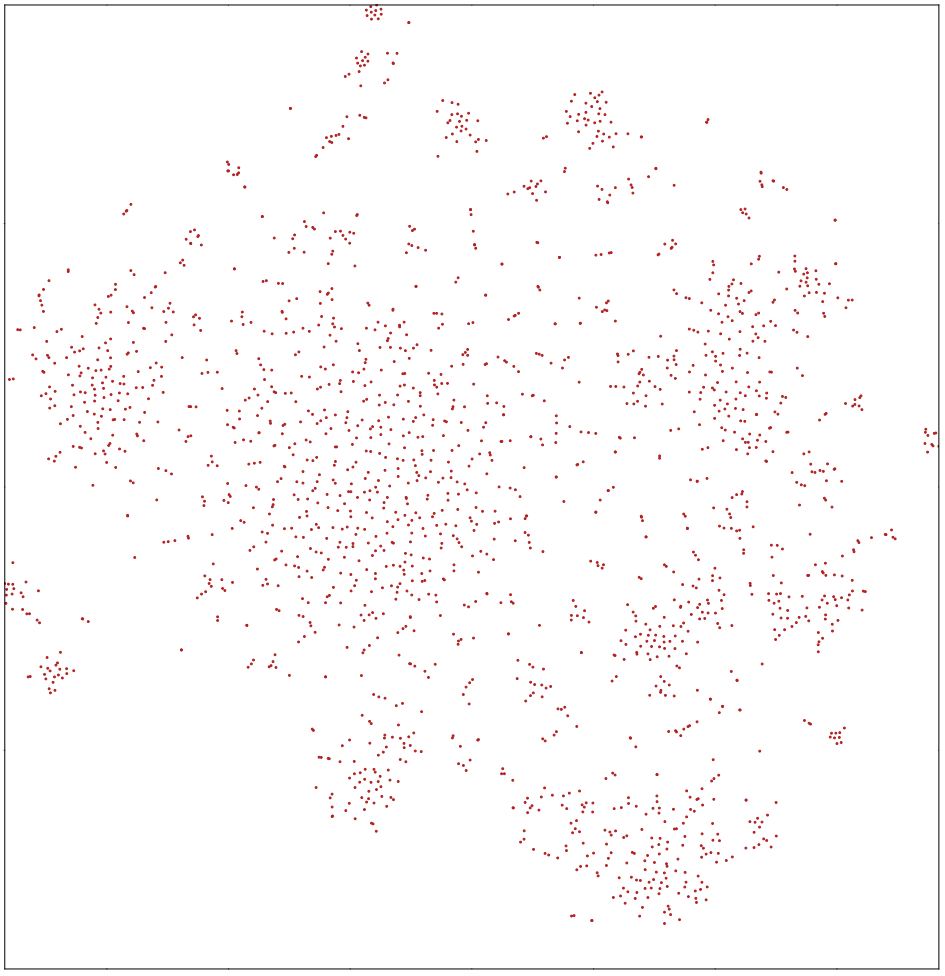

FIGURE 17: Visualisation of cwindow-50 embeddings projected down to 2D space with t-SNE.

We see that leveraging punctuation improves upon the baseline for all cases and that using both punctuation and capitalisation produces even better results for the embeddings whenever they improve upon the baseline. Unfortunately, the CharCNN embedding trained on Wiki-3 produces worse results than for Wiki-2, because the vocabulary of characters in the language model does not accept capitalised letters.

