Alf Niklas Håkonsen Kalmár & Borgar Lie

Robo Journalism

Specialization project, fall 2017

Under the guidance of Massimiliano Ruocco

Artificial Intelligence Group
Department of Computer and Information Science
Faculty of Information Technology, Mathematics and Electrical Engineering

Abstract

Formulating headlines for news articles can be time consuming and difficult. Having a automated way to generate headlines of high quality would be beneficial for most news companies. Another step forward would be to further control the generation in a desired manner.

In this project, we explore the seq2seq model and how this can be used to generate headlines for articles. We explore models that aims to generate qualitatively good headlines, as well as trying to control the generation by injecting categories into the seq2seq context vector. We also explore the effects of scaling the original loss function with a classification score from the generated sequence.

Our models are mostly inspired by the work of [Lopyrev, 2015], [Hu et al., 2017] and [Li et al., 2016]. The dataset used in our experiments is a set of norwegian news articles provided by NTB (the Norwegian news agency).

Preface

This report was written by Alf Niklas Håkonsen Kalmár and Borgar Lie during the autumn of 2017. The project is the final delivery of the Specialization Project (TDT4501) at the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU). We would like to thank Massimiliano Ruocco, Erlend Aune and Eliezer da Silva for the guidance during this project and for providing a good environment for exchanging ideas with them and our fellow students.

Alf Niklas Håkonsen Kalmár & Borgar Lie

December 13, 2017

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this project we will be looking into abstractive text summarization, especially headline generation, and further trying to control the generation by injecting additional data into the model.

1.1 Background and motivation

Having summaries for articles or any kind of text is useful for many reasons. For example, there exist a wide amount of financial data, but most do not really need all of the information provided in these texts, making short and consise summaries a lot more useful. News articles, papers and encyclopedias can be long and time consuming to digest. Often, you are not interested in reading all the material you come by, but only some of it. Summaries can help to get a grasp of what they are about, and may include all the information you need, or help you decide on whether or not to read the entire text.

Generating these summaries is time consuming, because 1) you need to read the text, and 2) you need to understand which parts are most important. This makes it very attractive to try to automatically generate such summaries using a computer program instead of having a human do it. This is something that has been researched for quite a while, and there are generally two different ways of going about it. These are *Extractive text summarization* and *Abstractive text summarization*.

Extractive text summarization focuses on choosing how, e.g. paragraphs and important sentences, produces the original document in precise form. The implication of sentences is determined based on statistical and linguistic features [Moratanch and Chitrakala, 2017]. A summarization that is produced using this method is basically a shortened version of the original document where the least important paragraphs or sentences are removed until a feasible size for a summary is achieved, and only the most important features are left.

Abstractive text summarization is the task of generating a short summary that captures the ideas of the original document. Abstractive, in this case, means that the summary is not a mere copy of a few sentences from the original document, but rather a compressed paraphrasing of the main contents, potentially using vocabulary not seen in the original document [Nallapati et al., 2016a].

Newspapers, among others, have an additional challenge that is related to summaries, that is, they need to provide a fitting headline for their articles. A fitting headline needs to reflect the content, as well as the scope, while being accurate and not having the property that some people might describe as *clickbait*[1]. Additionally, journalists may want to have the headline reflect certain features. One can for instance see differences between a headline meant for a news article about politics vs. a headline regarding a sports article.

1.2 Research goals and questions

Goal Controlling the generation of headlines based on the content of an article and a given category.

The main goal of this project is to explore and implement algorithms for generating meaningful headlines based on the content of an article. Further, we experiment with the incorporation of categories and an additional loss function to see how it affects the training, as well as the generation.

Research question 1 Is the headline generation controllable through the use of an input category, and if so, how does this affect the generation of the headline?

Research question 2 How will the incorporation of an additional loss function influence the training of the model and the final generation?

1.3 Research approach

We start by implementing a baseline for our work based on related articles and research. The baseline is a simple model for generating headlines based on the text of an article. The baseline is important to be able to both see that our model works as intended, generating headline suggestions for articles that makes sense, as well as being a starting point for further research and development.

Baseline results are extensively reported and discussed, both quantitatively and qualitatively, showing both the good and the bad results, showing that the task at hand is challenging, especially with limited amounts of data. The dataset used in our experiments have not previously been used for such a task, and is relatively small compared to other popular datasets[2].

We explore simple extensions to the baseline, merging the article content with the related category, trying to steer the generation towards that particular category. We also do some experiments including an additional loss to our model based on the classification score of the generated headline to see how that affects the training and the generation. The results from our experiments are compared to the baseline, as well as other state of the art results in the same area.

1.4 Contributions

In this project, we explore the seq2seq model and how this can be used to generate headlines for articles. We explore multiple models that aims to generate qualitatively good headlines, as well as

[1]Clickbait is a term used for content whose main goal is to get users to click, while possibly being misleading. `https://en.wikipedia.org/wiki/Clickbait`

[2]Comparison to other datasets is briefly discussed in Section 6.3

trying to control the generation by incorporating categories as input into our models. We present results showing how such models can generate qualitatively good headlines, even with limited amounts of data. All of the code that is used in our experiments is available on github[3]. The README file in the project folder explains how to run the relevant parts. The dataset used in our experiments is not publicly available, but the method is general, and should work for other datasets, given that they are large enough for the model to be able to generalize.

