CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Deep learning for post-OCR error correction on Swedish texts

Deep learning for post-OCR error correction on Swedish texts

Arvid Lundberg, Mattias Torstensson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2021

Master's thesis 2021

Deep learning for post-OCR error correction on Swedish texts

Arvid Lundberg, Mattias Torstensson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2021

Deep learning for post-OCR error correction on Swedish texts
Arvid Lundberg, Mattias Torstensson

Supervisor: Dana Dannélls, CSE
Examiner: Richard Johansson, CSE

Master's Thesis 2021
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2021

Deep learning for post-OCR error correction on Swedish texts
Arvid Lundberg, Mattias Torstensson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

As society becomes increasingly digital, the need to digitize physical documents and texts also increases. The most common technology for this purpose is Optical Character Recognition (OCR). Today's OCR systems are unable to guarantee a totally accurate scan. The quality of digitization varies and is often negatively impacted by features of the source material. Post-OCR correction is often performed on the text produced by the system with the aim of correcting any errors that are present.To our knowledge, there is currently no neural machine learning based post-OCR model available for Swedish. The purpose of this thesis is to develop and train a neural machine learning post-OCR correction model on a set of digitized and OCRed Swedish newspaper texts. When developing the model we took advantage of machine translation techniques as we view the problem as translating incorrect text to correct text. Several configurations of the model were tested, and the model managed to improve the evaluation of all metrics on the withheld validation and test sets. These improvements are, however, rather small and only manage to correct certain errors while skipping many others. Additionally, the system sometimes introduces new errors. While the results show improvement, they are not entirely satisfactory and we believe that additional tuning of hyperparameters and further research into synthetic data generation could lead to better results.

Keywords: Computer Science, Thesis, Machine Learning, Neural Networks, Deep Learning, Natural Language Processing, OCR, Post-OCR, Swedish.

Acknowledgements

We would like to thank our supervisor Dana Dannélls for her help and input during the project. A acknowledgement to Språkbanken Text is also in order as without their transcriptions of the newspaper materials, this project would not have happened. Lastly we thank our examiner Richard Johansson for the guidance he has given during the project.

Arvid Lundberg, Mattias Torstensson, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1

Introduction

As the demand for digitalization of texts with the use of Optical Character Recognition (OCR) increases, so does the demand on the quality of the produced digital text [1]. Today, OCR scans are not perfect and aspects of the source text, such as what font is used, can lead to errors being made in identifying characters. A high quality of digitalized text is important as the text in many cases is processed further where corrupt data could cause issues. For instance one can easily imagine how errors in processed medical records could lead to unwanted consequences.

There are three ways to approach the task of reducing such errors [2]: Source material refinement, improvements of OCR, and/or post-OCR correction. The latter is most popularly applied since it does not require material to be re-processed in order to improve results.

The task of post-OCR correction can be viewed as having two parallel sentences such as these:

> Wi önska ej i betraktelser af denna art indraga personliga hänsyftningar.
>
> Wi önſka ej i betraktelſer af denna art indraga verſonliga hänſyftningar.

Where the former is the sentence in its correct form while the latter is the sentence reproduced by the OCR. In the reproduced sentence there are certain spelling errors such as in the word “perosonliga” which has been interpreted as “verſonliga” by the OCR. The issue is then: Can these errors be automatically corrected if we do not have access to the correct sentence?

For English and French post-OCR methods using neural network based machine learning techniques have been implemented which outperformed previous statistical models for this task [3]. No such model has as of yet been implemented for Swedish, which will be the focus of this project.

In order to train a model that is able to correct OCR errors automatically, these models need error free data. Kungliga biblioteket (KB), the National Library of Sweden and Språkbanken Text, the Swedish Language bank at the University of Gothenburg, have scanned 200 Swedish newspapers from the time span 1818 to 2018. From each newspaper, two pages have been scanned which amount to 400 pages. These pages have also been manually transcribed and constitute the foundation for this project.

1.1 Aim of Thesis

The aim of this thesis is to find out if a neural machine learning model can be trained to learn how to correct errors from data based on the data collected from the newspapers. If the model manages to learn this it should be able to correct errors given the context in which they are given. For instance if there is a reproduced word "doar", which for this example would be a misspelling of the word "roar". Without context from the rest of the sentence you might be equally correct in saying that this should be the word "boar". The main question this thesis aims to answer is: Can we develop a deep learning based system that improves the baseline quality of the OCR of scanned Swedish newspapers?

1.2 Scope and Limitations

When working with neural machine learning models a common way of achieving better results is by acquiring a large amount of data. In our case, the data is somewhat limited and we have no way of obtaining more labeled data from the source.
In addition to limitations of data there exists other challenges posed by the properties of existing data. Firstly, the data, collected from newspapers, contains large amounts of named entities. These named entities are things such as locations, people, numerical years etc. which might be difficult for a model to correct when limited to the context of our training data. Secondly, the newspapers on which the dataset is built were collected over a large time span during which the grammar and spelling of the language has shifted. We therefore select only a subset of this data that originates from after 1907 as the main focus for this project. This should make the dataset more uniform and make it easier to achieve greater results on other texts from the same time span. However, by doing this we further limit the size of our dataset and while a system trained on this uniform dataset might perform better on similar texts it will likely perform worse on data from other time-periods.
Another data property which complicates the situation is that the OCR produced text and the ground truth (GT) is only aligned at sentence level and not at word level. This makes systems which would rely on a word to word translation unusable.

1.3 Ethical Considerations

Something to consider in regards to ethics is that depending on what data is used to train these kinds of systems it is possible that certain stigmas from the training data is propagated to the corrected text in unwanted ways. One example of this could be that the model 'corrects' a sentence like "...the doctor, she..." to "...the doctor, he..." due to the historical prevalence of doctors being separated by gender combined with the similarity between the spelling of the two pronouns.
Another considerations is that while these systems are designed to increase overall correctness, it is possible that new errors are incurred. If such errors are incurred in the wrong places on sensitive data it could lead to unwanted consequences.

Because of these risks one should consider what training data has been used in relation to the material it is applied on. In a case where sensitive data is involved, one should also consider manually inspecting the corrected text to ensure its integrity.

1.4 Related Work And Contributions

While similar projects have been performed for other languages there does not appear to be one for Swedish which primarily uses a neural network based approach. Bazzo et al. [4] discuss how the topic of OCR errors is still an open one, referencing a paper by Rigaud et al. [5], that covered 10 European languages, with Swedish not being one of them. In 2019 there was a master's thesis [6] on the topic of Swedish OCR correction which used support vector machines and an algorithm based on the Levenshtein distance for error detection and correction respectively. There has also been a recent paper [7] which partially dealt with post-correcting Swedish OCR results as it had Finnish texts which include both Finnish and Swedish versions. However the system they used, which was based this upon a paper by Silfverberg et al. [8], did not use deep learning. Furthermore they state that their algorithm can not affect spaces, i.e. errors where the system misses a space or introduces a new one would not be correctable, something which could potentially be solved with a system that uses character embeddings.
Chantal et al. [3] mention how spelling error correction has been compared to translation, a field where neural networks have surpassed older statistical methods. In a master thesis by Sara Salimzadeh [9] the use of similar methods as Chantal are successfully applied to the post-correction task. Aspects of these works lie as inspiration for this project.
Since no published work using deep learning methods to do error correction on Swedish text appear to be available, we believe that this project is a first of it's kind in this specific domain.
Because of this, there is also, to our knowledge, no data set available that can be used as a point of direct comparison between this project and other projects of a similar nature. As such, in this thesis we do not compare our results to these works. An estimate of performance compared to other models might possible by manually comparing the 'baseline relative' performance difference if desired. However, this is not directly comparable as the error types and frequencies might vary between the data sets. Instead the performance will be compared to the baseline of not performing any post OCR corrections.

2
Background

This section will outline some of the concepts which constitutes the base of the work and components used in the implementation of the system.

2.1 Optical Character Recognition

Optical character recognition or OCR is a term used to describe technologies which convert handwritten or otherwise analog text into digital text. The process of OCR consists of many steps each of which has its own difficulties. The first of these is optical scanning which creates a digital image of the source material and converts it to gray scale. The digital image is then segmented into it's constituent parts such as images, text segments or isolated characters or words. Following this is a pre-processing stage, here characters are normalized and repaired. After the pre-processing, the characters are digitized through feature extraction. Lastly, a post-processing step called grouping is applied in which characters which might have been incorrectly separated are grouped.

All of these steps are prone to produce errors. If the contrast in the scanning stage is wrong or a character is too damaged to be repaired in the pre-processing it could all lead to errors.

An additional step, post-OCR, is therefore applied in order to attempt to correct these. These post-corrections might be performed by rule based or context based systems. While post-correction might lead to improvements, no system currently exist which can guarantee totally accurate OCR as shown by the ICDAR competition in 2019 where improvements but not solutions were achieved [10].

2.2 Artificial Neural Networks

Artificial neural networks (ANN) are a mathematical model which is inspired by the naturally occurring neural networks in the human brain [11]. The main component of the neural network is the neuron. The neuron receives a signal and depending on different conditions it produces its own signal which is passed along the network to other neurons whose signals are eventually interpreted as an output response to the input.

A single neuron is usually modelled as a unit consisting of an activation function f, a set of weights $\vec{w}$, and a bias b. Using these in combination with the input vector

$\vec{x}$, the neuron produces an output o:

$$o = f(\vec{x} \cdot \vec{w} + b)$$

In its simplest form the activation function is the Heaviside step function:

$$H(x) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases}$$

In this case the single neuron's output can be interpreted as a simple binary classifier with a linear decision boundary, where the neuron's weights decide the slope of a hyperplane decision boundary. In the case of a two-dimensional input this makes the decision boundary a line perpendicular to $\vec{w}$ that intercepts the x_2 axis at $\frac{b}{w_2}$ [12]. An example of this can be seen in Figure 2.1.

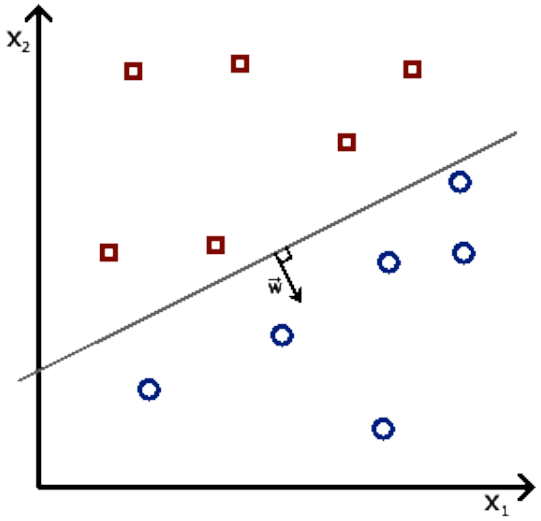


Figure 2.1: Figure of an example decision boundary, the boundary (shown in grey) separates whether the points on the input plane are classified as squares or circles.

In practice functions other than the Heaviside step function are used, with two examples of common choices being the Sigmoid function or the rectifier function $ReLU$. Note that since these are not binary functions it creates decision boundaries that are more ambiguous and makes a geometrical interpretation like Figure 2.1 significantly more difficult.
In modeling a neural network, neurons are commonly arranged as a feed forward network (illustrated in Figure 2.2) where the neurons are grouped in layers with acyclic connections in between them. These connections feed the output from a previous layer to the next where they are used as the inputs for the neurons in that layer. The signals are then propagated through the network until the final layer is reached, whose signals are instead interpreted as the output of the network. It's common to have different activation functions for the different layers, in particular the final output layer tends to be different from the rest. This allows a network to

create more complex decision boundaries than a single neuron, as well as potentially have more than one output value.
With multiple output values there is a commonly used activation function called Softmax (shown in Equation 2.1) that normalizes the outputs so that they all form a probability distribution together. For example, an ANN that tries to identify whether a sentence is in Swedish or English could have two outputs, one for each language, that then with the help of Softmax can be interpreted as the probability of the input sentence being Swedish or English respectively.

$$softmax(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} \tag{2.1}$$

Finding the correct values for the weights and biases in a network is required in order for it to deliver the desired outputs, and it is this process that is performed during the training of an artificial neural network.