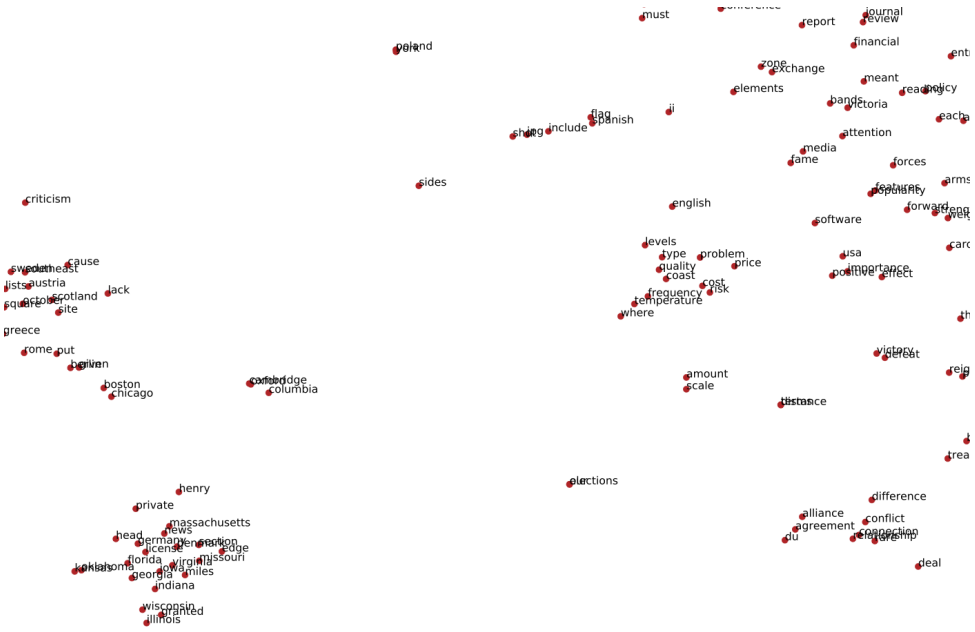

FIGURE 18: cwindow-50 projected down to 2D space with visible clusters for US locations and European locations.

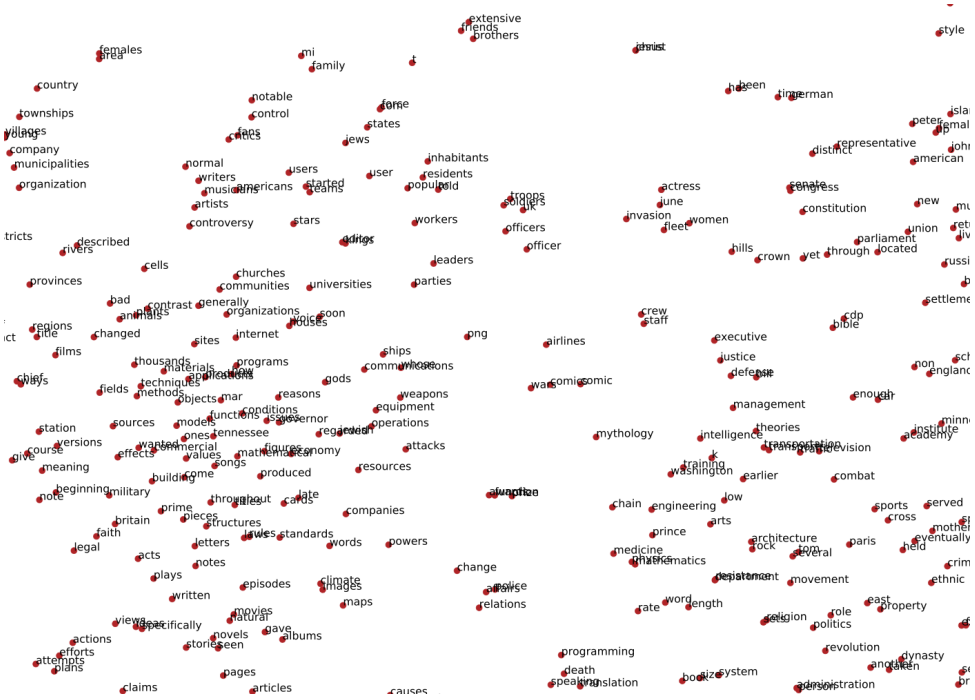

FIGURE 19: cwindow-50 projected down to 2D space with visible clusters for plays/literature and military.

