Bachelor Degree Project

Swedish Natural Language Processing with Long Short-term Memory Neural Networks

- A Machine Learning-powered Grammar and Spell-checker for the Swedish Language

Authors:
Johan Gudmundsson
Francis Menkes
Supervisor: Johan Hagelbäck
Semester. VT 2018
Subject: Computer Science

Abstract

Natural Language Processing (NLP) is a field studying computer processing of human language. Recently, neural network language models, a subset of machine learning, have been used to great effect in this field. However, research remains focused on the English language, with few implementations in other languages of the world. This work focuses on how NLP techniques can be used for the task of grammar and spelling correction in the Swedish language, in order to investigate how language models can be applied to non-English languages. We use a controlled experiment to find the hyperparameters most suitable for grammar and spelling correction on the *Göteborgs-Posten* corpus, using a Long Short-term Memory Recurrent Neural Network. We present promising results for Swedish-specific grammar correction tasks using this kind of neural network.

Keywords: natural language processing, machine learning, long short-term memory, recurrent neural networks, grammar correction, spelling correction, Swedish language

Preface

We would like to thank everyone who has provided valuable feedback on the drafts of this thesis, including students and teachers at Linnaeus University who peer-reviewed the different chapters and the final thesis. We would also like to give a special thanks to our supervisor Johan Hagelbäck who has provided useful insight and feedback on the topic.

Contents

1 Introduction

Natural Language Processing (NLP), a field studying human-computer interaction through computer processing of human language, is a broad topic of research, having applications for linguistics, business, teaching, and other areas. NLP incorporates work in text generation and comprehension by computers, text to speech and speech to text. Work in the NLP field has recently been supercharged by the application of neural networks (NN), a subset of machine learning (ML) [1]. However, as of today, most research on the topic has been done for applications in the English language. There has not been much research done in other languages, not due to a lack of interest, but due to a lack of resources and funding from governments and organizations [2].

One of the better-known applications of AI-powered NLP today is using it to correct grammar and spelling. The foremost implementation of this today is Grammarly, an ML-based grammar checking browser extension with over 10,000,000 downloads, also available for MS Office and as a standalone application [3]. Grammarly corrects grammar and spelling as the user types, using NLP to detect errors and provide corrections or suggestions. As of today, Grammarly is only available in English, reflecting the priority status of the language in the field [4]. A similar tool for grammar and spell-checking based on ML-powered NLP techniques is Microsoft's Bing Spell Check API, which, again, is only available for English [5].

The primary problem is that the size and availability of datasets for English are far vaster than those for other languages. Using NLP methods on a language with few speakers or few written works would be next to impossible, as the quality and quantity of written output are so important. Additionally, the written output is only one aspect of the resources required for effective NLP. Tools for preparing datasets into a machine-readable format are needed in order for NNs to create full and accurate models of natural language.

These tools are readily available for English, but scarce for other languages and especially smaller languages. As an example, one of the most

famous artificial intelligences (AI), IBM's Watson, only provides NLP tools for a limited number of languages, the smallest of which (Catalan) has 4.1 million speakers [6]. Even then, only English has full support. On the other hand, as long as a language has enough written output that using NLP becomes feasible, we feel that there is substantial value to be obtained from testing and implementing NNs working in these languages.

Swedish is one of those languages, and as native speakers of Swedish, we the authors feel that we have an opportunity to make a substantial contribution to the field. The utility of providing similar tools for other languages is obvious. The learning step for every language is different; otherwise, the same algorithms used for grammar and spell checking in English could be used for other languages [7]. Therefore, the onus is on ML researchers to develop and provide the algorithms and tools for working with natural languages in a wide variety of languages.

Datasets for NLP are measured by the number of words they contain, with larger ones measuring in the billions. The largest dataset available for Swedish, the Creative Commons-licensed Swedish Culturomics Gigaword Corpus, contains one billion words extracted from a range of sources from 1950 to 2015 [8]. The Gigaword Corpus is however unsuited to our research, as we require the data to be as grammatically correct as possible. As the Corpus contains sources such as social media, this is not a guarantee. Instead, we will use the smaller *Göteborgs-Posten* (GP) dataset, collected from the Swedish newspaper of the same name.

Every dataset is different, meaning that one algorithm cannot possibly cover them all. Different algorithms have different levels of accuracy when learning the same dataset. Thus, our objective with this work is twofold: to perform experiments and benchmarks to first find the optimal hyperparameters to use in working with the GP Corpus, and second, to find the best way the dataset can be preprocessed for training a grammar and spell-checking neural network model.

1.1 Background

1.1.1 Artificial Intelligence

Artificial Intelligence (AI) is a broad science, concerned mainly with creating computer systems that mimic human thoughts, reasoning, or creativity [9]. The term Artificial Intelligence was first coined in the 1950s, which was also when early AI research began. At the time, the focus was on exploring topics surrounding problem solving and symbolic methods. In the 1970s, the US Department of Defense started performing research on the topic and completed a street mapping project [10]. In 2003, it had created an intelligent personal assistant, long before we had Siri, Google Now, and other systems of their kind.

Today, Artificial Intelligence has gained in popularity mainly because of the easy access to large data collections, more advanced algorithms and better computing power and larger storage [11]. The cloud in particular has aided in the AI "boom", with many cloud providers offering specialized plans for AI research and training. Projects such as OpenAI [12] and Google AI [13] have also helped broaden access to AI by providing education and other materials.

1.1.2 Machine Learning

Machine Learning (ML) is a subset of AI. It is based on the idea that computer systems can learn from data to identify patterns and make decisions based on those patterns with minimal human interaction. While AI is concerned with mimicking human behavior, the ML field is concerned with training systems and giving them the ability to analyze and learn from data. ML is in use today in most industries [11].

1.1.3 Deep Learning

Deep learning (DL) is a field within ML, which aims to, based on a specific set of predefined variables, train a computer system so that based on those

variables it can learn how to identify patterns and accomplish tasks such as speech recognition, image classification, translation, etc. It differs from traditional ML, where data is run through pre-defined equations and algorithms, in that it allows a system to learn on its own through pattern recognition [14].

1.1.4 Artificial Neural Network

Artificial Neural Networks (ANN), or more commonly simply Neural Networks (NN), are data models loosely based on the neuronal structure in the human brain, but on a much smaller scale. In other words, artificial neural networks are computer systems that attempt to mimic a biological network of neurons [15].