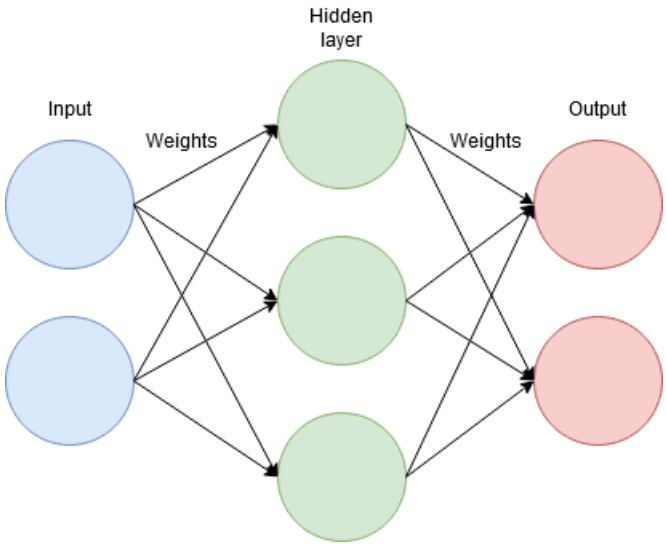


Figure 2.2: An illustration of a small neural network

2.2.1 Back-propagation

Training feed forward neural networks is done via the back-propagation algorithm. A function of the network's weights and biases called the *loss function* is created, which by construction has a global optimum where the network's parameters yield the correct outputs.
An example of such a function would be the mean squared error:

$$MSE = \frac{1}{N}\sum_{i}^{N}(y_i - o_i)^2$$

Here o_i is the network's output for the i'th training sample while y_i is its corresponding target output. It can be seen that if the outputs and targets are the same then the sum becomes zero, while at the same time if the target and outputs aren't equal then the squared term strictly increases. Thus the sum can not achieve a value

below 0 and has a global minimum when the outputs match the targets. In order to optimize a loss function, training data is fed through the network, creating the N outputs which are then used to calculate the loss. This gradient of the loss is then calculated with respect to the parameters of the network by expanding the o_i terms and using the chain rule, these gradients are then used to adjust their respective weights or biases in the direction of the optimum, generally through some variation of gradient descent.
There are multiple options when it comes to the choice of loss function and optimization method. The choice of loss function depends on what kind of output is wanted from the network.

2.2.2 Recurrent Neural Networks

Recurrent neural networks (RNN) refer to ANNs that are not necessarily arranged in the acyclic feed forward structure shown in Figure 2.2, but that instead have connections in other directions too. There are many ways to configure these connections, but of particular interest within our problem area are networks that iterate over a discrete input sequence while creating an output sequence, retaining information from previous parts of the sequence. A simple example of this would be a two neurons connected in sequence, where the first one takes its own previous output as well as the next sample from a sequence as input, a visualization of this network can be seen in Figure 2.3.

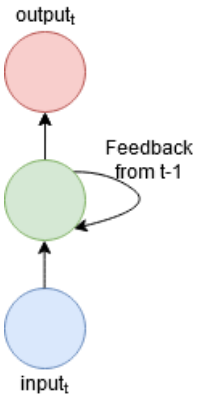


Figure 2.3: A simple recurrent neural network where the first neuron feeds back into itself, thereby forwarding data from a previous timestep.

This structure makes intuitive sense if for example one is trying to translate the sentence “not tasty” into a language where negation is done through inflection. When translating the word “tasty” one needs information from earlier in the sequence in order to know whether to translate it into its negative form or not. Similarly, other sequences such as games or movies (i.e. image sequences) rely on data from previous parts of the sequence to convey information such as motion.
Adding connections that feed back into a network in this manner complicates the learning process, a problem which can be solved by using *Backpropagation Through Time* (BTT), albeit at the cost of creating a larger network [12]. In BTT one gets

rid of the feedback connections by effectively copying the network for every part of the input sequence (or time step, hence the name), and creating connections to the next copy instead of feed back loops. An example of this can be seen in Figure 2.4.

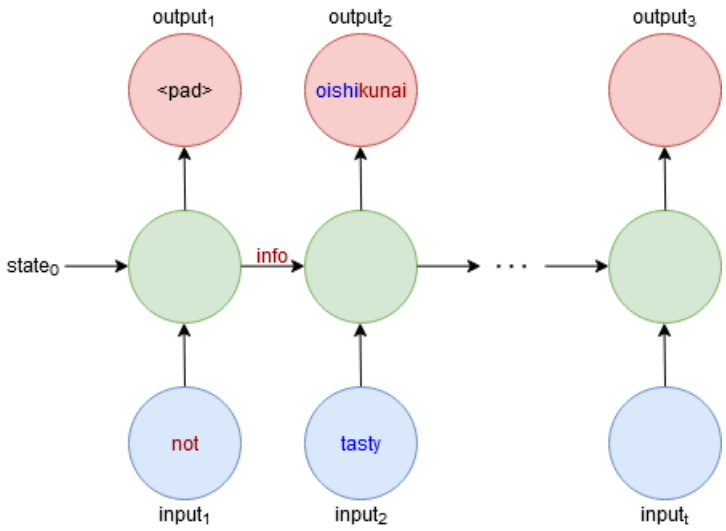


Figure 2.4: Figure showing the network from Figure 2.3 unrolled, removing the feedback loop structure. Also shown is a simple translation of the term "not tasty" from English to Japanese, where the Japanese word for tasty, "oishii", is negated by conjugating it to "oishikunai", a process requiring information from the preceding english word "not" which is passed forward through the connection between timesteps.

BTT in turn brings two new problems to light due to long sequences making the network deeper.

The first problem is that when gradients are backpropagated to update weights or biases in the network there tends to be a large amount of multiplications, and if the factors are not equal to 1 then this leads to the product changing exponentially. This leads to what is known as the *vanishing gradient problem* (or the *exploding gradient problem* if the factors are larger than 1), which slows down learning in the case of vanishing gradients or leads to NaN values for exploding gradients, which in turn breaks learning.

The second problem is that the memory required to train the network increases with sequence length, causing out of memory errors. A solution to this is shown in Section 2.3.2.

Long short term memory or LSTM is a recurrent neural network architecture which was invented in order to solve the issue of the vanishing gradient problem [13]. In this architecture the neurons consist of several *gates* and a cell state which regulates how information is passed forward.

2.3 Sequence to Sequence

One application of RNNs which is relevant to our objective is one that is used within today's state of the art translation systems, known as *sequence to sequence* (seq2seq). Seq2seq models are described as having a encoder-decoder structure [14]. In this structure an RNN acting as an encoder encodes the input sequence into a fixed-sized vector representation of its contents. The decoder, another RNN, then uses this representation to generate an output sequence [15]. In machine translation this internal representation is often referred to as 'interlingua', an abstract representation of the input sentence that is used by the decoder in order to generate the output sentence [16].
When working with sequences it is natural to use RNNs to model the encoder and decoder. Figure 2.5 shows an illustration of a seq2seq model. The encoder RNN sequentially receives the tokens in the input sequence while updating its hidden state. A token is an element of a sequence. In the case of natural language these are commonly characters or words. The hidden state that is produced after the encoder receives the final token of the input, the *end of sequence* token (<eos>), will be the vector representation of the input. This vector will then be set as the initial hidden state of the decoder and together with a <start>token, the decoder will iteratively produce the tokens of the output sequence. These tokens are used as inputs in the next time-step unless an <eos>token is produced in which case the generation is finished.

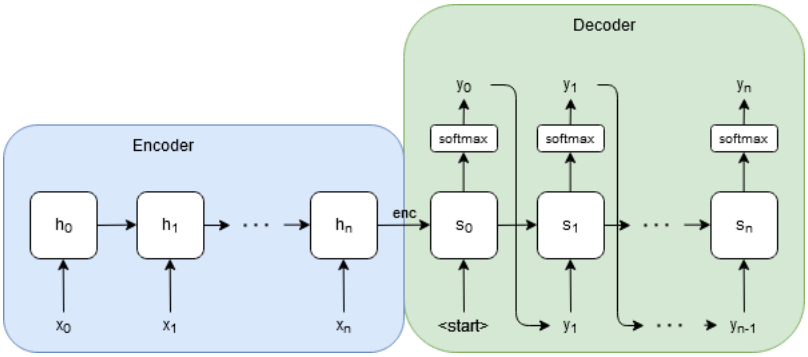


Figure 2.5: A diagram of a sequence to sequence model where the encoder and decoder are RNNs with hidden states h_i. The final hidden state *enc* of the encoder is used as initial hidden state in the decoder. The output from the decoder RNN is passed through a softmax layer in order to produce a probability distribution over the vocabulary from which the output character is picked. The final token of the input sequence x_n and the final output from the decoder y_n is the <eos>token.

2.3.1 Attention

A mechanism known as *attention* can be added as an extension to the seq2seq model [17]. The addition of this mechanism has shown to widely improve results in machine translation systems [18, 19]. There are two primary points which are changed with the inclusion of attention.

First, instead of using the final hidden state of an RNN as the encoding, the encoder RNN is replaced by a bidirectional RNN and each hidden state is used to encode the input. This effectively gives each token in the input its own encoding that also contains information about the rest of the input. To note here is that the hidden state of the encoder as depicted in Figure 2.6 now consists of two parts, one part for each direction. The arrows leading from h_i to the feed forward ANN represents the entire hidden state $(\overrightarrow{h_i}, \overleftarrow{h_i})$.
Second, an attention mechanism is added at the start of the decoder in the form of a feed forward ANN. This part of the network creates weights that represent the relative importance of each individual encoded input with respect to the previous state of the decoder. These weights are then multiplied by their respective encoded values and summed to form a new representation which the RNN part of the decoder uses in order to generate the next token, this is generally refered to as Bahdanau attention (based on [17]) as there are other ways of implementing attention systems. While weights are calculated for every character in the input, paying attention to the padding tokens is not desired, so a large constant value is subtracted from their weights before they are normalized with a Softmax layer.

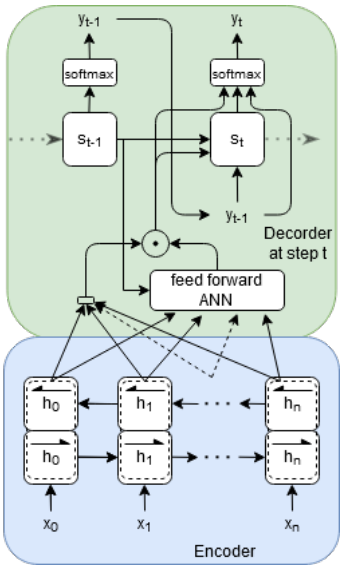


Figure 2.6: An illustration of the decoder step at time t. The hidden states from the encoder gets fed into a feed forward ANN which calculates what weighting these should get when generating token t. These weights are then multiplied by their respective hidden state after which they together with the hidden state s_{t-1} used to calculate state t as well as generate output y_{t-1}.

2.3.2 Truncated Backpropagation Through Time

Training RNNs with backpropagation through time comes with a caveat of having to unroll the network [12], or when using Pytorch, storing all the computations that have been done to reach a certain output [20]. This leads to storing every single character's attention weight calculation at every time step. This in turn causes VRAM usage to rapidly increase as model sizes and sequence lengths increase. To

alleviate this problem a method called *Truncated Backpropagation Through Time (TBPTT)* was used.
The algorithm is described as follows by Sutskever in their thesis *Training Recurrent Neural Networks* [21]:

> "It processes the sequence one timestep at a time, and every k_1 timesteps, it runs BPTT for k_2 timesteps, so a parameter update can be cheap if k_2 is small."

Instead of performing the weight update of the entire network at the end of the sequence, it is instead performed periodically throughout the sequence. TBPTT does however introduce bias to the gradients used during back propagation and removes theoretical convergence guarantees [22].