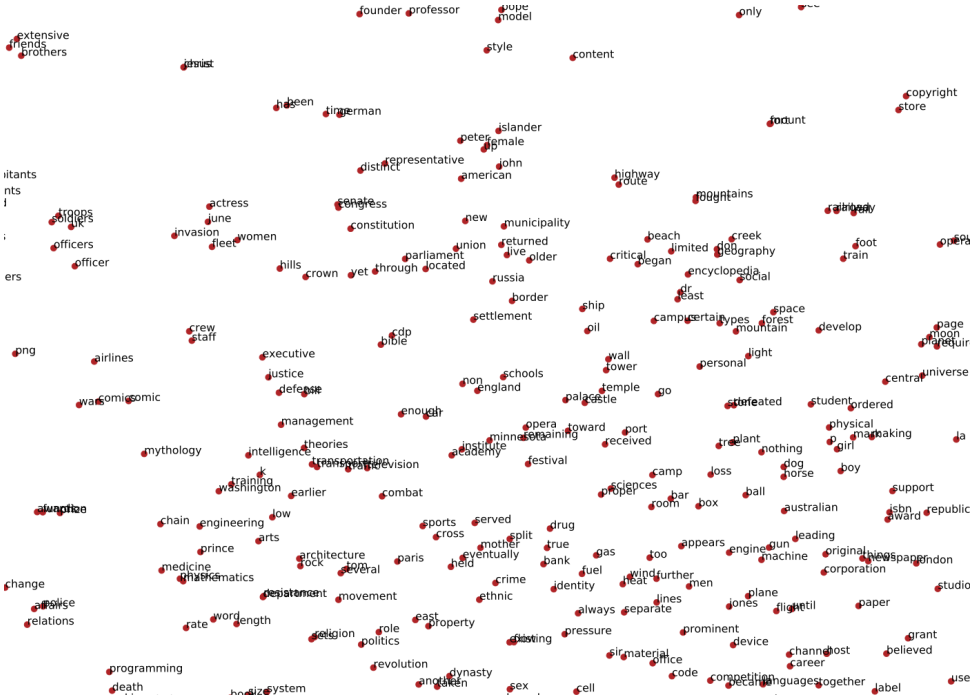

FIGURE 20: cwindow-50 projected down to 2D space showing word pairs such as boy and girl, and crew and staff.

4.5 Further experiments with word-level and character-level combined

Santos and Zadrozny (2014) combine word-level and character-level embeddings. To emulate their approach, we experiment here with combinations of CBOW-50 or cwindow-50 combined with CharCNN-Small-50, CharCNN-Small-100, and CharCNN-Small-50.

word-level	character-level	V-measure
CBOW-50	Small-50	0.5826
CBOW-50	Small-100	0.5786
CBOW-50	Large-100	0.5645
cwindow-50	Small-50	**0.5876**
cwindow-50	Small-100	0.5682
cwindow-50	Large-100	0.5440

TABLE 4.13: V-measures for combinations of word-level and character-level embeddings on the GHMM with 45 components.

Table 4.13 shows the six combinations and their performance on the part-of-speech tagging task. These are definite improvements over using the character-level embeddings alone, whose values can be revisited in Table 4.5. Table 4.4 contains the V-measures for CBOW-50 and cwindow-50, which are 0.5496 and 0.5822 respectively. Combining CBOW with any character-level embedding gives significant improvement. However, cwindow only slightly improves when CharCNN-Small-50 is used. For the other combinations it actually decreases the performance.

4.6 Test set performance

In this section, we take the best setups from all previous experiments and run them on the test set. These will be used on the Gaussian Hidden Markov model with 45 components, as that reflects the number of POS tags. The results can be found in Table 4.14. A close inspection of this table shows the same trends as performance on the development set. In short, SENNA embeddings produce the highest overall score, closely followed by cwindow on the Wiki-3 dataset. Again, the wang2vec embeddings outperform the word2vec embeddings. Regarding the character-level embeddings, the embeddings from the smaller language model are the best. PCA is indeed necessary to obtain competitive results and good language model performance is correlated with good POS induction performance. Combined embeddings do better than isolated character-level embeddings, however, only the combination of CBOW and CharCNN does better than CBOW alone, since the same does not hold for cwindow and CharCNN combined.

embedding type	dimension	extra features	V-measure
skipgram	100	none	0.5274
CBOW	50	none	0.5492
GloVe	20	none	0.5229
cwindow	50	none	0.5955
structured	20	none	0.5364
SENNA	50	none	**0.6506**
CharCNN-Large	100	PCA	0.5265
CharCNN-Small	100	PCA	0.5320
CharCNN-Direct	100	no PCA	0.4928
CharCNN-Large	1100	no PCA	0.5161
CharCNN-Small	525	no PCA	0.5052
CBOW	50	Wiki-3	0.6169
cwindow	50	Wiki-3	0.6382
CBOW, Small	50 + 50	combined	0.5807
cwindow, Small	50 + 50	combined	0.5870

TABLE 4.14: Test set performance as V-measures for the best of the embeddings on the GHMM with 45 components.