NNs use current knowledge on the behavior of neurons and how they communicate in order to replicate the learning process of the human brain, thus creating "intelligent" computer systems [16]. Much like a child learns how to tell apart dogs and cats by looking at several examples of both, an NN learns to recognize patterns by viewing thousands of examples of images, sentences, sounds, or other types of input [15].

NNs learning can be either supervised or unsupervised. In supervised learning, which is the most common form of learning, the inputs and desired outputs are provided to the NN, and it learns to predict the output from the input through an activation function. In unsupervised learning, only input data is provided. Unsupervised learning can then be used, among other uses, to separate this data into groups, for example by grouping customers by purchasing behavior [17].

1.1.4 Recurrent Neural Network

Recurrent Neural Networks (RNN) were first created in the 1980s. RNNs have an internal memory, which makes them very precise, at least in theory, in predicting what comes next. This makes them well-suited for sequential data such as texts. However, standard RNNs have inbuilt problems that sometimes cause them to either stop learning or take too long to learn [18].

Long Short-term Memory (LSTM) networks were created in the 1990s, made possible in part due to the high availability of large datasets and improved computing power. LSTM networks are extensions of RNNs, aiming to solve the issues with standard RNNs. LSTM networks are RNNs that are able to remember inputs for an extended period of time. They can read, write, and delete information from their memory, much like a computer [18].

1.1.5 Hyperparameters

Hyperparameters are variables that either determine the structure of the neural network (number of hidden layers, RNN size, etc.) or how the network is trained (learning rate, etc.). While there are ways to estimate the best values for these variables, often there is no choice but to experiment in order to find values that lead to a well-trained neural network, as every dataset and task is different [19].

The number of hidden layers determines how many layers of neurons there should be between the input and the output layer. A low number of hidden layers might cause underfitting while a high number will improve accuracy, but having too many hidden layers will not make any difference to the accuracy of the network while increasing the time it takes to train it. The dropout rate is a technique to avoid overfitting and will increase the validation accuracy, a low number will not make any difference to training, and a high number will cause under-learning. Direction tells the model if forward (direction 1) or backpropagation (direction 2) algorithms should be used [20].

Learning rate, as the name implies, is at what rate the network should learn. The learning rate (how much weights are adjusted towards the gradient) needs to be set within a specific range. If learning rate is too low, training will take a very long time. If it is too high, training may not converge (reach a reasonable level of accuracy) [21]. The number of epochs is how many epochs the training should last. The training data passes through the

network once each epoch, so the more epochs a network is trained, the more learning is reinforced. Too many epochs, however, can lead to overfitting. Finally, batch size is the number of samples passed to the network between each update of the training parameters [22].

1.1.6 Overfitting and Underfitting

Overfitting and underfitting are two common issues that can occur during the training of a network. Overfitting means that the model models the training data too well. In other words, it has seen the training data so many times that it knows it, but performs poorly when it is given new data that it has not been shown previously. Underfitting, on the other hand, means that the model cannot successfully model either the training data or generalize new data, which makes it unsuitable for its task [23].

1.1.7 Natural Language Processing

Natural Language Processing involves a number of steps that must be taken before ML algorithms can properly process a text. First of all, preprocessing must be done on the dataset in order to enhance its utility. One process used is called *lemmatization*. This process reduces a word to its base form or lemma, e.g., “car, cars, car's, cars' => car” [24]. Lemmatization is highly useful when attempting to correct grammar, as it aids in identifying the context of words and what part of speech they correspond to.

The GP Corpus is already preprocessed by adding not only lemmatization of the words in the corpus, but also what part of speech the words belong to as well as their *lemgrams*, a combination of lemma and grammatical meaning. As an example is the following sentence: “Hönan lade sina ägg i gräset” (The hen laid her eggs in the grass) [8]. After lemmatization, the sentence becomes “höna lägga sig ägg i gräs”. Each word, in addition, has a number of part of speech (POS) tags that provide further information. For example, “höna” is tagged NN for noun.

Another concept that will be important for our research is edit

distance, which is the difference between two sequences of characters [25]. For example, in “cat” vs. “car” the edit distance is 1, since only one character differs. Edit distance is of interest to us when building a spelling and grammar checker, since an edit distance of 1 often characterizes a misspelling, such as when somebody writes “thos” instead of “this”. An important consideration is that it is not always the case that an edit distance of 1 constitutes a misspelled word, as evidenced by the example of “cat” vs. “car”.

While a “dumb” spell-checker could accurately state that the word “thos” is misspelled, as it does not appear in the dictionary, taking context into consideration complicates matters. A word may exist in the dictionary, but when looking at it in context, it becomes obvious that the word is incorrect, as in the sentence “I though that he was finished.” The Distributional Hypothesis, first appearing in linguistics, states that words that appear in similar contexts have similar meanings [26]. Given that, it makes sense in NLP to group words according to the contexts they appear in, which should mean that when represented as vectors, words with a similar meaning will appear close to each other. Creating a so-called vector space model (VSM) with the word2vec predictive model is useful for learning and predicting the context in which words are meant to appear, aiding in correcting grammar and accurately identifying correctly-spelled words out of context [27].

The language model we use in this research is called sequence2sequence (seq2seq), a common model for NLP tasks such as machine translation. This model is suited for our research as correcting errors is closely related to translation. The source language in this case is grammatically incorrect Swedish, and the target language is orthographic Swedish. Rather than considering each word in isolation, a seq2seq model is trained on full sequences or sentences, hence the name. As in a word2vec model, this allows the model to learn context such as gender agreement and syntax structure [28], [29]. A seq2seq model is more relevant for our use case than a word2vec model as it can consider each sentence as a whole, and identify grammatical errors as well as spelling mistakes.

1.1.8 One-hot Vectors

One-hot vectors or one-hot encoding is a representation of categorical variables as binary vectors. This type of encoding is performed by first mapping all categorical variables to integer values. Then, each integer value is represented as a binary vector that has all zero values except the index of the integer, which is marked as 1.

One-hot encoding makes categorical data more expressive, and many ML algorithms cannot work with categorical data directly. Instead, the categories must be converted to numbers first, such as by using one-hot encoding. This is then required for all categorical data, both input and output [30].

1.2 Related work

Priya et al. propose a hybrid algorithm for spell-checking based on combining edit distance with n-grams, which group consecutive words in sets of two or three [31]. Though the report is not focused on spell-checking in the context of neural networks, the algorithm they design shows promise as it is demonstrably faster than using edit distance alone.

In [7], Hedlund et al. discuss the particular difficulties of Swedish concerning NLP. Although their research is concerned with cross-language information retrieval, a separate field, it is still relevant as it focuses on a number of language-specific problems that we will also face in our research.