2.3.3 Teacher Forcing

When training an RNN as described in Section 2.3 outputs are fed forward and used as the input to the following step. An issue arises in this approach if the untrained network outputs the wrong character during a time-step. This unexpected incorrect token is then used in the next time step and can cause cascading errors leading to an output that becomes nonsensical. While the training of the network should eventually correct these errors, it can delay convergence.
Teacher forcing is a method that attempts to circumvent this issue. In this method the input to the next step is instead set to be the expected output from the previous step. Training in this manner allows each time-step to produce outputs as if all previous outputs were correct, even though they might not have been.
While this method can help accelerate convergence, it might also cause an issue known as exposure bias [23, 24]. This issue is due to the fact that the network has only seen the correct inputs during training which cannot be guaranteed during test stages. This would therefore cause the network to perform poorly in the case of an incorrect output. The effect of this issue is, however, debated [25].
This issue can be mitigated by not always using teacher forcing during training, instead randomly sampling from either the model's previous output or the expected previous output. This is often done in a manner where the probability starts of favoring the expected and then transitions to favoring the model's output [26].

2.3.4 Decoding

As mentioned in Section 2.3 the seq2seq model outputs a sequence, in our case of characters. In order to choose a token the decoder outputs the likelihood of each token in the vocabulary through the use of a softmax layer at each time step.
Given this likelihood, the decision of which token to output is to be made. During training a simple and common method is to greedily select the character which corresponds to the highest likelihood at the given time step. While this method might not produce the most optimal outputs and has some characteristic problems, the quality of the output is sufficient for training.
A problem with this greedy strategy originates from the overarching goal of the system, which is to produce the sequence of outputs with the highest probability.

More specifically, the greedy strategy runs the risk of selecting a token early in the decoding which makes the completed sequence have a lower total probability than if another token had been selected at that step [27]. One could consider keeping track of all possible sequence permutations during decoding so that the one with the highest likelihood can be selected. While this strategy is theoretically sound, it is in practice infeasible because of the enormous amount of combinations and the required computational resources required. Instead an approximation of this method, called beam search, is employed during validation and testing since higher decoding performance is preferred during these.

2.3.5 Beam Search

The beam search strategy is a strategy that approximates the method of keeping track of every output permutations likelihood [15, 28, 29]. This is done via a *beam*. The beam keeps track of the k most likely output sequences at each time step, where k is a user defined parameter. The network is then ran for each of these k *hypotheses* from which a new set of k softmax outputs are produced and this process is repeated. This is illustrated in Figure 2.7 for a couple of time steps where k is set to be three. Observe that the three selected sequences are the ones which have the highest likelihood from all three softmaxes that result from the previous k most likely hypotheses combined. The decoding continues in this fashion until a sequence ending with an end of sequence token has the highest probability or until a max length has been reached.

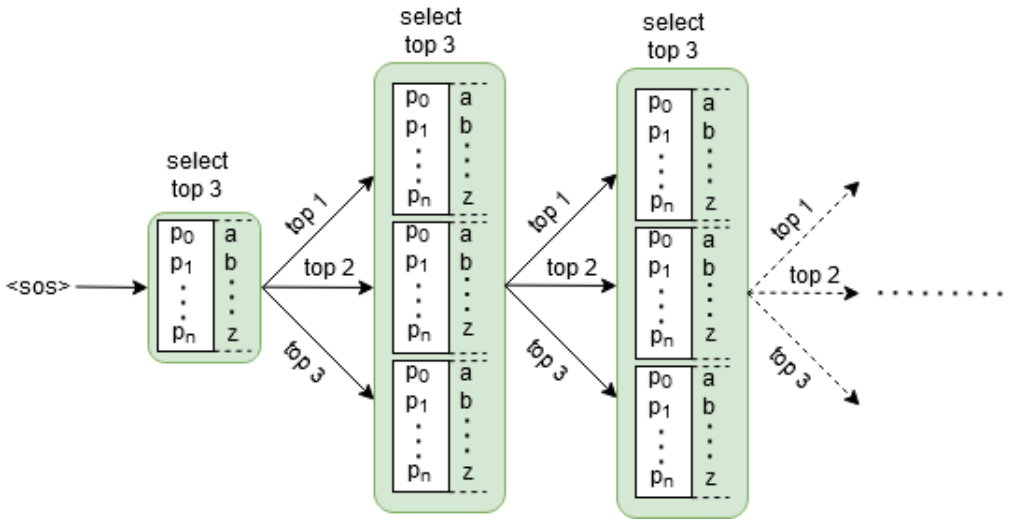


Figure 2.7: An unrolling of a beam over time. In this example k is three. In each time step the softmax for each hypotheses is produced and the next k hypotheses are selected from the combination of these.

2.4 Evaluation Methods

To evaluate the performance of the model the dataset is split into three parts, a training, a validation, and a test set. During training only the training set is used

to actually train the network through backpropagation. The validation set is used during training to gauge its performance on unseen data, in particular when selecting hyperparameters. Lastly the test set is used to gauge the performance of the final models, as its data has been withheld throughout the entire training process.
Since manual evaluation of our models would be far too time-consuming and expensive, automatic methods are used. The methods employed here are ones that are frequently used in related work [10], namely:
Word error rate (WER) is a measure of how many words have been inserted, deleted or substituted in the produced text. WER is calculates as follows,

$$\frac{I + D + S}{N} \tag{2.2}$$

where I is the number of incorrectly inserted words, D is the number of words missing in the hypotheses, S is the number of words which have been substituted between hypotheses and the reference and N is the total number of words in the reference text [30].
Character error rate (CER) is a measure of how many characters have been inserted, deleted or substituted in the produced text. This metric is calculated using the same equation as WER (2.1), but in this case, I is the number of incorrectly inserted characters, D s the number of missing characters, S is the number of characters which has been substituted and N is the total number of characters in the reference [31].
Precision is the ratio of words in the output which is also present in the reference. Precision is calculated as $P = O/N$ where O is the how many of the words in the output are also present in the reference text, and N is the number of words in the output.
Recall is the ratio of words in the reference which is also present in the output. Recall is calculated as $R = T/M$ where T is the number of words in the reference which has also been accounted for in the output and M is the number of words in the reference.
F-score is the harmonic mean between recall and precision. F-score is calculated as

$$2 * \frac{P * R}{P + R} \tag{2.3}$$

where P and R is precision and recall respectively [16, 32].
Since the goal of our system is to reduce the number of errors in the final text we naturally want to see a reduction in WER and CER, and an increase of the F-score and its parts.

3

Data

Training data is needed in order to train neural networks for correcting text. For this project, a set of OCR-produced texts together with its corresponding manually transcribed *ground truth* (GT) are used for this purpose. The OCR-produced texts are scans of different *frames* in the newspapers which were used to create this dataset. Unfortunately the source images are under copyright licence and can therefore not be published. The scanned frames are of varying size. As such, the scanned sequences are also of varying length. The scanning of the source material was done three times, each time using different systems. The systems used were Abbyy FineReader, version 11.1.16, and the open source system Tesseract, version 4, as well as a weighted combination of the two called AbbyyTesseract [33, 34]. Therefore each sample contains the OCR-produced text output from each of these systems along with its GT. Errors produced by the different systems are different. As such, a model trained on errors output from one system cannot necessarily be expected to perform well in correcting errors from another system.

It is worth noting that factors relating to the quality of the source material such as paper quality, ink or lighting might affect the output from the OCR, and that these factors are not taken into consideration for in this project.

3.1 Data Analysis

3.1.1 Text Lengths

A brief analysis of the length in characters of each text in the dataset was performed, both for the transcribed ground truths as well as the three OCR types. Histograms of the amount of characters per text can be seen in Figure 3.1, note that the vertical axis has a logarithmic scale. It can be seen that all four of the text sets have similar distributions and that they all exhibit a rapid decline in frequency as the length of the file increases.

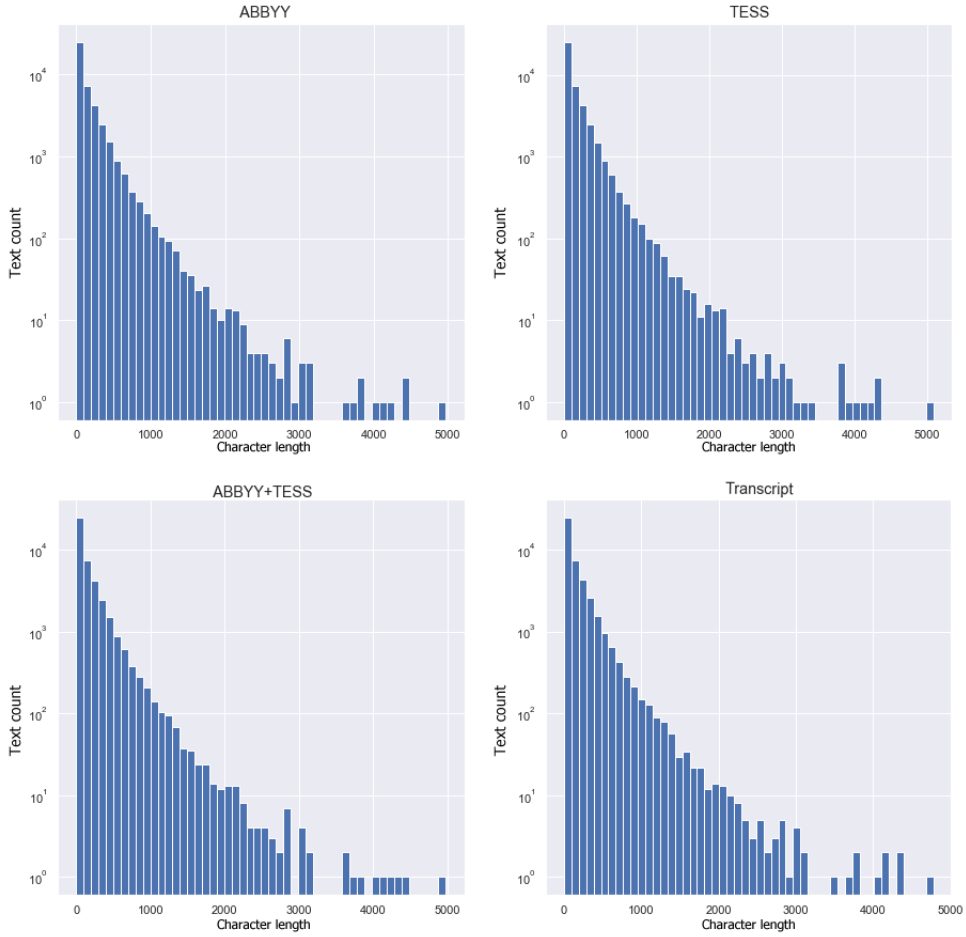


Figure 3.1: Histograms of the amount of characters per text file in the datasets in logarithmic scale.

The means, medians, and total character counts were also calculated and can be seen in Table 3.1. The median and mean for the transcript, i.e. the ground truth, is slightly smaller than the OCR results, potentially indicating that the OCR is more likely to generate characters than miss them.

Table 3.1: Metadata of the different text-sets

Text	Mean	Median	Total
Abbyy	158.20	67	6932505
Tess	158.85	67	6961102
Abbyy+Tess	157.96	66	6922088
Transcript	157.19	65	6888886

A similar analysis has been performed on the difference in length between the transcribed and the scanned text, that is, the transcript's count minus the OCR count. These can be seen in Figure 3.2.