Chapter 5

Discussion and Conclusion

5.1 Discussion

The embeddings used in Lin et al. (2015) were skipgram and structured skipgram from Mikolov et al. (2013b) and Ling et al. (2015a). On top of these, we have included CBOW, cwindow, GloVe, SENNA, and character-level embeddings in this project. We have used these embeddings as input to a POS tagging system and we also looked at embedding quality by examining the nearest neighbours and visualising them in 2D space. We have been able expand on our embeddings by performing further experiments, including adding a standard embedding for out-of-vocabulary words, making changes to the preprocessing of the embedding training corpus, concatenating the best word-level embedding with the best character-level embedding, and picking out the best settings for a run on the test set. We believe our main contribution to the existing body of work to be a thorough comparison of word-level and character-level embeddings on a syntactical task. This section will discuss the findings by analysing the results of the experiments, criticise the approach used where appropriate, and consider limitations to the work.

5.1.1 Word-level embeddings

It has been possible to replicate the results of Lin et al. (2015) with the Gaussian Hidden Markov model on English data. Using skipgram embeddings results in a V-measure between 0.51 and 0.54, whereas using structured skipgram improves upon this with a V-measure between 0.53 and 0.57, which is consistent with the baseline model. Observing the different embeddings and dimensions, the overall findings indicate that SENNA embeddings produce the highest V-measure with an average of 0.64. Since these embeddings have been pretrained by Collobert et al. (2011) with two large data sets different from the data sets used for the other embeddings,

and the corpus has been tokenised in a different way, it is hard to isolate the cause of the good performance of these embeddings. When inspecting the embeddings file, it shows that SENNA has embeddings for sequences of non-alphabetical characters, whereas all the other embeddings do not accept non-alphabetical sequences of characters. This implies that SENNA has embeddings for punctuation marks found in PTB. SENNA also produced one of the weaker sets of nearest neighbours. On top of this, SENNA has been trained on a much larger data set, and both of these reasons may have led to the good performance of SENNA. Based on these points, SENNA is therefore excluded from further discussion.

Comparing the word-level embeddings among each other, it is observed that GloVe performs the worst, although not bad. It scores slightly lower than the baseline skipgram. The main difference between GloVe and the other embeddings is that it is based on a corpus statistics model, which takes into account global counts in a log-bilinear fashion. Because of its unimpressive performance as POS tagging input, we argue that this is due to the unsuitability of statistics-based models for syntactical information encoding. Before moving on to the word2vec extended family of embeddings, we will discuss the window context size experiment done with GloVe. Increasing the window size strongly reduces performance of GloVe. As Lin et al. (2015) already provided evidence for the same result with the word2vec embeddings, we reinforce the conclusion that small context windows are truly better for syntactical tasks.

Surprisingly, Lin et al. (2015) did not try out CBOW embeddings, but our experiments show that it does much better than skipgram. Therefore, it is not unexpected that cwindow embeddings do even better. They are supposed to be the improved version of CBOW, since they *do* encode word order and therefore provide more syntactical information. The main difference between these two models are that Ling et al. (2015a) increase the number of parameters for wang2vec in order to model word position in the context window. When comparing them side-by-side, it is shown that the extensions by Ling et al. (2015a) do better than word2vec. We can attribute this to the increased number of parameters needed to model word order and confirm our hypothesis that word order encoding is an important feature for POS tagging. The difference between skipgram and CBOW is that the former predicts the context, whereas the latter is predicted using the context. This gives us reason to believe that CBOW incorporates context information, and therefore outperforms skipgram.

5.1.2 Relation between embedding size and performance

Having established the influence of the type of embedding on its performance on POS induction, we move on to the influence of embedding size and wonder if there is a universal embedding size most suited for the task at hand. Wang2vec embeddings slightly prefer smaller embedding sizes, whereas word2vec slightly prefers larger embedding sizes. On the other hand, GloVe shows

good performance for either sizes. Taking averages across all different embeddings, we find that a dimension of 100 produces the best V-measure (0.538), and dimension 20 the worst (0.519). This contradicts Lin et al. (2015)'s work, where a 100D vector is the worst dimension. However, this can be explained that Lin et al. (2015) only looked at skipgram and structured skipgram, whereas we have compared more embeddings than just those.