Sharma and Kaushik have highlighted the importance of deep learning and neural networks for natural language processing in [32]. They argue that most of the problems found in NLP can be solved by using these machine learning techniques.

Weiss [33] has created a Deep Learning spell-checker using Python, Theano, and Keras, using a billion-word dataset that he injected with random noise [34]. Though he is mainly concerned with spelling rather than grammar, spelling is an important consideration in this study, and his work has very much in common with the context that we will be working in.

Kharazmi and Kharazmi use word2vec to assess both local and global coherence of a document, which is relevant to our research as an important feature of well-written texts is that they are locally (adjacent sentences) and globally (adjacent paragraphs or remote sentences) coherent [35]. This aids in understanding how a neural network can be trained to understand subtler aspects of grammar.

Ismail and Shahidur Rahman [36] is an example of a study concerning NLP powered by machine learning for a non-English language, in this case, Bengali. They use machine learning techniques to create word clusters using the n-gram model, in order to group similar words.

Gu and Lang [37] propose a text corrector for Chinese using a seq2seq neural network language model, achieving a good result with a special biased-decoding method. Though most research on the topic is done on NLP for English, this work and the previous study show that there is both a need and an interest for research done on other languages.

1.3 Problem formulation

We would like to contribute to research in the field of NLP from the perspective of a non-English language, specifically Swedish. Using the GP Corpus as a dataset, we want to investigate how to achieve spelling and grammar correction adapted to the Swedish language, by testing different inputs and ways to preprocess the dataset to best train a grammar and spell-checking machine learning model.

1.4 Motivation

Natural Language Processing is one of the most studied topics within AI today. Unfortunately, for many languages there are not enough substantial corpora available, often because governments do not do enough to promote language studies and make available language resources [2]. As a result, most research, implementations, and examples within Natural Language Processing are only available for the English language or have limited support for languages other than English [38]. For that reason, we want to

explore how to implement NLP in a non-English language, specifically, Swedish. Our goal is to further the field of NLP by demonstrating how a model can be trained to detect Swedish-specific grammatical errors at the same time as it corrects spelling.

Another motivation for our research is that it can be used as the basis for a browser extension or application that corrects grammar and spelling as the user types in Swedish. Creating such an application would allow our research to have applications both on a scientific and commercial level, increasing its potential value.

1.5 Objectives

O1	Prepare data from the Swedish newspaper *Göteborgs-Posten* to be used in a language model for spelling and grammar checking.
O2	Filter out usable sentences from the collected data
O3	Implement algorithms for injecting artificial noise into the data representing incorrect spelling and grammar
O4	Split the sentences into three datasets for training, validation, and testing
O5	Create LSTM model, including scripts for training, testing, and use
O6	Train the LSTM model
O7	Test the LSTM model
O8	Fine-tune the LSTM model
O9	Evaluate the training and testing results

The first objective (O1) is to download data from the source (in our case, Gothenburg University Språkbanken [39]) and preprocess it in a format that can be used as input to an NLP model. The next step (O2) is to filter the data so that only usable sentences are saved.

Algorithms must then be created (O3) that inject noise representing grammar and spelling mistakes that mimic human behavior. Ideally, this would be done by consulting datasets of common Swedish mistakes. However, as no such dataset exists, we do this artificially and randomly. The

next step (O4) is to split the collected data into training, validation, and testing sets and use the algorithms from the previous steps to artificially inject noise. In the next step (O5), we create an LSTM model and the scripts needed to train, test, and use it.

Finally, (O6-O9) the neural network needs to be trained and tested using the scripts created in the previous step, intermediate results analyzed and used to fine-tune the model, and the final results evaluated.

1.6 Scope/Limitation

Though we hope that our research will be useful for the field of NLP in general for languages outside of English, we realize that the specificity of each dataset means that, apart from the general methods, theories, and technologies used in this work, our research will only be applicable for work in Swedish and more specifically for data collected from the Swedish newspaper *Göteborgs-Posten*.

Another consideration is that we only intend to find a suitable algorithm to work with our dataset and fine-tune the hyperparameters in order to achieve a good level of accuracy. A fully-featured, completely tested application is beyond the scope of this project. We hope to expand and continue work on it in the future, if all goes well.

The size of the training set is also a limitation, as the larger a dataset is, the longer it takes to train. As we are limited in both time and hardware, we are not able to use full datasets, instead creating smaller sets of 50,000 sentences from the full data. We discuss this further in the Future Work chapter.

1.7 Target group

Our target group is primarily computer scientists working in the fields of machine learning and natural language processing. We hope to contribute to research by focusing on a non-English language and provide the means to improve existing knowledge and methods. However, since natural language processing algorithms depend heavily on the dataset and thus language being

used, our primary target group is researchers working in the field of NLP in the Swedish language.

Another target group, which if reached, could open possibilities for even more groups to be reached, is software developers who could be interested in expanding upon our research and incorporating trained grammar and spell-checking models in their applications.

1.8 Outline

The Method chapter describes the methods and data used in this thesis, as well as a discussion on reliability and validity. The Implementation chapter discusses the procedures and scripts employed our work on an abstract level and how we use them to perform data collection, data preprocessing, training, validation, and testing. The Results chapter presents the results of training our LSTM network. The Analysis chapter discusses these results and offers suggestions as to how we achieved them and how they can be improved. The Discussion chapter discusses implications and possible uses for our research. Finally, the Conclusion chapter presents our conclusions as well as future work to be done on this topic.

2 Method

The core of the work will involve building the training model to work with data collected from the Swedish newspaper *Göteborgs-Posten.* The corpus is presumably almost entirely free from errors; thus, we will need to introduce errors in order to be able to test that the neural network is capable of properly detecting and offering corrections for these errors. Thus we also need to create a noisy version of the dataset, which can be used together with the regular version in order to train the network. The model learns by mapping incorrect sentences to correct sentences, and adjusting weights when the model fails to produce the correct version of a sentence from an input with added noise.

In order to find the best starting hyperparameters to use for training the network, we will be performing a controlled experiment. Finding the best hyperparameters is often a case of trial and error [40], meaning that we will have to train the model several times with slightly different hyperparameters. The independent variables that will be changed from experiment to experiment are the following: batch size, number of hidden layers, RNN size, learning rate, dropout rate, and direction. The dependent variables, which will be used to measure the performance of each model, are the following: validation loss, typo correction accuracy, gendered article correction accuracy, *särskrivning* (split compound words) correction accuracy, total accuracy, and false positives rate. These variables will be explained in the Results section.