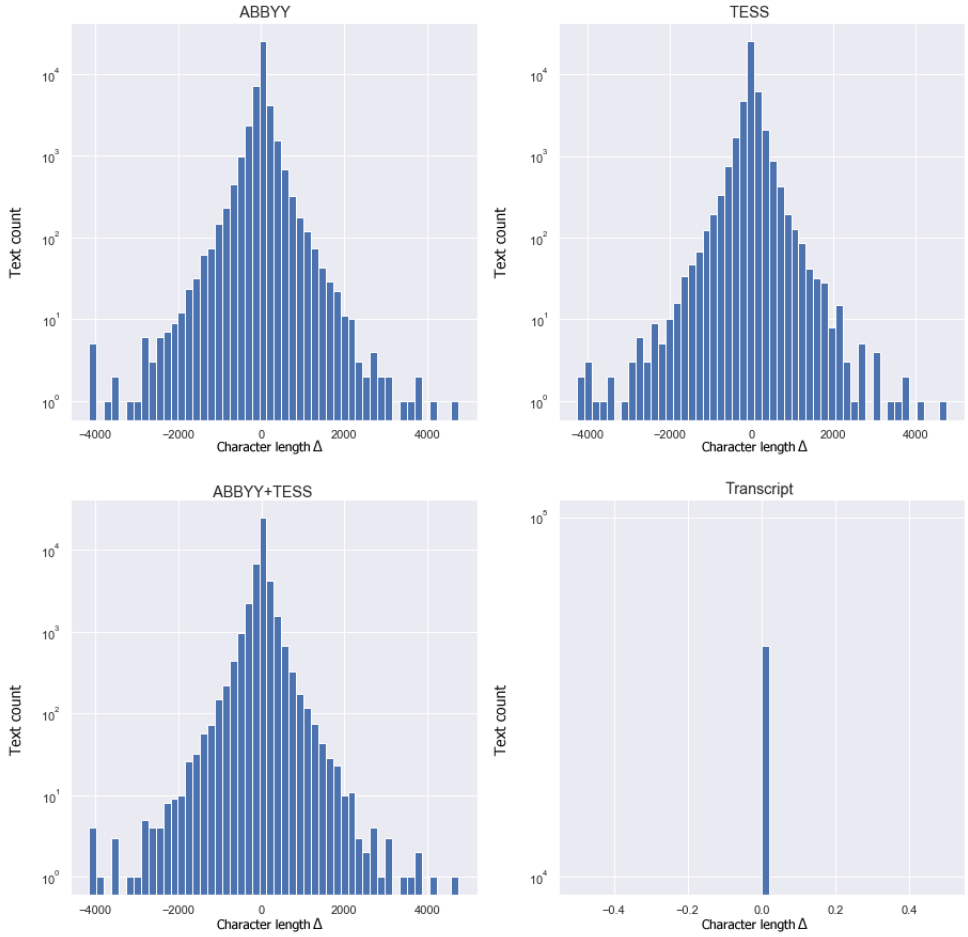


Figure 3.2: Histograms of the difference in character count between each OCR text file versus its corresponding ground truth file in logarithmic scale.

The histograms show a similar behaviour to the character lengths, showing a rapid decline in frequency as the absolute difference grows. As such most files have a relatively small difference in length, but there exists extreme outliers. Manual inspection of some outliers show that either the OCR or the transcript is empty while the other contains an entire paragraph of text, this appears to potentially be a case of the OCR scanning a different section of the page than the transcript for some reason rather than the OCR failing completely.

3.1.2 Character to Character Mapping

To gain further information on what kind of errors occur an investigation into which characters are mapped to which has been performed. However, since the texts aren't aligned by character in the dataset it's not possible to directly read which character is mapped to which. Instead an approach described in a paper by Scott MacKenzie et al.[35] was used.
First off the minimum string distance, otherwise known as the Levenshtein distance, is calculated. This value represents the minimum possible amount of actions one needs to perform to transform one string into the other, with actions being limited to

substitution, removal, and addition of characters. The distance between two strings was calculated with the algorithm described in a paper by R. William Soukoreff et al. [36]:

```
function r(x,y)
    if x==y
        return 0
    return 1

function MSD(A,B)
    for i=0 to len(A)
        D[i,0] = i
    for j=0 to len(B)
        D[0,j] = j
    for i=1 to len(A)
        for j=1 to len(B)
            D[i,j] = min(D[i-1,j]+1,
                         D[i,j-1]+1,
                         D[i-1,j-1]+r(A[i],B[j]))
    return D
```

This yields a matrix which contains the minimum string distance as its bottom right value.

Second, this matrix is then used by the algorithm described in the paper by Scott MacKenzie et al. in order to create all possible optimal alignments between the two sentences. An issue arises here as there are potentially multiple ways of converting one string into the other with the same distance, and the amount of variants can increase exponentially when one adds an additional character to one of the strings. As such, we limited our Levenshtein distance to a maximum of 7. Any string pair beyond this value was discarded and not used. As a result approximately a quarter of the string pairs for each OCR type were discarded for this part of the analysis.

With alignments generated one can naively count how many times each character is mapped to each character, however, it is not known which of the generated alignments is the 'real' one that the OCR performed. Because of this, the counts are weighted by the amount of alignments that were generated for that specific text pair [35]. For example if a string pair creates two possible alignments, and in one of them a specific character is deleted and in the other alignment the same character is instead substituted by second character, then the count would be increased by 0.5 for deleting that character and 0.5 for substituting it with the second character.

For each string pair and OCR method, all character to character mappings are counted in this manner. This yields a confusion matrix representing how many times each character is mapped to another. By dividing each row of the matrix with that row's sum we acquire the probabilities of each ground truth character being mapped to each OCR character.

From these probabilities the common errors can then be directly read, one example of such a confusion matrix can be seen in Figure 3.3, where the Abbyy dataset was analysed. Note that this matrix is far too large to be easily read and is in-

From	To & probability		
a	a=0.99283	å=0.00391	ä=0.00099
A	A=0.95425	Å=0.01842	DEL=0.01033
Å	Å=0.86420	A=0.07407	Ä=0.04938
b	b=0.99289	h=0.00546	DEL=0.00029
B	B=0.98320	R=0.00336	D=0.00335
1	1=0.92887	i=0.02873	I=0.01305
2	2=0.98849	DEL=0.00528	3=0.00115
0	0=0.96848	9=0.01020	O=0.00657
o	o=0.99609	ö=0.00145	e=0.00050
O	O=0.95603	Ö=0.01409	DEL=0.00668

Table 3.2: Table containing the three most common mapping probabilities from a character to another in the Abbyy dataset for some characters. 'DEL' represents the character being missed, or deleted, by the OCR.

stead intended to give an overview of how most characters tend to mostly map to themselves, however if one wants to attempt to view the mappings more directly there are character labels for each row and column that can be seen if one zooms in significantly. Table 3.2 shows some probabilities from this confusion matrix in a more readable format. Here it can be seen that for example 'Å' is mapped to 'Ä' around 5% of the time. Note that the disconnected nature of the diagonal in the matrix stems from some characters only showing up in either the ground truth or the OCR, similarly some characters are missing from the font used by the plotting library, presumably affecting the plot in some way.

These probabilities can be used directly to introduce errors with a similar distribution to another text. This is done by simply iterating over every character in a string and replacing it with a new character sampled from the current character's row in the probability matrix, or removing the character if the sample maps to removal. Note that this does not allow addition of new characters, but the distributions are similar nonetheless.

Figure 3.4 shows a heatmap similar to that in Figure 3.3, but of the difference between the character to character mappings of the Abbyy-Tesseract combined OCR model, and a mapping derived from introducing errors with the Abbyy-Tesseract's mapping to the original ground truth. With the exception of pure white or black pixels that should stem from characters not available in both matrices (and thus being assigned a default value of 0 in the subtraction), it can be seen that both matrices have quite similar values as the difference is close to 0 for the most part. While we are interested in creating similar errors to those that the OCR performs, it should not necessarily be detrimental to have some other errors introduced into the text as well, so for the purposes of creating similar data this method of mapping the characters seems sufficient.

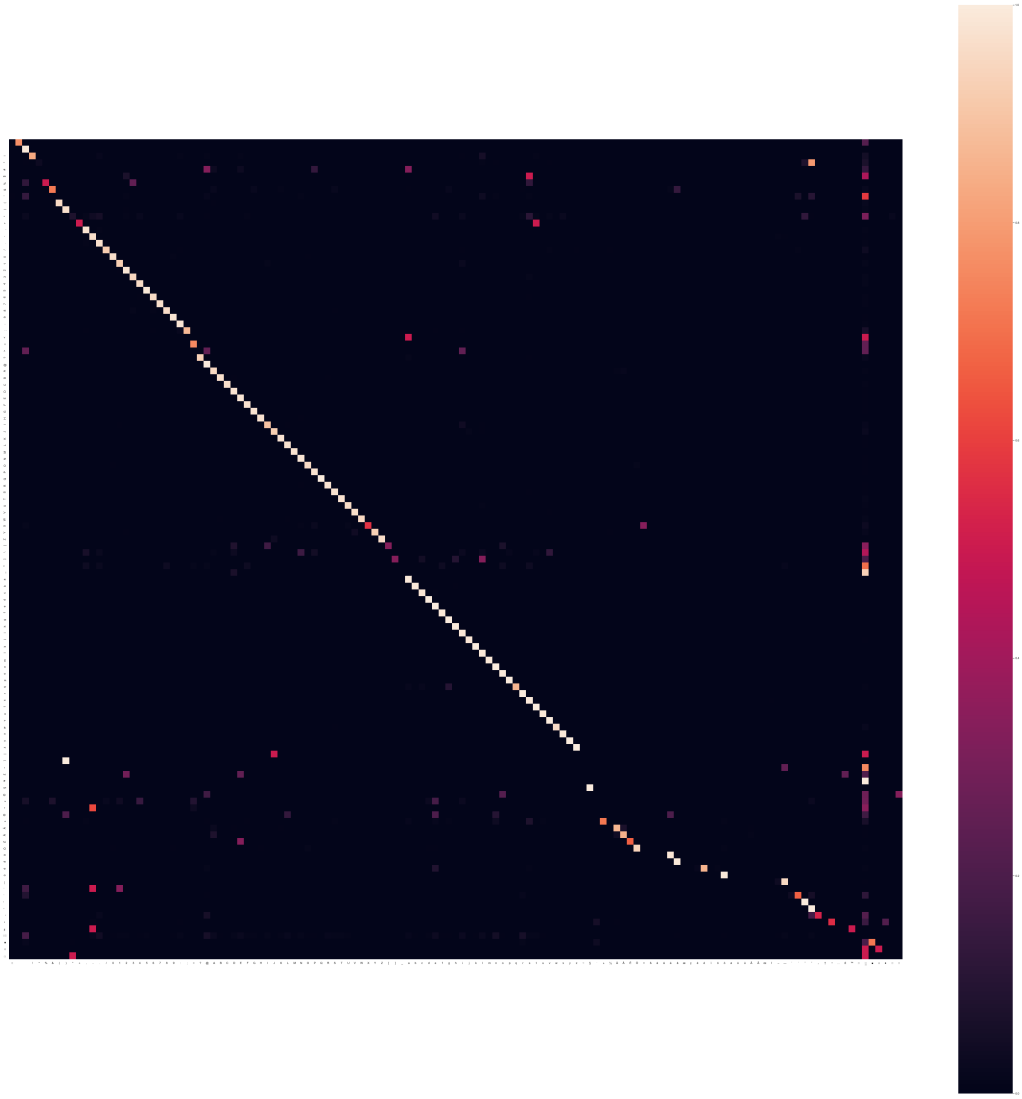


Figure 3.3: Heatmap of confusion matrix containing probabilities of character to character mappings for the Abbyy dataset. Rows represent the truth character and columns represent the OCR character. The unicode character 'U+2591' (omitted from this document due to technical reasons) represents a missing character, i.e. addition or deletion of the character.