5.1.3 A comparison of 45 versus 100 components

The previous paragraphs have taken the average of the results of the two GHMMs that were trained. The two different GHMMs were trained with different settings: either with 45 components, which is the exact number of tags, or with 100 components. Althogh this is double the norm, it allows for the model to label according to a "many-to-less" manner, where the hard constraints of 45 tags are relaxed. All the V-measures of all experiments have been taken and compared. The mean V-measure for GHMM-45 is 0.5238, which is lower than the mean of GHMM-100, which is 0.5332. We find that for GHMM-100 there is a trade-off between completeness for homogeneity.

5.1.4 A look at character-level embeddings

Now on to the character-level embeddings. We have trained several different language models and extracted embeddings from these, as outlined in Table 3.1. These include unedited embeddings of size 1100 (CharCNN-Large) and 525 (CharCNN-Small). The dimensionality of these embeddings have both been reduced to the set of 20, 50, 100, and 200 dimensions to match the other embedding sizes. Adjustments to the hyperparameters have been made to directly obtain embeddings of these size. Regarding the embeddings reduced with PCA, the way PCA works is that it ranks dimensions according to their variance in that dimension. It will always find the same ranking of dimension variance. This means that the 100D vectors are just the 200D vectors where the latter half of the dimensions is dropped, and so on. Generally speaking, for the embeddings of 20, 50, 100, and 200, the larger dimensionality produces better embeddings for the CharCNN embeddings, which indicates that some essential information is lost when dropping too many from the dimensions ranked by PCA. However, dimensionality reduction is necessary. Looking at the "pure" embeddings gotten directly from the LSTM model, CharCNN-Small (525D) and CharCNN-Large (1100) perform worse than their reduced counterparts. On top of that, they require more computational resources than the other embeddings due to their size. This suggests that dimensionality reduction is still an important part of distributional vector representations. One may wonder whether training large vectors and then performing dimensionality reduction on the other embeddings will also result in better vectors. Another point of interest is the performance of the directly-sized embeddings. These are all worse than their

same dimensional counterparts that are reduced with PCA. In fact, they are even the worst set of embeddings from both word- and character-level models. Actually, when examining the data on the test set perplexity of the language models that each produced these values, we see that a low perplexity is correlated with a high V-measure, for both 45 and 100 component GHMMs. This is interesting, because it potentially means that good representations for a language model are good representations in general, and perhaps even for syntactical tasks.

5.1.5 Qualitative differences between word- and character-level

Unfortunately, the character-level embeddings from the Kim et al. (2016) model did not bring a large increase in system performance. Therefore, it is interesting to look at the embeddings from a qualitative viewpoint. There is a stark contrast between the nearest neighbours of vectors from the Mikolov et al. (2013b) versus these character-level representations. The first produces good semantic representations, whereas the latter encode word structure, and really show how sophisticated they are when we query for words consisting of a lot of morphemes. Therefore, as Santos and Zadrozny (2014) have preceded us, concatenation of the best word-level embedding with the best character-level embedding may result in better performance. Section 5.1.6 will discuss the results of this experiment in particular.

5.1.6 Analysis of further experiments

Some logical extensions of the standard experiments follow after viewing the results. One of these is the problem of assigning a representation to out-of-vocabulary or low-frequency words. The current approach is to assign the zero vector, but it can be argued that a better method is to assign them the average of low-frequency words. This has been done for one type of embedding, however, the result was a slight reduction in performance. There was only enough time to attempt this for one embedding, and because of that, we cannot conclude that this OOV solution does not work.

Another extension is to tokenise the corpus in a similar fashion to the GHMM training data, Penn Treebank. Two extra versions of the data have been made, and they have shown that keeping both punctuation already existing in PTB as well as capitalisation greatly improves the performance. This is mainly due to having well-trained representations of certain punctuation marks, which otherwise would have been replaced with a zero vector. Leveraging capitalisation also gives the POS induction system more information, either on whether the word is the beginning of the sentence, or if the word is for example a proper noun or a geographical location.

Finally, we follow Santos and Zadrozny (2014) by combining word-level and character-level representations. These experiments show that this approach obtains good results on POS tagging,

better than using the character-level representations alone. For CBOW-50, we find improvement as well, with the largest difference in performance coming from adding a 50-dimensional character-level embedding. There is less of a boost for cwindow-50, where adding CharCNN-50 gives minimal improvement. There is even a decline when adding 100-dimensional character-level embeddings. We inspect the nearest neighbours for the vectors where a 100-dimensional character-level embedding is concatenated to a 50-dimensional word-level embedding. This allows us to conclude that because of the 2:1 ratio of character-level entries versus word-level entries, the character-level counterpart dominates the word-level embedding for nearest neighbour embeddings, resulting in a lower performance than usual. This is further supported by V-measures that are higher when the word-level embedding and character-level embedding are more equally represented in the combined embedding in a ratio of 1:1.