2.1 Approach

As we stated earlier, edit distance is a technique that can be used in order to detect spelling errors in a text by measuring the amount of characters that two words differ by. An edit distance of 1 usually indicates a misspelled word, but in many cases a misspelled word is the same as another, correctly spelled word. This means that edit distance alone is not enough to accurately correct spelling or other grammar mistakes.

Our approach will involve combining edit distance with a language

model, in order to accurately detect words that are out of place, even if they exist in the dictionary. For example, in the sentence “the cat has four wheels”, the word “car” is misspelled with an edit distance of 1 (r → t). A language model based on word vectorization would be able to detect that the word “cat” is out of place in this sentence, even if it is spelled correctly. In order to mimic misspelled words, we artificially inject noise into the dataset with an edit distance of 1.

As well as focusing on correcting typos, we have selected a couple of common grammar mistakes in Swedish, which we will train our model to recognize and correct. This is not an exhaustive list, rather, we intend this to be a starting point for future work in this context. The grammar mistakes we have chosen are *särskrivning* (separate writing) and gender agreement errors.

In Swedish, compound words formed by joining two nouns together (e.g. *kycklinglever*, “chicken liver”) should never be written separately, as this changes the meaning of the phrase (*kyckling lever*, for example, translates to “chicken is alive”). However, it is common for even native speakers to make this mistake, due in part to influences from English, which makes it an important inclusion in our model.

Swedish is a gendered language, and each noun has a gender with a corresponding article that must be used with it. The indefinite article (equivalent to “a/an” in English) can be either *en* or *ett* while the definite article (equivalent to “the”) is marked by using the suffix *-en* or *-et*. While using the wrong article is not as common among native speakers, non-native speakers often make this kind of mistake.

2.3 Reliability and Validity

A problem with the reliability of our research is that it will be difficult to test whether the software is accurately correcting spelling and grammar mistakes. For other machine learning tasks, such as image processing or machine translation, there are available algorithms or data that the model can be tested against in order to compare it to other models performing the same task. For machine translation, for example, the BLEU algorithm is used to verify the

accuracy of a model [41]. However, as far as we know, these do not exist for Swedish grammar and spell checking tasks.

Nevertheless, we can create our own testing sets by leaving aside part of the training data for later testing, in order to check how accurate the model is. However, even if we achieve very positive results, we are only showing that the model is capable of doing what we taught it to do - which is very different from being a complete spelling and grammar checker. Ideally, the accuracy of the model should be compared to the performance of a human proofreader, but we will have to leave this as future work.

With that in mind, it is beyond the scope of this work to provide a fully-tested and commercially viable application. Our testing requirements are therefore lower and much more manageable, though they remain a challenge to the reliability of our results.

2.4 Ethical Considerations

We do not believe there are any ethical considerations to take into account.

3 Implementation

Our language model and accompanying scripts are written using the Python programming language, a popular language for ML tasks. There are several frameworks available for these tasks available, including Keras [42], PyTorch [43], and TensorFlow [44]. Though any of these frameworks would serve us well in our research, we chose Google's TensorFlow, for its wide usage and the excellent availability of tutorials and quality documentation. We specifically use the GPU variant of TensorFlow to enable GPU acceleration and train the models faster.

Our code is structured as a number of scripts responsible for different tasks, from downloading and preprocessing datasets to scripts detailing the neural network and scripts for training and running it. We used the code in [45] as a starting point.

Table 3.1 shows the specifications of the computer used to train the model. This machine was the primary on which we trained the model, though we also used FloydHub for various testing purposes [46].

OS	Ubuntu 17.10
RAM	32GB
Processor	Intel i7 8700K
GPU	Asus GeForce GTX 1060 Dual OC 2xHDMI 2xDP 6GB

Table 3.1: Specifications of the computer used to train the model.

3.1 Scripts

This section details the scripts we use in our implementation, as well as a short description of what each script does.

3.1.1 Data Collection - download_data.py

This script downloads the raw data that we use to train the network. The data is in the form of sentences collected from the newspaper *Göteborgs-Posten* in

1994 and between 2001 and 2013, as well as open data collections from various Swedish publishers between 2001 and 2013. The data is downloaded from the Gothenburg University Språkbanken resource and then extracted. After preprocessing the data, which is done in the script below, the data is saved in a data folder that is later accessed by the train_lstm_model script and used to train the model.

3.1.2 Data Preprocessing - preprocess_data.py

This script first parses the XML files from the downloaded data, stores the sentences as plaintext, and finally stores the sentence data as pickle files for convenience [47]. The script then cleans the sentences by replacing different kinds of quotation marks with single quotation marks and removing unwanted special characters, allowing only the following list: `-!?/;"'%&<>.()[]{}@#:,|=*`. The script then counts the unique characters contained in the sentences and creates lookup tables in a one-hot format by converting characters to integers and storing both the characters and integers in separate lists called vocab_to_int and int_to_vocab. Finally, it chooses good sentences by only saving sentences that are between 10 and 92 characters in length, that terminate in a full stop, question mark, or exclamation mark, and that contain at least five words. Finally, the script stores the usable sentences in a pickle file for convenience.

3.1.3 lookup.py

A tiny library for converting text to integers using the vocab_to_int lookup table.

3.1.4 Artificially Inject Noise - noise_maker.py

This script contains three functions used to artificially introduce spelling and grammar mistakes in the text.

The first function injects artificial noise into the word vectors in integer form. It does so by iterating over each character representation in a sentence and, with a probability decided by a configurable parameter named threshold, randomly inserts an error. Errors take three forms, each of which is

equally likely: a character can either be inserted or be removed, or two characters can swap location. There are two versions of this function: in the newer version, a maximum of one error can be introduced per word. Our results contain models trained with both versions of the function.

The second function artificially introduces *särskrivningar*, compound words written separately, in the text by simply adding a space after a word if it matches any of the 1000 most common indefinite singular nouns in the dataset. As an example, the Swedish word *husvagn* (trailer) would be changed to the grammatically incorrect *hus vagn* (house carriage).

The third and final function changes the gendered articles *en* and *ett* (a) into their opposite. For example, *en hund* (a dog) would be changed to the grammatically incorrect *ett hund*.