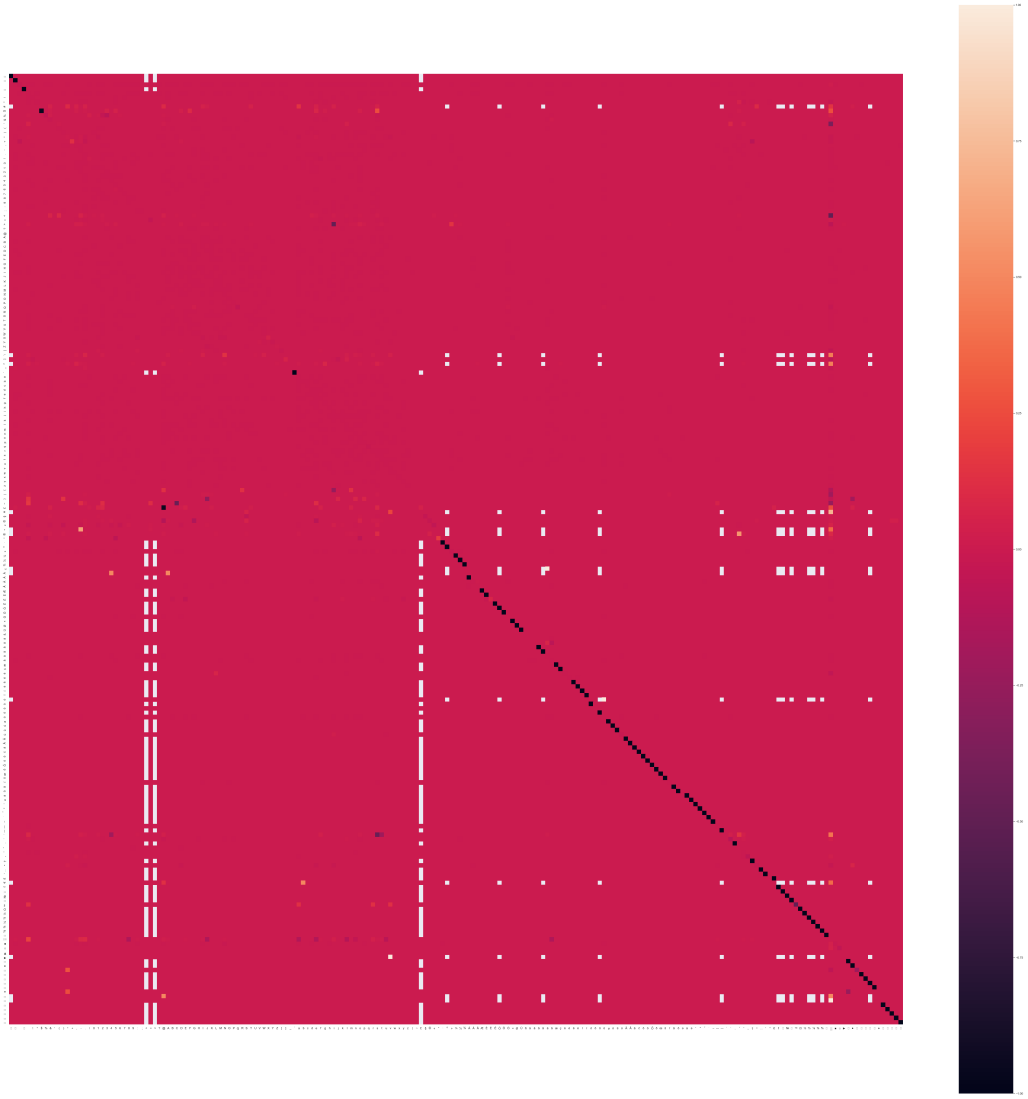


Figure 3.4: Heatmap of the difference between the original probability matrix and the one created from generating artificial errors on the ground truth data with the Abbyy-Tesseract combined OCR model's character to character mappings.

3.2 GigaWord Corpus

In addition to the dataset resulting from the OCR-scans another dataset called the GigaWord Corpus will also be used [37]. This dataset, also hosted by the Swedish Language Bank contains texts which together constitute a billion words. These texts are collected from different sources from 1950 and onwards and come from a variety of genres. These genres include fiction, governmental dealings, science, news and social media. This dataset is made for the purpose of research and experimentation on Swedish texts. Therefore texts are mostly Swedish with the exception of some samples. These exceptions are likely due to the material it is collected from. For instance movie and song titles or subtitles can be expected to appear in newspapers occasionally. Although this dataset does not fully reflect the time span of the OCR

data between 1907 and 2018, there aren't any large spelling reforms within this time period. Therefore the GigaWord dataset should be comparable at a spelling level, even if the language as a whole might be used slightly differently.

4
Method

4.1 Data processing

4.1.1 Preprocessing the Newspaper Material

The preprocessing of the OCR-dataset consists of a number of steps.

- Removal of samples which have been deemed insufficient for training purposes. Samples in this category are: Samples where the length difference between the OCR-text and the GT is deemed too large. These are removed since they appear to be faulty samples stemming from some mis-alignment during annotation; Samples which are below three characters long. This category is removed since we believe that these samples contain little to no context on how it should be corrected. Here we also remove empty samples which exist as a consequence of the OCR frame-based scans.
- Removal of tags added by the annotators. These tags were used to indicate font types, italics, bold text among other characteristics of the text.
- On each sample a <sos> *start of sequence* and <eos> *end of sequence* token is added in order to define sentence boundaries as well as enabling the generation of variable length output from the system.
- Replacement of the newline character "\n" with a space.
- The data is split into batches in which each sample of a batch is padded with a number of <pad> tokens. Each sample is padded to so that its length is the same as the longest sample in the batch plus 15.

In addition, a character-based vocabulary is constructed. This vocabulary contains a character to integer mapping which is used to translate all samples into their integer representations which is used by the system.
The choice to use a character-based vocabulary instead of a word-based one is that most corrections that are needed are on a character level. Using a character-based vocabulary also brings some advantages when dealing with smaller datasets. In particular the risk of running into unknown tokens in a new setting is far less since the amount of unique character tokens are considerably fewer than the amount of unique word tokens.

4.1.2 Preprocessing the GigaWord Corpus

The GigaWord Corpus, as described in Section 3.2, contains texts which originates from a row of different sources such as government or the news. The use of this additional dataset is intended to extend the OCR dataset. Since the OCR dataset

originates from newspapers, it is natural to extract the parts from the GigaWord Corpus which correspond to news sources.

The first instance of news material in the GigaWord Corpus is from 1965 with 53624 sentences. The news material is somewhat sparsely distributed over time up until 1994, with 89 175 sentences from 1976 and 364 226 sentences from 1987 being the only occurrences other than the 1965 ones before 1994. After 1994 the material is plentiful with millions of sentences.

In an effort to balance the data over time and make the size more manageable, 50000 sentences from each of the time periods 1960-1969, 1970-1979, 1980-1989, 1990-1994, 2000-2009 and 2010-2015 were collected and used further.

These sentences or samples are further processed with the addition of the addition of <sos>,<eos> and <pad> tokens in the same way as it is done in Section 4.1.1.

This data is further modified by introducing errors to its (mostly) error-free text using the probabilities calculated in Section 3.1.2. Psuedocode where indexing details have been omitted for the character error generation can be seen below. Note that this function does not introduce any new characters, it only substitutes or deletes them.

```
for char in string:
    if char in prob_matrix:
        char_probs = prob_matrix[char]
        string[char] = sample_char(char_probs)
        if string[char] == 'DEL':
            string[char] = ''

return string
```

Three modifications to the probability matrix were also implemented:

First, a small constant (5.0) was added to the confusion matrix containing the counts of each character mapping, this was done so that there is a nonzero chance of some completely new error occuring during generation, albeit a very small one as for example the character ‘a’ was mapped to itself approximately 39250 times in the Abbyy data it was generated from.

Second, the newline character ‘\n’s row and column in character mapping matrix were set to 0, with the exception of the ‘\n’ to ‘\n’ mapping which was set to 1. Doing this ensures that the probability of newline being mapped to something or vice versa is 0, preventing newline from being added or removed. This was done due to new lines relying on external information not available in a raw text format, i.e. the column width of a given newspaper.

Third, an option to multiply the counts where a character was mapped to itself by a factor was added. This factor adds a way to increase the amount of errors that occur for all characters without changing the ratio between the incorrect characters. Looking at only the probability to get the correct mapping or any incorrect mapping we have that the probability of an error $p_{err} = c_{err}/(c_{ok} + c_{err})$ where c is the count for correct or error mappings. Using this we can see that the multiplication factor f increases the ratio of errors, r, by a factor between 1 and f^{-1} depending on how

many more correct samples there are compared to error samples:

$$r = \frac{c_{err}/(c_{ok} + c_{err})}{c_{err}/(c_{ok}f + c_{err})} = \frac{c_{err} + c_{ok}}{c_{err} + c_{ok}f} \implies \lim_{c_{ok}\to 0} r = 1 \tag{4.1}$$

$$\text{similarly for large } c_{ok}: \quad \lim_{c_{ok}\to \text{inf}} r = \frac{1}{f} \tag{4.2}$$

Thus by setting $f = 1/x$ we can increase the amount of errors generated by a factor up to x.

4.2 Developing the Model

Our model is a character level sequence to sequence model as described throughout Section 2.3. It was implemented in Python [38] using the PyTorch [39] framework. Some modifications and limitations have been made to the model in order to speed up implementation.
First, truncated backpropagation through time described in Section 2.3.2 is used in order to reduce memory usage during training for long sequences. Instead of implementing TBTT for arbitrary k_1, k_2 it was restricted to $k_1 = k_2$, as implementation becomes significantly easier, effectively boiling down to a single function call to detach PyTorch's computation graph every k_1 time steps.
The second limitation is that the model only supports single layer LSTMs.
Lastly, the attention implementation is limited to Bahdanau attention, there are other ways to implement attention, but they have not been tested as they require creating a different network structure.

4.2.1 Model Walkthrough

This section will walk through the model as a whole to give an overview of the system, from just having the preprocessed OCR data to training and evaluating the model.
Starting off, the OCR data and its corresponding transcript were briefly analysed as shown in Section 3.1.1, while this analysis is barely used directly it does confirm some assumptions on the dataset, such as a rapid frequency decrease with respect to character count (Figure 3.1), as well as the OCR being relatively balanced around adding or removing characters (Figure 3.2). After this the character to character mapping described in Section 3.1.2 was performed, generating confusion matrices containing the amount of times single characters were mapped to each other by the OCR systems, and by extension the frequencies of their mappings. These in turn are used to expand the dataset by generating new errors in new texts as described in Section 4.1.2, choice of error factor was left as a hyperparameter and as such varies across different training attempts, and are therefore shown along with the rest of the hyperparameters in the different models in Section 4.2.2.
With the data generation finished the training the model is next. First, most of the OCR data, subject to restrictions such as minimum and maximum length and length difference between OCR and transcription, is loaded into the training set with

the remainder being put into the validation and test sets. This split is fixed and does not change between model iterations. The generated data from the GigaWord corpus is also loaded into the respective datasets subject to the length constraints. Note that the GigaWord data has been preemptively split into the three datasets and the validation and test parts use errors that have been generated once with a factor of 1, i.e. with a distribution similar to the original OCR data. In contrast the training part of the corpus is freely re-generated with a freely chosen factor.
Second, this data is used to build a 'vocabulary', creating a mapping each unique character to an integer. This mapping also includes special characters for padding, unknown characters, start and end of sentence tokens.
Third, the network model is instantiated according to chosen hyperparameters and training begins. Throughout training the network iterates over the training dataset in batches and uses backpropagation to update its weights. For each minibatch the length of every input is padded to have the same total length, potentially adding extra pad tokens beyond the longest sentence's max length to allow for generating extra characters. These sentences are then fed through the encoder, followed by the decoder iterating over the sentence as described in Section 2.3.1. Every k_1 steps of the decoder (or upon reaching the final character), truncated backpropagation through time (Section 2.3.2) is performed using NLLLoss (Negative Log Likelihood loss), implemented with Pytorch [39]. Here *gradient clipping* is used, potentially setting the gradient to a maximum value if it is exceeded in order to prevent exploding gradients. Between each full iteration over the training data the loss of the validation set is calculated as well. Training continues until either the loss stagnates or the validation loss begins to increase instead of decreasing.
Once training is finished the entire validation and test datasets are fed through the network, whose outputs are then used to calculate performance metrics. Note that unlike the loss calculation these calculations use the beam search method described in Section 2.3.5
The validation set's metrics are viewed and used to gauge how well the network performs. New hyperparameters are then chosen manually, primarily based upon viewing the training and validation loss of the network in an attempt to identify whether the network for example requires more or less complexity, larger or smaller learning rates, or potentially further augmented data. The test set's metrics are ignored until all training is finished, and are then used to compare the models on completely unseen data during training. This process is then repeated in an attempt to find a network that successfully generalizes when shown the validation data.

4.2.2 Model Instances

Since results of neural network models are dependent on hyper-parameter choices and data selection, some different instances of the model have been trained. These instances differ to each other by different choices of these parameters. Some parameters, however, are shared across instances. These are that all instances have an encoder and decoder which contain one layer each and the TBPTT length is $k1 = k2 = 100$. Additionally the beam size used for decoding is set to 10.
The instances we have worked with have used OCR-data originating between 1907

and 2018. This cutoff is made because of the spelling reform during this time, allowing data from the GigaWord Corpus be used as an extension with similar data. The samples used in each instance are limited to be between 5 and 200 characters and have a maximum length difference between input and GT for the OCR-data i is set to $3 + 0.1 * L$, where L is the input string's length.