5.2 Conclusion

This study's aim was to experiment with different word embeddings as input to a unsupervised part-of-speech tagging system, which requires the embeddings to encode syntactical information. With specific interest in neural network models that are able to automatically extract useful features from training data, this study has been able to make a thorough comparison of word embeddings that would be suitable for part-of-speech tagging. For POS tagging, the word embeddings need to be able to encode relevant information, such as word context, word meaning, and morphological structure in order to disambiguate between possible POS tags. Three types of word embeddings created from neural networks were used in this project: prediction-based models, corpus statistics-based models, and character-level embeddings trained within a neural language model. This study has argued that for prediction-based and corpus statistics-based models, a small context window and context word order sensitivity are important, whereas character-level embeddings will be able to leverage the word's morphological structure.

As part of this work, several different word embeddings models have been used, including word2vec, structured word2vec, Global Vectors, SENNA, and character-level embeddings. It has been possible to replicate the baseline and produce results which surpass the baseline. The main findings include the following results. Prediction-based models where the context of a word is used to predict the middle word, such as CBOW and cwindow, are stronger than models where the opposite happens, such as (structured) skipgram. We suggest that models from the former variety are able to extract features from the context of the word for the vector representation, such as whether the target word is preceded by a determinant or an adjective, which raise the possibility that the target word is a noun. Word embeddings that are able to predict the context from the target word do well too, but less so than their counterparts. Especially the cwindow and structured skipgram models did well; since they are context word

order sensitive, they are able to extract more relevant features than the models in the word2vec family. Although character embeddings are readily capable of representing morphologically complex words and show relevant nearest neighbours, in isolation they do not obtain state-of-the-art performance. This has lead to combining the best word-level embeddings with the best character-level embeddings, which produced competitive results. This study has been able to show that combining word-level embeddings with character-level embeddings improve upon the performance of either embedding isolated. However, greater improvements have been made possible by tokenising the original embedding training data in a similar fashion to the part-of-speech tagging training data. This includes keeping words capitalised and not removing punctuation marks. Without good representations for punctuation, the model loses a lot of performance through its inability to properly represent these tokens. Modelling capitalisation improves the model significantly. This is due to capitalisation of words containing important information for POS tagging. A capitalised word either appears at the beginning of the sentence and has increased probability to be one of the part-of-speech tags that statistically appear at the beginning of the sentence a lot as well, or it is a proper noun or geographical location, and can therefore be tagged more easily. Coming back to the problem of not being able to represent some tokens, the present study has attempted to improve low frequency or out-of-vocabulary words by using an averaged vector instead of the zero vector as representation for such words. The data suggest that this does not significantly improve performance, although this study has been limited by trying this out only for a single type of embedding. Further research is needed to determine whether this holds more generally. In order to investigate the quality of the produced word embeddings the five nearest neighbours for three sets of words has been found and representative embeddings for word-level and character-level embeddings have been projected to 2D space and visualised. This has been able to provide a more in-depth understanding of both types of embeddings. Namely, word-level embeddings generally show proximity to semantically related words, whereas character-level word embeddings show proximity to similarly structured words. Especially when querying for a morpheme-rich word as “inconsiderable”, the character-level embeddings return words with similar subwords.

In short, the main contributions of this work are a thorough comparison of the quality and performance of word-level and character-level embedding for a syntactical task, an evaluation of several “external” methods of improving embeddings for unsupervised part-of-speech tagging, and a deeper understanding of how to improve representation of syntactical information in word embeddings.