3.1.5 Create training, validation and testing sets - create_training_sets.py

This script splits the sentences into a training set, a validation set, and three testing sets, then uses noise_maker to inject noise into sentences randomly. The three different testing sets are used for typos, *särskrivning* errors, and gendered article errors. The script creates twenty sets in total. The first ten are clean and noisy versions of training, validation, and three testing sets using all the proper sentences collected in preprocess_data.py. The other ten are smaller versions of the same ten sets. We use the smaller sets to train the model due to the aforementioned time and hardware constraints. The smaller training sets contain 50,000 sentences, while the other sets all contain 9,000 sentences each.

The model is trained on the training set, while the validation set consists of data that the model does not see during training. It is used to ensure that the model has not become overfit, meaning that it has learnt the training set very well, but fails when presented with other data. The testing set is used to measure the accuracy of the dataset once training is complete, and consists of data the model has not encountered during either training or validation [47].

3.1.6 LSTM Model - lstm_model.py

This script contains an abstract implementation of an LSTM model using TensorFlow. This file is the core of the model, defining the different encoding and decoding layers of the model as well as defining the different types of inputs that the model can take. In the model, we use seq2seq with an attention wrapper. The model contains a training decoding layer and an inference decoding layer, which create training and inference logits respectively. A final decoding layer creates the decoding cell and the attention wrapper.

3.1.7 Train LSTM Model - train_lstm_model.py

This script trains the model, by running training data through the model over a predetermined number of epochs. Though the main logic of the network is written in lstm_model, the number of epochs, batch size, number of hidden layers and size of the RNN, learning rate, dropout, and other vital hyperparameters are loaded here from an external YAML config file. From these hyperparameters, a graph is created using the lstm_model script. The training then starts, using the training and validation sets created earlier.

While training, the script produces several logs and saves two different checkpoints. The logs are used to give an overview of the training and provides data for our controlled experiments. The checkpoints save the state of the model. In case the program crashes or has to be paused, training can continue from the first checkpoint, which functions as a backup. The second checkpoint is intended to be used to run and test the model later. It is only saved once the validation loss of the model has reached a new lowest point. By running validation on the model using the validation dataset, which the model does not see during training, we can ensure that the model has not learned the training data too well. We save the checkpoint only when the model has reached a new lowest validation loss, rather than at the end of every epoch. This is called early stopping and is a proven technique for avoiding overfitting [48], [49].

3.1.8 Custom Use of LSTM Model - use_lstm_model.py

This script restores the second checkpoint and allows the model to be used with custom data.

3.1.9 Test LSTM Model - test_lstm_model.py

This script restores the second checkpoint and allows the model to be tested using the testing set created earlier. The script runs the testing set through the LSTM network and calculates accuracy for typos, gender, split compounds, and the total accuracy.

3.1.10 Run LSTM Network - run_network.py

Finally, while each script can be run on its own, this script runs the whole process from beginning to end.

4 Results

As we mention above, the neural network has a number of hyperparameters that can be changed in order to alter the effectiveness of training. If certain hyperparameters are too low or too high, the neural network will not produce the desired result. The relevant hyperparameters are number of epochs, batch size, number of hidden layers and size of the RNN, learning rate, and dropout. Though the number of epochs was set at 10000 each time, we set a stop whereby the model would stop training if the model ran for ten epochs without achieving a new low in the validation loss. The full results contain the number of epochs the model ran before aborting.

Loss is the output of the loss function, which calculates the error of a single training example [50]. The loss we present in our results is the average loss for a whole epoch. Accuracy, on the other hand, is the percentage of testing sentences that were 100% correctly predicted. In order to measure how well the model performs with the different errors we introduced into the dataset, we calculate four different levels of accuracy: typo correction accuracy, *särskrivning* (split compound words) correction accuracy, gender agreement correction accuracy, and average accuracy.

Another consideration is the dataset used. We use the same dataset to train the neural network each time; however, in different runs we preprocess the data differently. We did this in order to explore the best way that the dataset can be prepared in order for the network to learn the most common grammar and spelling mistakes in Swedish. Later runs are trained with a script that adds a maximum of one error per word, while earlier runs have no such restriction. For more information, see the Implementation chapter.

Table 4.1 presents an abridged version of the results before limiting errors per word. Table 4.2 presents results obtained after limiting errors per word. For full results, including the preprocessing method used for each, see the appendix.

Key

- *b* = batch size
- *h* = number of hidden layers
- *s* = RNN size
- *lr* = learning rate
- *d* = dropout
- *l* = loss
- *a* = average accuracy, calculated as (*tA* + *sA* + *gA*) / 3
- *tA* = typo accuracy
- *sA* = *särskrivning* accuracy
- *gA* = gender accuracy

b	h	s	lr	d	l	a	tA	gA	sA
16	8	512	0.0005	0.3	0.0583	49.66%	6.98%	69.37%	73.00%
16	4	512	0.0005	0.3	0.0629	48.80%	6.66%	66.6%	73.14%
16	4	512	0.00001	0.3	0.0455	41.23%	3.62%	45.32%	74.74%
32	4	512	0.0005	0.3	0.0379	53.41%	8.34%	73.09%	78.81%
16	8	512	0.0005	0.3	0.0769	35.66%	4.10%	43.82%	59.07%
32	4	256	0.0005	0.3	0.0329	53.08%	7.76%	73.00%	78.49%
32	4	512	0.0005	0.3	0.0451	56.43%	7.42%	74.11%	87.76%

Table 4.1: Results of models trained with different hyperparameters.

Key

- *b* = batch size
- *h* = number of hidden layers
- *s* = RNN size
- *lr* = learning rate
- *d* = dropout
- *l* = loss
- *a* = average accuracy, calculated as (*tA* + *sA* + *gA*) / 3
- *tA* = typo accuracy
- *sA* = *särskrivning* accuracy
- *gA* = gender accuracy

b	h	s	lr	d	l	a	tA	gA	sA
32	4	512	0.0005	0.3	0.0361	56.14%	9.60%	71.71%	87.10%
64	4	512	0.0005	0.3	0.0259	58.71%	12.52%	77.22%	86.38%
128	4	512	0.0005	0.3	0.0762	40.51%	3.26%	52.23%	66.04%
64	4	512	0.0005	0.4	0.0307	61.50%	14.39%	81.3%	88.9%
64	4	512	0.0005	0.5	0.0290	59.61%	14.56%	74.93%	89.36%
64	4	512	0.0005	0.45	0.0227	62.52%	14.6%	82.78%	90.21%

Table 4.2: Results after improving preprocessing.