For all shown instances the Abbyy OCR-data was used in combination with the extracted GigaWord Corpus news sentences, with the latter having synthetically generated errors. Tesseract and the mixed Abbyy-Tesseract model are therefore unused for all shown models. Similarly, all shown models use the Adam optimizer.

We have trained many more instances than the ones reported here, however due to a mishap with Python's pickle function when storing them their training parameter data was lost with the exception of neuron count and embedding dimensionality. Although the networks themselves are still available we have chosen to omit them as none of them performed better than our best model on any metric, it is impossible to comment on their parameter choices, and because it would require manually testing which data was used during training in order to reconstruct the same vocabulary. As a side effect of this the exact batch sizes for instances 6 through 8 are also missing, they used a batch size in the 10-15 range due to the larger network requiring more VRAM, preventing larger batch sizes.

The different hyperparameter settings and which data was used for each instance is shown in Tables 4.1 and 4.2. In Table 4.1 the parameters are as follows:

- hid size: Number of features of the hidden state of the LSTM.
- emb size: Number of features of the embedded characters. This is significantly smaller than a word level implementation due to there being far fewer unique characters than words.
- lr: The learning rate.
- weight decay: The weight decay used by the optimizer, in this case PyTorch's Adam implementation which uses L2 normalization.
- batch size: How many data samples are processed simultaneously.
- drop rate: Chance for a value of the embedded characters to be set to 0 during training.
- tf ratio: Chance of using teacher forcing during training per character.

Table 4.1: Hyperparameters used for the different instances.

Instance	hid size	emb size	lr	weight decay	batch size	drop rate	tf ratio
Instance 1	128	4	0.001	0.00001	50	0.2	1
Instance 2	128	4	0.001	0.00001	50	0	1
Instance 3	128	4	0.001	0.00001	50	0	0.5
Instance 4	128	4	0.001	0.00001	50	0	0
Instance 5	128	4	0.001	0.00001	50	0.2	1
Instance 6	400	3	0.001	0	$10 - 15$	0	0.5
Instance 7	450	3	0.001	0	$10 - 15$	0	0.5
Instance 8	384	3	0.001	0	$10 - 15$	0	0.5

Table 4.2: Data used for different instances.

Instance	Data
Instance 1	OCR-data Abbyy Gigaword Corpus factor 0.5
Instance 2	OCR-data Abbyy
Instance 3	OCR-data Abbyy
Instance 4	OCR-data Abbyy
Instance 5	OCR-data Abbyy Gigaword Corpus factor 1 Gigaword Corpus factor 0.5 Gigaword Corpus factor 0.3
Instance 6	OCR-data Abbyy Gigaword Corpus factor 1 Gigaword Corpus factor 0.5 Gigaword Corpus factor 0.3
Instance 7	OCR-data Abbyy Gigaword Corpus factor 1 Gigaword Corpus factor 0.5 Gigaword Corpus factor 0.3
Instance 8	OCR-data Abbyy Gigaword Corpus factor 1 x4

4.3 Evaluation

Evaluation of the model is done in two stages. The first stage is during training and the second during post-training.

To observe the progress of the training and and to relatively quickly be able to tell if an experiment has promise or not is done through the loss function. As the model trains, it constantly computes the loss function for each sample. The loss, which is a measure of how close the output is to the target is averaged over the batch, and further averaged over the epoch. The epoch average therefore shows how close on average the batches during the epoch were to their targets. By observing the change of the epoch average through epochs, one would expect its value to decrease and approach zero as training proceeds. In instances where the loss average over epochs is increasing over several epochs it is clear that something is wrong. This could occur due to mis-configuration of hyperparameters and experiments were this happens are discarded. At the end of each epoch, the average loss over the validation set is also calculated. This step gives an indication of how well the system is able to adapt to data it has not seen before. The expectation is that this loss would also decrease over epochs but might at some point start increasing as the system learns the data in the training set too well and forgets how to generalize, also known as overfitting. In this case the training should be stopped at the lowest point where the validation loss were the lowest. Note that for Instances 5-8 the validation and test sets include parts of the original GigaWord dataset and their loss is therefore not directly comparable to Instances 1-4.

After training is completed the system produces outputs for the samples contained

in the validation and test set. On these produced outputs, the *word error rate*, *f-score*, *recall* and *precision* (described in Section 2.4) as it relates to the ground truth is calculated and compared to their values in their original form.

5

Results

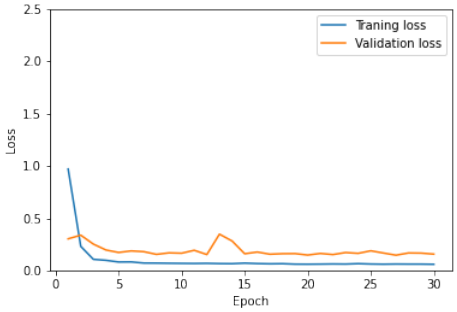
In this chapter the results produced by the different instances of our model are presented.

The training was done in varying lengths for the different instances. The smaller instances (Instances 1-4) were trained for 30 epochs each. The larger instances (Instances 5-8) were trained varying number of epochs. The shortening of training in regards to epochs is due to each epoch taking significantly longer because of either; a larger network size, or using significantly more data. These instances were therefore trained until further training seemed fruitless with regards to the training and validation loss.

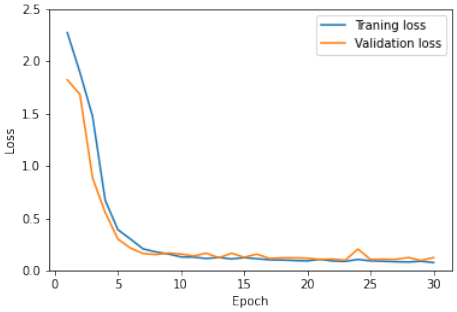
5.1 Quantitative Evaluation

The training and validation loss over epochs of the different instances can be seen in Figure 5.1. The training loss of all instances have a rapid decline during the first few epochs of training, with the exception of Instance 3 and 4 which take slightly longer. After this rapid decline, the loss of all instances seem to plateau and remain somewhat stationary for the remainder of the training.

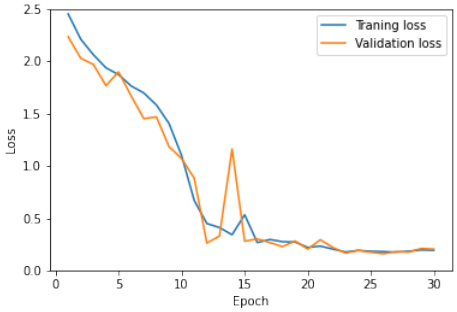
Tables 5.1 and 5.2 show the performance metrics of the model instances on the validation and test sets respectively. The tables also show the baseline metrics for the input data, before it was sent through the network. The colored arrows indicate whether the values are larger or smaller than the baseline with up and down arrows respectively, as well as whether they are better or worse with blue and red colors respectively. From these tables it can be seen that most instances, with the exception of Instance 2, were unable to produce outputs which resulted in an improvement of the word based metrics (F-score,recall, precision, WER) and instead made them worse. Despite this most instances performed better on the character based metric CER with the exception of Instance 3 and 8, with Instance 3 performing the worst on all metrics.



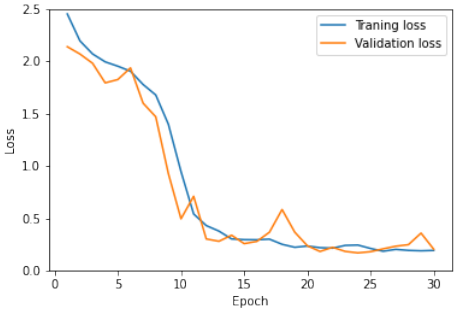
(a) Instance 1



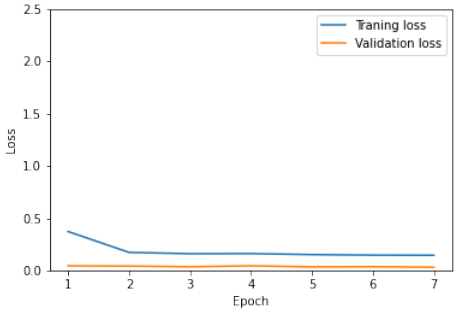
(b) Instance 2



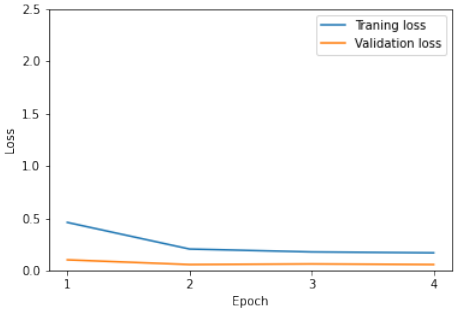
(c) Instance 3



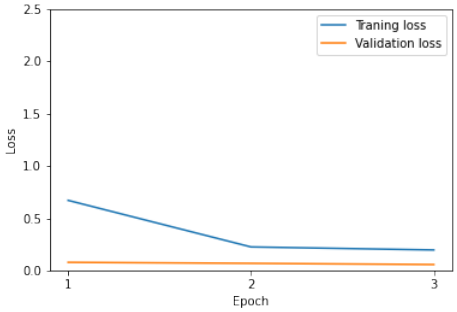
(d) Instance 4



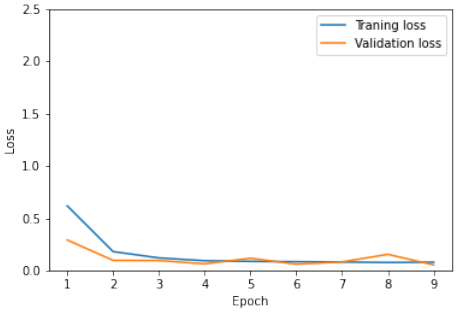
(e) Instance 5



(f) Instance 6



(g) Instance 7



(h) Instance 8

Figure 5.1: Loss graphed over epochs for the different instances. Note that Instances 5-8 include subsets from the GigaWord dataset and are not directly comparable to Instance 1-4.

Table 5.1: Evaluation metrics of the networks outputs on the validation set, as well as a baseline acquired by calculating the metrics on the network's input.

Instance	F-score	Recall	Precision	WER	CER
Baseline	0.8508 -	0.8526 -	0.8507 -	0.1755 -	0.0830 -
Instance 1 (30)*	0.8462 ↓	0.8488 ↓	0.8454 ↓	0.1806 ↑	0.0475 ↓
Instance 1 (10)*	0.8457 ↓	0.8464 ↓	0.8467 ↓	0.1793 ↑	0.0476 ↓
Instance 2	0.8547 ↑	0.8552 ↑	0.8566 ↑	0.1677 ↓	0.0462 ↓
Instance 3	0.2906 ↓	0.2923 ↓	0.2955 ↓	0.7638 ↑	0.6565 ↑
Instance 4	0.8462 ↓	0.8488 ↓	0.8454 ↓	0.1806 ↑	0.0475 ↓
Instance 5	0.7621 ↓	0.7637 ↓	0.7620 ↓	0.2634 ↑	0.0734 ↓
Instance 6	0.8026 ↓	0.8040 ↓	0.8035 ↓	0.2277 ↑	0.0595 ↓
Instance 7	0.8319 ↓	0.8334 ↓	0.8319 ↓	0.1919 ↑	0.0534 ↓
Instance 8	0.7097 ↓	0.7123 ↓	0.7096 ↓	0.3235 ↑	0.0852 ↑

Table 5.2: Evaluation metrics of the networks outputs on the test set, as well as a baseline acquired by calculating the metrics on the network's input.