5.3 Future work

Although this work presents a comprehensive study of word- and character-level embeddings for part-of-speech tagging, there are many interesting ideas which are suitable for follow-up studies. For example, this study was only able to use pre-trained SENNA embeddings of size 50 which were created in 2011. With current resources, a GPU-supported version of SENNA can be built to quickly train embeddings of several sizes and on the same training data, in order to be able to properly compare SENNA with other embeddings. The study by Lin et al. (2015) used the Gaussian Hidden Markov model and the Conditional Random Field autoencoder with Gaussian emissions. This work has only experimented with the former model, however, using the created embeddings for the CRF model is definitely something worthwhile doing. For dimensionality reduction of the character-level embeddings Principal Component Analysis was used, however, there are more methods available for doing this. Using Latent Semantic Analysis or (denoising) autoencoders are just two other ways of performing dimensionality reduction. It would be interesting to see if character-level embedding quality is strongly affected by the algorithm for dimensionality reduction used. We have found that the character-level embeddings extracted from the language model by Kim et al. (2016) are very suitable for representing morphologically complex words. This study only included English data and since English is not that morphologically complex, it will be interesting to see if character-level embeddings will increase performance for morphologically rich languages. Finally, other related work include different, novel ways of creating character-aware word-level embeddings. For example, Ling et al. (2015b) created a compositional character model that uses a bidirectional LSTM over character-level representations for producing word-level embeddings. Luong et al. (2013), on the other hand, use recursive neural networks to construct vector representations on the morpheme level. Both of these structure-aware vector representations use a completely different approach from Kim et al. (2016) and are therefore worth investigating.

Tag	Description	Examples
CC	coordinating conjunction	and, but, either
CD	cardinal number	twenty-nine, zero
DT	determiner	the, many, such, a
EX	existential *there*	there
FW	foreign word	trois, hund, fuego
IN	preposition or subordinating conjunction	among, whether, towards
JJ	adjective or ordinal numeral	third, yellow, well-done
JJR	adjective, comparative	cheaper, larger
JJS	adjective, superlative	cheapest, largest
LS	list item marker	first, A, B, C
MD	modal, auxiliary	could, should, might
NN	noun, singular or mass	humour, wind, book
NNS	noun, plural	students
NNP	proper noun, singular	Edinburgh, Volkswagen
NNPS	proper noun, plural	Americas
PDT	predeterminer	all, both, half
POS	possessive ending	's '
PRP	personal pronoun	herself, it, they
PRP$	possessive pronoun	my, their, ours
RB	adverb	physically, swiftly
RBR	adverb, comparative	greater, larger, further
RBS	adverb, superlative	bluntest, furthest, most
RP	particle	up, per, off
SYM	symbol	% &
TO	"to" as preposition or infinitive marker	to
UH	interjection	gosh, huh, anyways
VB	verb, base form	ask, give
VBD	verb, past tense	asked, gave
VBG	verb, present participle or gerund	asking, giving
VBN	verb, past participle	asked, given
VBP	verb, present tense, not 3rd person singular	ask, give
VBZ	verb, present tense, 3rd person singular	asks, gives
WDT	WH-determiner	which, that, whichever
WP	WH-pronoun	which, whom, who
WP$	WH-pronoun, possessive	whose
WRB	Wh-adverb	how, whenever, where
$	dollar	US$
“	opening quotation mark	“ ‘
”	closing quotation mark	’ ”
(	opening parenthesis	([{
)	closing parenthesis	)] }
,	comma	,
-	dash	-
.	sentence terminator	. ! ?
:	colon or ellipsis	: ; . . .

Table 1: Part-of-speech tags used in the Penn Treebank dataset.

Bibliography

Al-Rfou, R., B. Perozzi, and S. Skiena

2013. Polyglot: Distributed Word Representations for Multilingual NLP. *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, Pp. 183–192.

Ammar, W., C. Dyer, and N. A. Smith

2014. Conditional Random Field Autoencoders for Unsupervised Structured Prediction. *Advances in Neural Information Processing Systems 27*.

Andreas, J. and D. Klein

2014. How much do word embeddings encode about syntax? *ACL*, 2:822–827.

Attardi, G.

2015. DeepNL: a Deep Learning NLP pipeline. *Proceedings of NAACL-HLT*, Pp. 109–115.

Baum, L. E., T. Petrie, G. Soules, and N. Weiss

1970. A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, 41(1):164–171.

Bengio, Y., R. Ducharme, P. Vincent, and C. Janvin

2003. A Neural Probabilistic Language Model. *The Journal of Machine Learning Research*, 3:1137–1155.

Berg-kirkpatrick, T., A. Bouchard-cote, J. Denero, and D. Klein

2010. Painless unsupervised learning with features. *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, (June):582–590.

Biemann, C.