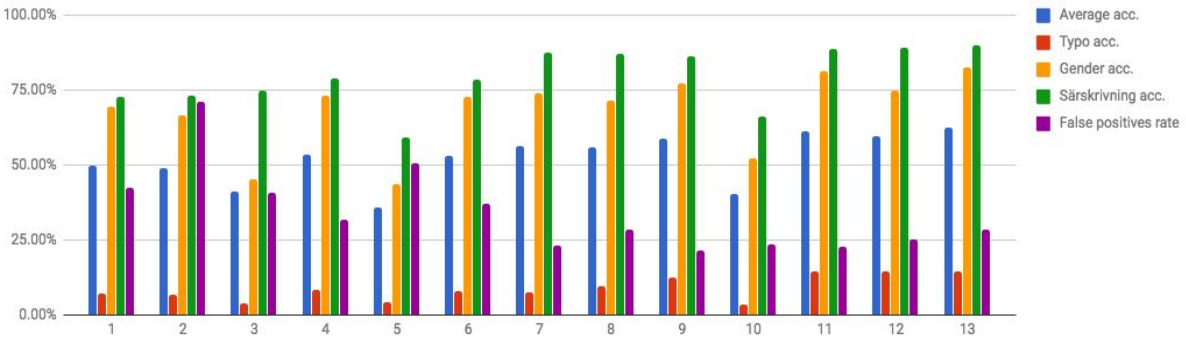

Image 4.1: Column chart of results.

5 Analysis

The dominant trend in the results was that while the loss is low, accuracy is also very low. This may be due to a number of reasons, including that we were not able to train the model for as long or as many times as we desired due to time and resource constraints. We discuss other reasons, and possible solutions, in the following chapter. However, a number of important observations can still be made from our results.

The first result that should be noted is that increasing the batch size seems to have made a substantial difference in the lowest achieved loss and highest accuracy. However, increasing batch size to 128 had a detrimental effect, with the best result achieved at 64. Increasing the number of hidden layers in the neural network, on the other hand, does not seem to have had an impact on the results. This is because a model can only utilize a certain number of hidden layers efficiently, and increasing the number of layers past that point increases the length of time needed to train the model without improving performance [51]. It is difficult to estimate the correct number to use, but we appear to achieve our best results with 4.

While our model shows good results for detecting *särskrivning* and incorrectly used gendered articles, accuracy scores for detecting typos were very low. The low loss achieved by our model compared to the high accuracy does show, however, that while the model was good at detecting and correcting many different types of typos, it performed much worse at detecting every single error in a sentence. Another consideration is that while sentences with either *särskrivning* or gender errors generally only had one error per sentence, a sentence with typos could have as many as one per word, decreasing accuracy while loss stayed low.

One reason for the amount of typos in the sentences is that in the first few runs, despite the error threshold being very high (meaning that few errors were inserted overall), many words were rendered unrecognizable due to several errors being applied to the same word. Here is an example of a sentence with too much noise inserted: “då slapp han att io appxporo vände sig åome ilifqtkön , resnerade hna .” Even to Swedish speakers, the sentence

is almost entirely unintelligible.

In subsequent runs, we changed the preprocessing so that words could only have a maximum of one typo in each, which improved typo accuracy. The same sentence as above with the new preprocessing becomes: “då slapp ha natt tiop appor vände sfig om iliftkön , reosnerade han .” While still quite noisy, a Swedish speaker can now understand it.

Though accuracy remained low after correcting preprocessing, subjective analysis of the testing results show another reason for this:

```
Testing Input: mern deet ahr int majoriteetn brytot si om ,
säger roberst bull .
Testing Output: men det har inte majoriteten brytt sig om ,
säger roberst bull .
```

In the above sentence, noise has been injected into in eight out of eleven words, with the network catching and correcting all the typos but one. However, the model considers this a failed test from an accuracy standpoint, as the proper noun “Robert” has not been corrected. Generally, uncommon words such as proper nouns are filtered out in seq2seq networks, as they can confuse the model. A more representative accuracy score could have been obtained by giving partial credit to sentences such as the above, with only one error, but this was not within the scope of our research.

Finally, it appears our model has at least learned some measure of context, as it is good at correcting typos where the first or last letter has been moved to another word:

```
Testing Input: hur skulzle du beoskriv adin musxik ?
Testing Output: hur skulle du beskriva din musik ?
```

In this correct example, the model was able to move the “a” from “adin” to where it belonged, at the end of “beskriva”. This shows that the model is able to focus on more than one word at time.

6 Discussion and Conclusion

While our findings are very preliminary, and more work and time must be spent on improving results, our findings illustrate both some of the problems and possible solutions facing AI-powered natural language processing in non-English language today.

While our results show that today's datasets are large and complete enough to train a model, the poor accuracy of our model is indicative of another problem. Ideally, as we have seen, we would start with a dataset of real sentences with a variety of grammatical problems, such as common spelling errors and *särskrivning*, as well as the corresponding sentences corrected by a human. As this dataset does not exist, another alternative would be to programmatically create such a dataset such as we have done, but based on real, common spelling and grammar errors in Swedish, rather than emulating typos.

Our results shows a wide gap in our model between the accuracy levels for typo correction and grammar correction. While our typo correction results are disappointingly inaccurate, we mention several reasons for this in the preceding chapter that are not related to the dataset problem. The more interesting question is rather how closely our noise injection mimics real human errors. Better datasets would not only result in a better-trained model, but also in a model that is more adept at correcting real errors that humans make, as opposed to random noise.

The model had very high accuracy when correcting *särskrivning* errors and incorrectly-used gendered articles, which is very promising. Again, we used a crude method for artificially injecting *särskrivning* errors, which led to many odd errors that humans are unlikely to make. A more refined method of generating *särskrivning* errors would be an arduous but worthwhile task. Incorporating such a method should increase the accuracy of our model even further while lowering the rate of false positives. However, the fact that the model did so well at correcting Swedish-specific mistakes is very promising, as it means that Swedish grammar and spelling correction can soon be on par with the same task in English.

Moreover, the fact that we were able to achieve subjectively impressive results within our limited timeframe and budget shows that the field of machine learning is more open than ever to those that wish to enter it. This also means that there is a more pressing need than ever to create datasets for interesting fields of research, as the limiting factor is no longer the availability of powerful hardware, but the difficulty to procure good data.

Our initial problem formulation involved investigating the ideal conditions and preprocessing for working with a specific NLP task: correcting grammar and spelling in Swedish. Although the work suffered from the lack of good datasets, the fact that we were able to overcome this problem and still achieve high accuracy levels for the two Swedish-specific errors lead us to conclude that we have achieved overall success. We were able to highlight the specific problems facing the field of ML-powered NLP in non-English languages, while showing that a model can easily be trained to recognize and correct at least a subset of grammatical errors in Swedish.