Instance	F-score	Recall	Precision	WER	CER
Baseline	0.8566 -	0.8591 -	0.8555 -	0.1647 -	0.0844 -
Instance 1 (30)*	0.8516 ↓	0.8541 ↓	0.8507 ↓	0.1711 ↑	0.0487 ↓
Instance 1 (10)*	0.8499 ↓	0.8521 ↓	0.8491 ↓	0.1737 ↑	0.0485 ↓
Instance 2	0.8606 ↑	0.8622 ↑	0.8604 ↑	0.1585 ↓	0.0477 ↓
Instance 3	0.2358 ↓	0.2352 ↓	0.2421 ↓	0.8140 ↑	0.6704 ↑
Instance 4	0.8516 ↓	0.8541 ↓	0.8507 ↓	0.1711 ↑	0.0487 ↓
Instance 5	0.7670 ↓	0.7692 ↓	0.7660 ↓	0.2581 ↑	0.0700 ↓
Instance 6	0.8063 ↓	0.8087 ↓	0.8053 ↓	0.2168 ↑	0.0615 ↓
Instance 7	0.8286 ↓	0.8306 ↓	0.8280 ↓	0.1961 ↑	0.0544 ↓
Instance 8	0.6892 ↓	0.6918 ↓	0.6885 ↓	0.3490 ↑	0.0939 ↑

5.2 Qualitative Evaluation

With Instance 2 showing itself to be clearly superior to the other instances on all metrics, a more thorough inspection will focus on this instance.

Tables 5.3 and 5.4 show the character to character mappings (as in Section 3.1.2) between the input and target text, and the output and target text respectively.

Table 5.3: Table containing the three most common mapping probabilities from a character in the test set OCR the test set truth for some characters. 'ADD' represents the character not being in the truth and being added by the OCR (i.e. it should be deleted).

From	To & probability		
a	a=0.97439	å=0.00751	s=0.00626
A	A=0.96188	Å=0.02354	Ä=0.00673
Å	Å=0.89583	Ä=0.06250	Ä=0.04167
b	b=0.97349	h=0.00241	k=0.00241
B	B=0.97378	C=0.00375	E=0.00375
1	1=0.94466	ADD=0.01444	I=0.00758
2	2=1.0	N/A	N/A
0	0=0.98734	O=0.00633	Ö=0.00317
o	o=0.98207	ö=0.00279	c=0.00239
O	O=0.96109	Ö=0.01816	G=0.00778
space	space=0.87273	ADD=0.11298	tab=0.00697

Table 5.4: Table containing the three most common mapping probabilities from Instance 2's output on the test set to the truth for some characters. 'ADD' represents the character not being in the truth and being added by the model (i.e. it should be deleted).

From	To & probability		
a	a=0.97254	s=0.00606	å=0.00597
A	A=0.96825	Å=0.01474	Ä=0.00680
Å	Å=0.81356	A=0.08475	Ä=0.08475
b	b=0.97934	h=0.00729	ADD=0.00497
B	B=0.96259	s=0.00420	M=0.003717
1	1=0.95976	l=0.00805	ADD=0.00771
2	2=0.98800	ADD=0.00749	C=0.00400
0	0=0.97806	O=0.00627	.=0.00314
o	o=0.98008	ADD=0.00391	ö=0.00279
O	O=0.94981	Ö=0.02317	G=0.00772
space	space=0.97966	ADD=0.01054	tab=0.00697

Table 5.5 shows some manually selected samples from the test set using the model. The samples have been chosen to display some different behaviours of the model for different types of input and target data. Note that for a large majority (1007 out of 1327) of the input to the network and its output are identical, and from those 688 out of 1007 have an identical target output.

Table 5.5: Examples from the test set produced with corresponding output from Instance 2 and its GT. Errors are highlighted in bold where possible.

Input	**Brännvlnatlllvärknlngen** i riket.
Output	Brännvinstillvärkningen i riket.
Truth	Brännvinstillvärkningen i riket.
Input	i Malmberget i **dsg** Tisdag den 13 Juni kl. 8,15 e. m.
Output	i Malmberget i **dsg** Tisdag den 13 Juni kl. 8,15 e. m.
Truth	i Malmberget i dag Tisdag den 13 Juni kl. 8,15 e. m.
Input	På tal om arbetsavtalslagarna sade han sig ha tagit deras fall med jämn- mod. Man har nämligen en känsla av att tiden ännu ej är mogen för en så- dan lagstiftning.
Output	På tal om arbetsavtalslagarna sade han sig ha tagit deras fall med jämn- mod. Man har nämligen en känsla av att tiden **än känsla av att tiden ännu ej är mogen för en så- dan lagstif**
Truth	På tal om arbetsavtalslagarna sade han sig ha tagit deras fall med jämn- mod. Man har nämligen en känsla av att tiden ännu ej är mogen för en så- dan lagstiftning.
Input	Sommarsäsongens modärnaste Damhattar och **Monte- r/ngaartlklar**. Enkla oklädda Hattar från 1,25 st., som monteras efter beställning, samt monterade Hattar och Mo- dellhattar upp till 75 **å** 100 kr st.
Output	Sommarsäsongens modärnaste Damhattar och Monte- ringaartikar. Enkla oklädda Hattar från 1,25 som monteras efter beställning, samt monterade Hattar och Mo- dellhattar upp till 75 **å** 100 kr st.
Truth	Sommarsäsongens modärnaste Damhattar och Monte- ringsartiklar. Enkla oklädda Hattar från 1,25 st., som monteras efter beställning, samt monterade Hattar och Mo- dellhattar upp till 75 à 100 kr. st.
Input	Illa känner man de svenska bönderna ooh svenska bönders tänkesätt, om man ej förstår **oeh** vet att **d • t** brödet **skulls** för dem bli alltför hårdsmält.
Output	Illa känner man de svenska bönderna och svenska bönders tänkesätt, om man ej förstår och vet att **der drödet skulls** för dem bli alltför hårdsmält.
Truth	Illa känner man de svenska bönderna och svenska bönders tänkesätt, om man ej förstår och vet, att det brödet skulle för dem bli alltför hårdsmält.

Table 5.6 shows examples produced by Instance 3, the instance which has the poorest metric scoring of any instance. In these examples it can be seen that the model does not appear to have a proper response to the input and mostly produce nonsense.

Table 5.6: Examples from the test set produced with corresponding output from Instance 3 and its ground truth. Errors are highlighted in bold.

Input	Aftonbladets ˆ **KLICHÉANSTALT**]
Output	**ATAbT**.
Truth	AFTONBLADETS KLICHÉANSTALT.
Input	Sista sidan i **Sthlroa- o. KvälUupplagorna 2t Sre**, alla dagl upplagorna 30 öre.
Output	Sista sidan i **Sthla dagl upplagorna db.** **spplagorna db. spplagorna db. spplagorna d. bpplagorna**
Truth	Slsta sidan: Sthlms- o. Kvällsupplagorna 26 öre, alla dagl upplagorna 30 öre.

5.2.1 Intermittent Inspection

An intermittent inspection of Instance 1 was made. By comparing the output at epoch 10 and epoch 30 (a period over which the loss hardly changes) some specific changes in the output can be seen. Table 5.7 shows the input, output and ground truth of a sample after training for 30 epochs. From this table one can see that while some errors remain in the output such as the failure to correct “langt” to “lungt” and “Janl” to “Jan”, the correction of “oeh” and “osh” to “och” is done. The same sample was inspected for the checkpoint created at epoch 10. At this checkpoint the model produces the output presented in Table 5.8. Here it can be seen that, while the correction of “oeh” is still present the correction of “osh” is not.

Table 5.7: A sample input from the test set to the model at epoch 30 with its output and GT. Errors are highlighted in bold

Input	lagnt **oeh** stilla avled i Kiruna **dea** 12 **Janl** kl. 3.4B f. m., djupt sörjd och sak-nad av oss, 5 syskon, varder härmed släktingar **osh** vännar tillkännagivst.
Output	lagnt och stilla avled i Kiruna **dea** 12 **Janl** kl. 3.48 f. m., djupt sörjd och sak-nad av oss, 5 syskon, varder härmed släktingar och vännar tillkännagivst.
Truth	lugnt och stilla avled i Kiruna den 12 Jan kl. 3.48 f. m., djupt sörjd och sak-nad av oss, 5 syskon, varder härmed släktingar och vännar tillkännagivst.

Table 5.8: A sample from the test set input to the model at epoch 10 with its output and GT. Errors are highlighted in bold

Input	lagnt **oeh** stilla avled i Kiruna **dea** 12 **Janl** kl. 3.4B f. m., djupt sörjd och sak-nad av oss, 5 syskon, varder härmed släktingar **osh** vännar tillkännagivst.
Output	lagnt och stilla avled i Kiruna **dea** 12 **Janl** kl. 3.48 f. m., djupt sörjd och sak-nad av oss, 5 syskon, varder härmed släktingar **osh** vännar tillkännagivst.
Truth	lugnt och stilla avled i Kiruna den 12 Jan kl. 3.48 f. m., djupt sörjd och sak-nad av oss, 5 syskon, varder härmed släktingar och vännar tillkännagivst.

6
Discussion

6.1 Quantitative Evaluation

It can be clearly seen in Tables 5.1 and 5.2 that every instance but Instance 2 performs worse than the untouched OCR texts on word level metrics, with that instance just barely surpassing the baseline. All instances but the third however do see an improvement at a character level. Outside of Instance 2, Instance 5 is particularly interesting, in Figure 5.1 it can be seen that this model achieves the lowest validation loss. However, this model had an extended validation set compared to Instances 1-4, which when combined with its overall poor performance on the metrics indicate that it is likely that the generated data does not represent the original data, as this loss reduction must stem from performing better on generated data and not the original.
In Tables 5.3 and 5.4 it can be seen that the Instance 2 model generally has similar probabilities of correctly mapping a character to itself, but there are variations in both directions. Of particular interest is the mapping of space which shows a drastic increase in correct mappings and by extension a decrease in instances where the space wasn't in the target text. This indicates that a lot of the networks performance appears to come from removing extra faulty spaces added by the OCR.

6.2 Qualitative Evaluation

Most outputs of Instance 2 simply mimic the input, because of this the model does not appear to have any issues with rarer words such as named entities or numbers. An example of this can be seen in the second example sentence in Table 5.5 where the model simply reproduces the sentence, replicating the names and the numbers, as well as the time formatting at the end of the sentence. At the same time however it fails to correct the error in the word "dag".
The third sentence shows an example of the model repeating previous words without that being a part of the input. This is a common issue in text generation where language models often get stuck repeating some part of a sentence over and over [40, 41], although in our case it managed to break out of the loop.
Sentence four showcases multiple changes to the input text, it fixes one character error, skips over another. This is then followed by deleting the entire "st.," structure (used to indicate how many of a thing there are, effectively translated to "pieces" in English) in the sentence.
The fifth sentence shows an almost successful correction of "d • t" into "det", which

at a glance looks like two substitutions, but in reality the source sentence includes two spaces around the bullet character which the model has successfully deleted, this falls in line with the observations seen in Tables 5.3 and 5.4. Unfortunately the model did also incorrectly change the final character into an 'r'.
Finally, the first sentence showcases a completely successful correction, substituting three characters in order to form the correct spelling.
From the examples shown for Instance 3 in table 5.6. It is clear why this instance scored so poorly on the metrics as its output has little to no resemblance to the input let alone the GT.
Although most outputs of Instance 2's are a mirroring of the input, it does show promise. This implies that the system has managed to encode the information in the input to such a degree that is was interpretable for the encoder which in turn could decode it. In addition to the mirroring we also see some corrections. Observing the outputs of Instance 3, seen in Table 5.6, we see that this instance mostly produced nonsense and is completely useless. In comparison to Instance 3, we can see a large quality difference and conclude that Instance 2 shows some success.

6.3 Word and Character based metrics

In Tables 5.1 and 5.2 a general trend can be seen where all models worse on word level corrections than character level ones. This is somewhat expected as the network is optimized at a character level. One can imagine a sentence with two words, where one has many incorrect characters and the other word has none. If the network were to correct all but one character in the first word and then introduce a single error into the second both words would be incorrect while the overall character error rate would be significantly reduced.
A possible explanation as to why the instances which are trained on larger amounts of data did not perform above the baseline for most metrics could be that the generation of errors simply is not good enough. It might be that the assumption that errors exist in a unigram fashion, that is, that the errors are made independently of surrounding characters, is incorrect. Errors might instead exist in n-gram fashion, depending on n sequential characters. If this is the case then instances trained on the errors introduced on the Gigaword corpus were trained on errors that do not occur naturally in the original data.