2006. Unsupervised Part-of-Speech Tagging Employing Efficient Graph Clustering. *ACL-CoLing 2006 - Student Research Workshop*, Pp. 7–12.

Brown, P. F., P. V. DeSouza, R. L. Mercer, V. J. Della Pietra, and J. C. Lai

1992. Class-Based n-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–479.

Christodoulopoulos, C., S. Goldwater, and M. Steedman

2010. Two Decades of Unsupervised POS induction: How far have we come? *2010 Conference on Empirical Methods in Natural Language Processing*, Pp. 575–584.

Clark, A.

2003. Combining distributional and morphological information for part of speech induction. *Proceedings of EACL 2003*, Pp. 59–66.

Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa

2011. Natural Language Processing (Almost) from Scratch. *The Journal of Machine Learning Research*, 12:2493–2537.

Graça, J., K. Ganchev, B. Taskar, and F. Pereira

2009. Posterior vs. Parameter Sparsity in Latent Variable Models. *NIPS - Advances in Neural Information Processing Systems*, Pp. 664–672.

Griffiths, T. L. and S. Goldwater

2007. A fully Bayesian approach to unsupervised part-of-speech tagging. *ACL'07 - 45th Annual Meeting of the Association of Computational Linguistics*, (June):744–751.

Hochreiter, S. and J. Schmidhuber

1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Johnson, M.

2007. Why doesn't EM find good HMM POS-taggers. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, (June):296–305.

Jurafsky, D. S. and J. H. Martin

2000. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.*

Kim, Y., Y. Jernite, D. Sontag, and A. M. Rush

2016. Character-Aware Neural Language Models. *AAAI.*

Levy, O., Y. Goldberg, and I. Dagan

2015. Improving Distributional Similarity with Lessons Learned from Word Embeddings. *Transactions of the Association for Computational Linguistics*, 3:211–225.

Lin, C.-C., W. Ammar, C. Dyer, and L. Levin

2015. Unsupervised POS Induction with Word Embeddings. *Human Language Technologies: The 2015 Annual Conference of the North American Chapter of the ACL*, Pp. 1311–1316.

Ling, W., C. Dyer, A. W. Black, and I. Trancoso

2015a. Two/Too Simple Adaptations of Word2Vec for Syntax Problems. *Naacl-2015*, Pp. 1393–1398.

Ling, W., T. Luis, L. Marujo, R. F. Astudillo, S. Amir, C. Dyer, A. W. Black, and I. Trancoso

2015b. Finding Function in Form: Compositional Character Models for Open Vocabulary Word Representation. *Emnlp-2015*, Pp. 1520–1530.

Luong, M.-T., R. Socher, and C. D. Manning

2013. Better Word Representations with Recursive Neural Networks for Morphology. *CoNLL-2013*, Pp. 104–113.

Manning, C. D.

2012. Part-of-Speech Tagging from 97% to 100%: Is It Time for Some Linguistics? In *Computational Linguistics and Intelligent Text Processing*, Pp. 171–189.

Mikolov, T., K. Chen, G. Corrado, and J. Dean

2013a. Distributed Representations of Words and Phrases and their Compositionality. *Advances in Neural Information Processing Systems 26*, Pp. 1–9.

Mikolov, T., G. Corrado, K. Chen, and J. Dean

2013b. Efficient Estimation of Word Representations in Vector Space. *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, Pp. 1–12.

Mnih, A. and K. Kavukcuoglu

2013. Learning word embeddings efficiently with noise-contrastive estimation. *Advances in Neural Information Processing Systems*, Pp. 2265–2273.

Pascanu, R., T. Mikolov, and Y. Bengio

2012. On the difficulty of training recurrent neural networks. *Proceedings of the 30th International Conference on Machine Learning*, 28(2):1310–1318.

Pennington, J., R. Socher, and C. D. Manning

2014. GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, Pp. 1532–1543.

Rosenberg, A. and J. Hirschberg

2007. V-measure: A conditional entropy-based external cluster evaluation measure. *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, 1(June):410–420.

Santos, C. and B. Zadrozny

2014. Learning Character-level Representations for Part-of-Speech Tagging. *Proceedings of the 31st International Conference on Machine Learning*, ICML-14(2011):1818–1826.

Van Der Maaten, L. J. P. and G. E. Hinton

2008. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605.

Řehůřek, R. and P. Sojka

2010. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Pp. 45–50.