6.1 Future work

The nature of our work prevented us from spending an excessive amount of time on any single training session, instead having to try a number of different setups and ways to preprocess data. In the future, with more time, we would like to continue from the results we have achieved by training the model for more extended periods of time, such as days or weeks, in order to see how results can improve. This would also necessitate using larger datasets, as the datasets that we used stopped early well before training finished in order to avoid overfitting.

As we mentioned earlier, the most pressing future work that must be done is developing improved datasets. While corpora of grammatically correct and semi-grammatically correct texts are widely available in Swedish, a dataset containing common spelling and grammar mistakes and their corrected versions would dramatically improve results.

While we used an LSTM neural network in our work, recent research shows that another type of neural network, convolutional neural networks (CNN) show good results for machine translation tasks. Though technical

difficulties prevented us from testing such a model in our work, it is another valuable line of research. Facebook has developed a promising open source toolkit for machine translation using a CNN model called fairseq, which shows promise for adaptation to grammar and spelling correction tasks [52].

Another improvement would be to use several neural networks in sequence instead of only one. A setup of this kind would involve, for example, a word2vec model to first group similar words together, then piping the results to a succession of RNNs that each focus on a particular type of grammar or spelling error. This way of training a model would ensure that it learns each specific error well before proceeding to the next.

Finally, while we focused on exploring the algorithms and hyperparameters and datasets that would fit best in grammar and spelling correction for Swedish, the final goal of our research is to lay the groundwork for a tool that corrects Swedish spelling and grammar as the user types. While unfortunately, the state of NLP in Swedish means that a production-ready version of this software in the near future is unlikely, a prototypal application using an improved version of our model would be a good starting point and an essential part of future work.

References

[1] Q. Ma, “Natural language processing with neural networks,” in *Language Engineering Conference, 2002. Proceedings* [Online]. Available: http://dx.doi.org/10.1109/lec.2002.1182290

[2] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*. 2014 [Online]. Available: http://www.nltk.org/book

[3] “Grammarly.” [Online]. Available: https://www.grammarly.com

[4] “Does Grammarly support languages other than English?,” *Grammarly Support*. [Online]. Available: http://support.grammarly.com/hc/en-us/articles/115000090971-Does-Grammarly-support-languages-other-than-English-. [Accessed: 25-Mar-2018]

[5] “Bing Spell Check API,” *Microsoft Azure*. [Online]. Available: https://azure.microsoft.com/en-us/services/cognitive-services/spell-check/. [Accessed: 05-Apr-2018]

[6] “IBM Knowledge Center.” [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SS8NLW_12.0.0/com.ibm.swg.im.infosphere.dataexpl.engine.doc/r_i18n-supported.html. [Accessed: 05-Apr-2018]

[7] T. Hedlund, A. Pirkola, and K. Järvelin, “Aspects of Swedish morphology and semantics from the perspective of mono- and cross-language information retrieval,” *Inf. Process. Manag.*, vol. 37, no. 1, pp. 147–161, 2001 [Online]. Available: http://dx.doi.org/10.1016/s0306-4573(00)00024-8

[8] S. Rødven Eide, N. Tahmasebi, and L. Borin, “The Swedish Culturomics Gigaword Corpus: A One Billion Word Swedish Reference Dataset for NLP,” 2016 [Online]. Available: http://www.ep.liu.se/ecp/126/002/ecp16126002.pdf

[9] “Machine Learning: What it is and why it matters,” *SAS*, 18-May-2018. [Online]. Available: https://www.sas.com/en_us/insights/analytics/machine-learning.html. [Accessed: 20-May-2018]

[10] W. Thompson, H. Li, and A. Bolen, “Artificial intelligence, machine learning, deep learning and beyond,” *SAS*. [Online]. Available: https://www.sas.com/en_us/insights/articles/big-data/artificial-intelligence-machine-learning-deep-learning-and-beyond.html. [Accessed: 21-May-2018]

[11] “Artificial Intelligence – What it is and why it matters,” *SAS*, 11-May-2018. [Online]. Available: https://www.sas.com/en_us/insights/analytics/what-is-artificial-intelligence.html. [Accessed: 20-May-2018]

[12] “OpenAI.” [Online]. Available: https://openai.com/. [Accessed: 20-May-2018]

[13] “Education – Google AI,” *Google AI*. [Online]. Available:

https://ai.google/education/. [Accessed: 20-May-2018]
[14] “What is deep learning?,” *SAS*, 26-Apr-2018. [Online]. Available: https://www.sas.com/en_us/insights/analytics/deep-learning.html. [Accessed: 20-May-2018]
[15] “A Basic Introduction To Neural Networks.” [Online]. Available: http://pages.cs.wisc.edu/~bolo/shipyard/neural/local.html. [Accessed: 20-May-2018]
[16] D. Beeman, “Modeling the Brain: Simplified vs. Realistic Models.” [Online]. Available: http://ecee.colorado.edu/~ecen4831/cnsweb/cns0.htm. [Accessed: 20-May-2018]
[17] J. Brownlee, “Supervised and Unsupervised Machine Learning Algorithms,” *Machine Learning Mastery*, 16-Mar-2016. [Online]. Available: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/. [Accessed: 21-May-2018]
[18] N. Donges, “Recurrent Neural Networks and LSTM,” *Towards Data Science*, 25-Feb-2018. [Online]. Available: https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5. [Accessed: 21-May-2018]
[19] “Hyperparameter optimization for Neural Networks,” *NeuPy*. [Online]. Available: http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html. [Accessed: 23-May-2018]
[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986 [Online]. Available: http://dx.doi.org/10.1038/323533a0
[21] H. Zulkifli, “Understanding Learning Rates and How It Improves Performance in Deep Learning,” *Towards Data Science*, 21-Jan-2018. [Online]. Available: https://towardsdatascience.com/understanding-learning-rates-and-how-it-improves-performance-in-deep-learning-d0d4059c1c10. [Accessed: 30-Apr-2018]
[22] P. Radhakrishnan, “What are Hyperparameters ? and How to tune the Hyperparameters in a Deep Neural Network?,” *Towards Data Science*, 09-Aug-2017. [Online]. Available: https://towardsdatascience.com/what-are-hyperparameters-and-how-to-tune-the-hyperparameters-in-a-deep-neural-network-d0604917584a. [Accessed: 21-May-2018]
[23] J. Brownlee, “Overfitting and Underfitting With Machine Learning Algorithms,” *Machine Learning Mastery*, 21-Mar-2016. [Online]. Available: https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/. [Accessed: 21-May-2018]
[24] “Stemming and lemmatization.” [Online]. Available:

https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html. [Accessed: 01-Mar-2018]