6.4 Loss

As seen from the results in the previous chapter, the loss of all instances seemed to plateau. Although the plateau was present during significant portions of some instances training time, they showed no sign of overfittining. There are, however, some signs that especially the instances operating on larger datasets experience some underfitting as the loss plateaus at a value much higher than their counterparts as well as the validation set. It is not surprising that underfitting makes results worse for the networks operating on larger datasets as an increase in dataset size often makes this issue worse [42, 43].

To solve the issue of underfitting, it is common to increase the dimensionality of the network. Unfortunately this is something we were unable to do in this project as the training of larger networks simply took too long with the computational resources we had readily available. For reference the larger networks took upwards of a 10 hours per epoch on an RTX 2070 Super due to their small batch size and large datasets. In hindsight TBTT should have been tested further to see how low $k1, k2$ could have been set without a significant performance impact as smaller values would decrease memory usage and thereby allow for larger batches and faster training.

6.4.1 Loss as a Metric

The results produced by Instance 1 were promising, with a very low training loss and relatively low validation loss. The instance does not show any signs of overfitting even though the loss has plateaued. In spite of the plateauing of the loss of Instance 1 shown in Figure 5.1, the output is changing slightly as shown by the correction to the word "och" which cannot be observed during the evaluation after 10 epochs.

That the apparent change in the output which corresponds to a correction does not appear until later in training and displays a negligible change in the loss function is of note. This suggests that the difference in loss from a nearly correct sequence and one which has been corrected is rather small. This is due to the fact that most corrections consists of a change to a small amount of characters, and the change of one or two characters in a string of up to 200 constitutes only a small change. This means that although the loss gives an indication of how close to the target the output is, it might not give a clear indication of when the network learns to make corrections.

Furthermore, the network generates the entire output from scratch based on the encoder's interlingua, so a significant chunk of the loss graphs could portray the network learning how to replicate the input as output. In other words, the baseline loss lies close to 0 when compared to the starting point of the network's training, once again indicating that the loss at a glance does not indicate good performance until it has reached the baseline loss one would get from simply copying the input.

6.5 Teacher Forcing

The effect of teacher forcing for this problem was explored over Instances 2-4. Here Instance 2 had teacher forcing of 1 while Instances 3 and 4 had 0.5 and 0 respectively. The change of the ratio from 1 to 0.5, as in Instance 3, seem to have made our most successful instance settings perform extremely poorly. The change to a ratio of 0, as in Instance 4, performed similarly to most of the instances with being slightly worse than baseline in all metrics except the CER.

Linearly decreasing the TF ratio between each epoch was initially tested but this was removed in favor of a constant value as the amount of epochs required to reach a stable loss seemed to change pretty drastically depending on other hyperparameters, either causing TF to not reach 0 or to reach it early in training.

6.6 Size of Validation and Test set

With the small size of the original OCR dataset it follows that the corresponding validation and test sets are small as well. Because of their small size it is possible that what is contained in the validation and test set is not fully representative of the dataset. This is further emphasized by the validation and test sets being made out of specific pages' samples instead of randomly picked samples from any page. One can imagine that if, for example a page has a rare font it would when combined with the small dataset skew the validation and test set errors.

7
Conclusion and Future work

7.1 Conclusion

The aim of this project has been to develop a neural network model and train it to do post-correction on the OCR-data collected from the newspapers with the purpose of improving it's quality. The implemented model was a seq2seq model with attention, which has shown promising results on similar tasks in other languages. Quality is measured through a number of evaluation metrics, which are applied before and after the model was tested. As the model Instance 2 shows an improvement in all metrics, we conclude that an increased quality has been achieved. Additionally we conclude that for the question of whether or not a deep learning model could improve the OCR accuracy for Swedish texts, the answer is yes. There is however reasons to believe that conclusions relating to the full dataset cannot be made with confidence due to reasons such as the small size of the validation- and test-set. The fact that the number of actual corrections made by the system is relatively small as well as the generation of some new errors suggests that there is still a lot of improvements that could be made.

7.2 Future Work

We do not believe we achieved the best results on the task of post-OCR with deep learning methods. Rather, we believe that with additional effort and with perhaps slightly different approaches one could achieve success. As we have discussed, we do believe that with different hyperparameter settings better results could be achieved. In particular instances with larger amounts of neurons and layers are of particular interest to examine. In addition to an extended parameter search it would also be of interest to observe how other metrics than the loss change over epochs, as suggested in Section 6.4.1 this choice of loss might not show the entire picture.
In a continuation of this project it would also be advisable to examine n-gram errors in the data and potentially adapt the way errors are introduced to additional data. Ideally you would want more data from the source but as it is very expensive to create transcriptions, this might not be feasible.
Another continuation is that since we only worked with data from the Abbyy system, attempts at correction of data from the systems other than Abbyy should be examined. Additionally if sufficient data originating from the 19th century is available, training and correction attempts on that part of the dataset is also of interest. Slightly different approaches could also be used while training the model, such as

using transfer learning with a pre-trained model of the Swedish language as a base. Alternatively one could use it with a model trained to simply copy every input to the output, which could potentially reduce the chance of the model generating repeated outputs as shown in one example of Table 5.5.

Bibliography

[1] G. V. Research, "Optical Character Recognition Market Size, Share & Trends Analysis Report By Type (Software, Services), By Vertical (Retail, BFSI, Government, Education, Healthcare), By Region, And Segment Forecasts, 2019 - 2025 Summary," 2019.

[2] M. Volk, L. Furrer, and R. Sennrich, "Strategies for reducing and correcting ocr errors," in *Language Technology for Cultural Heritage* (C. Sporleder, A. van den Bosch, and K. Zervanou, eds.), (Berlin, Heidelberg), pp. 3–22, Springer Berlin Heidelberg, 2011.

[3] C. Amrhein and S. Clematide, "Supervised OCR Error Detection and Correction Using Statistical and Neural Machine Translation Methods," *J. Lang. Technol. Comput. Linguistics*, vol. 33, pp. 49–76, 2018.

[4] G. T. Bazzo, G. A. Lorentz, D. Suarez Vargas, and V. P. Moreira, "Assessing the Impact of OCR Errors in Information Retrieval," in *Advances in Information Retrieval* (J. M. Jose, E. Yilmaz, J. Magalhães, P. Castells, N. Ferro, M. J. Silva, and F. Martins, eds.), (Cham), pp. 102–109, Springer International Publishing, 2020.

[5] C. Rigaud, A. Doucet, M. Coustaty, and J. Moreux, "Icdar 2019 competition on post-ocr text correction," in *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1588–1593, 2019.

[6] P. Simon, "OCR post-processing of historical Swedish text using machine learning techniques," 2019.

[7] S. Drobac and K. Lindén, "Optical character recognition with neural networks and post-correction with finite state methods," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 23, pp. 279–295, Dec 2020.

[8] M. Silfverberg, P. Kauppinen, and K. Lindén, "Data-driven spelling correction using weighted finite-state methods," in *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*, (Berlin, Germany), pp. 51–59, Association for Computational Linguistics, Aug. 2016.

[9] S. Salimzadeh, *Improving OCR Quality by Post-Correction*. PhD thesis, Universiteit van Amsterdam, 2019.

[10] C. Rigaud, A. Doucet, M. Coustaty, and J.-P. Moreux, "ICDAR 2019 competition on post-OCR text correction," in *Proceedings of the 15th International Conference on Document Analysis and Recognition*, (Sydney, Australia), pp. 1588–1593, 2019.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] B. Mehlig, *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press, 2021.

[13] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[14] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1724–1734, Association for Computational Linguistics, Oct. 2014.

[15] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," NIPS'14, (Cambridge, MA, USA), p. 3104–3112, MIT Press, 2014.

[16] J. Eisenstein, *Introduction to Natural Language Processing*. Adaptive Computation and Machine Learning series, MIT Press, 2019.

[17] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ArXiv*, vol. 1409, 09 2014.

[18] T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, (Lisbon, Portugal), pp. 1412–1421, Association for Computational Linguistics, Sept. 2015.

[19] S. Chaudhari, V. Mithal, G. Polatkan, and R. Ramanath, "An Attentive Survey of Attention Models," 2020.

[20] "A Gentle Introduction to torch.autograd." `https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html#computational-graph`. Accessed: 2021-04-09.

[21] I. Sutskever, *Training Recurrent Neural Networks*. PhD thesis, CAN, 2013. AAINS22066.

[22] C. Tallec and Y. Ollivier, "Unbiasing truncated backpropagation through time," p. arXiv:1705.08209, 2017.

[23] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba, "Sequence level training with recurrent neural networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[24] F. Schmidt, "Generalization in generation: A closer look at exposure bias," pp. 157–167, 01 2019.

[25] T. He, J. Zhang, Z. Zhou, and J. Glass, "Quantifying Exposure Bias for Open-ended Language Generation," *arXiv e-prints*, p. arXiv:1905.10617, May 2019.

[26] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, "Scheduled sampling for sequence prediction with recurrent neural networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, (Cambridge, MA, USA), p. 1171–1179, MIT Press, 2015.

[27] Prakhar Mishra, "Word Sequence Decoding in Seq2Seq Architectures," 2019. Available at `https://towardsdatascience.com/word-sequence-decoding-in-seq2seq-architectures-d102000344ad`, Accessed: 2021-06-03.

[28] Renu Khandelwal, "An intuitive explanation of Beam Search," 2021.

[29] Wikipedia, "Beam search." https://en.wikipedia.org/wiki/Beam_search. Accessed: 2021-06-03.
[30] Wikipedia, "Word error rate." Available at: https://en.wikipedia.org/wiki/Word_error_rate, Accessed: 2021-06-03.
[31] Rafael.C.Carrasco, "Text digitisation." Available at: https://sites.google.com/site/textdigitisation/qualitymeasures/computingerrorrates, Accessed: 2021-06-17.
[32] Wikipedia, "F-score." https://en.wikipedia.org/wiki/F-score. Accessed: 2021-06-03.
[33] ABBYY, "Abbyy finereader." http://finereader.abbyy.com. Accessed: 2021-06-03.
[34] Google, "Tesseract OCR." https://github.com/tesseract-ocr/. Accessed: 2021-06-03.
[35] I. S. MacKenzie and R. W. Soukoreff, "A Character-Level Error Analysis Technique for Evaluating Text Entry Methods," NordiCHI '02, (New York, NY, USA), p. 243–246, Association for Computing Machinery, 2002.
[36] R. W. Soukoreff and I. S. MacKenzie, "Measuring Errors in Text Entry Tasks: An Application of the Levenshtein String Distance Statistic," in *CHI '01 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '01, (New York, NY, USA), p. 319–320, Association for Computing Machinery, 2001.
[37] S. R. Eide, N. Tahmasebi, and L. Borin, "The Swedish Culturomics Gigaword Corpus: A One Billion Word Swedish Reference Dataset for NLP," in *Linköping Electronic Conference Proceedings. Digital Humanities 2016. From Digitization to Knowledge 2016: Resources and Methods for Semantic Processing of Digital Works/Texts, July 11, 2016, Krakow, Poland*, (Linköping), Linköping University Electronic Press, 2016.
[38] P. S. Foundation, "python." https://www.python.org/. Accessed: 2021-06-03.
[39] PyTorch, "Pytorch." https://pytorch.org/. Accessed: 2021-06-03.
[40] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi, "The Curious Case of Neural Text Degeneration," 2020.
[41] Z. Fu, W. Lam, A. M.-C. So, and B. Shi, "A theoretical analysis of the repetition problem in text generation," *ArXiv*, vol. abs/2012.14660, 2020.
[42] S. N. Kasturi, "Underfitting and Overfitting in machine learning and how to deal with it!," 2019. Available at https://towardsdatascience.com/underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6fe4a8a49dbf, Accessed: 2021-06-03.
[43] V. Silaparasetty, "How to Handle Overfitting and Underfitting in Machine Learning," 2019. Available at https://medium.datadriveninvestor.com/how-to-handle-overfitting-and-underfitting-470a1f7389fe, Accessed: 2021-06-03.