[25] K. Kukich, "Technique for automatically correcting words in text," *ACM Computing Surveys*, vol. 24, no. 4, pp. 377–439, 1992 [Online]. Available: http://dx.doi.org/10.1145/146370.146380

[26] M. Sahlgren, "The Distributional Hypothesis," *Rivista di Linguistica*, vol. 20, no. 1, pp. 33–53, 2008 [Online]. Available: http://soda.swedish-ict.se/3941/1/sahlgren.distr-hypo.pdf. [Accessed: 03-Mar-2018]

[27] "Vector Representations of Words," *TensorFlow*. [Online]. Available: https://www.tensorflow.org/tutorials/word2vec. [Accessed: 03-Mar-2018]

[28] "Neural Machine Translation (seq2seq) Tutorial," *TensorFlow*. [Online]. Available: https://www.tensorflow.org/tutorials/seq2seq. [Accessed: 23-May-2018]

[29] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," 2014 [Online]. Available: http://dx.doi.org/arXiv:1409.3215 [cs.CL]

[30] J. Brownlee, "How to One Hot Encode Sequence Data in Python," *Machine Learning Mastery*, 12-Jul-2017. [Online]. Available: https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/. [Accessed: 21-May-2018]

[31] M. Priya, R. Kalpana, and T. Srisupriya, "Hybrid optimization algorithm using N gram based edit distance," in *2017 International Conference on Communication and Signal Processing (ICCSP)*, 2017 [Online]. Available: http://dx.doi.org/10.1109/iccsp.2017.8286823

[32] A. R. Sharma and P. Kaushik, "Literature survey of statistical, deep and reinforcement learning in natural language processing," in *2017 International Conference on Computing, Communication and Automation (ICCCA)*, 2017 [Online]. Available: http://dx.doi.org/10.1109/ccaa.2017.8229841

[33] T. Weiss, "Deep Spelling: Rethinking Spelling Correction in the 21st Century," *Medium*, 22-Mar-2016. [Online]. Available: https://machinelearnings.co/deep-spelling-9ffef96a24f6. [Accessed: 14-Mar-2018]

[34] C. Chelba *et al.*, "One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling," Google, 2013 [Online]. Available: http://arxiv.org/abs/1312.3005

[35] M. A. Kharazmi and M. Z. Kharazmi, "Text coherence new method using word2vec sentence vectors and most likely n-grams," in *2017 3rd Iranian Conference on Intelligent Systems and Signal Processing (ICSPIS)*, 2017 [Online]. Available: http://dx.doi.org/10.1109/icspis.2017.8311598

[36] S. Ismail and M. Shahidur Rahman, “Bangla word clustering based on N-gram language model,” in *2014 International Conference on Electrical Engineering and Information & Communication Technology*, 2014 [Online]. Available: http://dx.doi.org/10.1109/iceeict.2014.6919083
[37] S. Gu and F. Lang, “A Chinese Text Corrector Based on Seq2Seq Model,” in *2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*, 2017 [Online]. Available: http://dx.doi.org/10.1109/cyberc.2017.82
[38] “IBM Knowledge Center.” [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SS8NLW_12.0.0/com.ibm.swg.im.infosphere.dataexpl.engine.doc/r_i18n-supported.html. [Accessed: 05-Apr-2018]
[39] “Resources,” *Språkbanken*. [Online]. Available: https://spraakbanken.gu.se/eng/resources. [Accessed: 15-Apr-2018]
[40] “Hyperparameter optimization for Neural Networks,” *NeuPy*. [Online]. Available: http://neupy.com/2016/12/17/hyperparameter_optimization_for_neural_networks.html. [Accessed: 21-May-2018]
[41] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, “BLEU: A Method for Automatic Evaluation of Machine Translation,” in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, Philadelphia, Pennsylvania, 2002, pp. 311–318 [Online]. Available: https://doi.org/10.3115/1073083.1073135
[42] “Keras Documentation.” [Online]. Available: https://keras.io. [Accessed: 13-May-2018]
[43] “PyTorch.” [Online]. Available: https://pytorch.org/. [Accessed: 13-May-2018]
[44] “TensorFlow.” [Online]. Available: https://www.tensorflow.org/. [Accessed: 13-May-2018]
[45] D. Currie, “Creating a Spell Checker with TensorFlow,” *Towards Data Science*, 18-May-2017. [Online]. Available: https://towardsdatascience.com/creating-a-spell-checker-with-tensorflow-d35b23939f60. [Accessed: 23-May-2018]
[46] “FloydHub - Deep Learning Platform - Cloud GPU.” [Online]. Available: https://www.floydhub.com. [Accessed: 15-Apr-2018]
[47] “11.1. pickle — Python object serialization — Python 2.7.15rc1 documentation.” [Online]. Available: https://docs.python.org/2/library/pickle.html. [Accessed: 15-Apr-2018]
[48] T. Shah, “About Train, Validation and Test Sets in Machine Learning,” *Towards Data Science*, 06-Dec-2017. [Online]. Available: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7.

[Accessed: 23-May-2018]
[49] C. V. Nicholson, A. Gibson, and Skymind team, “Early Stopping - Deeplearning4j: Open-source, Distributed Deep Learning for the JVM.” [Online]. Available: https://deeplearning4j.org/earlystopping. [Accessed: 23-May-2018]
[50] Mate Labs, “Everything you need to know about Neural Networks,” *Hacker Noon*, 01-Nov-2017. [Online]. Available: https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491. [Accessed: 23-May-2018]
[51] J. Heaton, “The Number of Hidden Layers,” *Heaton Research*, 01-Jun-2017. [Online]. Available: https://www.heatonresearch.com/2017/06/01/hidden-layers.html. [Accessed: 23-May-2018]
[52] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, “Convolutional Sequence to Sequence Learning,” Jul. 2017 [Online]. Available: http://dx.doi.org/arXiv:1705.03122v3 [cs.CL]

A Appendix 1

Raw data, as well as examples of sentences processed by the model in testing, can be found at the following repository:

https://github.com/JohanG2012/Swedish_NLP_LSTM_Result

The code for the model, including all the scripts mentioned in the Implementation chapter, can be found in the following repository:

https://github.com/JohanG2012/Swedish_NLP_LSTM