1.5 Report overview

We start by diving into some important background theory, explaining the general concepts and methods underlying the models used in the project. Afterwards, we briefly review the most relevant work in the current state of the art related to our experiments, followed by a explaination of the architectures we have used. Further, we report the results from our experiments, followed by a general evaluation and a discussion around them. Finally, we review possible improvements and discuss relevant future work.

[3]Github repository: `https://github.com/borgarlie/TDT4501-Specialization-Project`

Chapter 2

Background Theory

In this chapter we will cover subjects that are relevant for the report. It serves as a gentle introduction to the different techniques and terminologies used.

The chapter starts off by introducing some basic building blocks. These include deep neural networks, long short-term memory and convolutional neural networks. The chapter concludes with a brief overview of regularization strategies.

2.1 Artificial neural network

Artificial neural networks (ANN) is inspired by networks of biological neurons in the brain. A neuron, as illustrated in Figure 2.1, is a basic information-processing unit, and consist of a cell body, soma, a number of fibres called dendrites and a single long fibre called the axon. One single neuron have very little processing power, but an army of these can outperform todays supercomputers [Negnevitsky, 2005, p. 165-166].

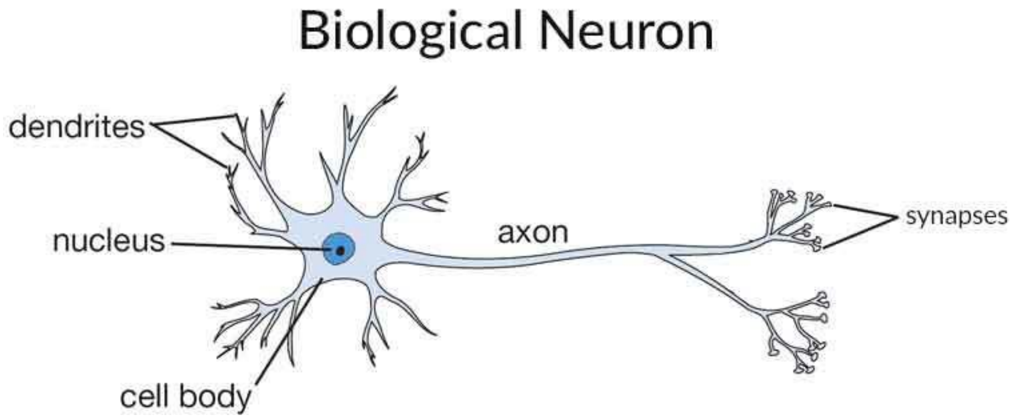


Figure 2.1: Illustrates a biological neuron[1].

[1]Neuron illustration: `https://cdn-images-1.medium.com/max/1600/0*l4ohhbrwQ5MGvmGc.jpg`

An artificial neural network is basically a number of simple processing units like neurons, and these units are connected with weighted links passing signals through the network. Each neuron can receive multiple input signals, but will produce only one output signal which can split to other neurons or be the output for the network [Negnevitsky, 2005, p. 167-168]. Figure 2.2 is a graphic illustration of a typical ANN.

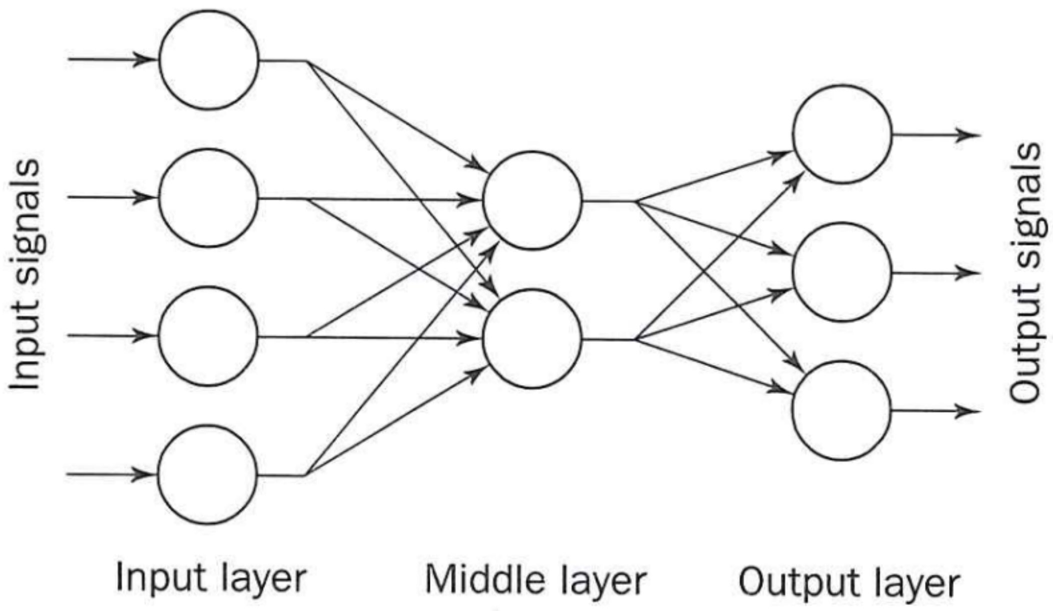


Figure 2.2: Architecture of a typical artificial neural network. Figure is retrieved from [Negnevitsky, 2005, p. 167]

A neuron can be viewed as a simple computing element. It receives several input signals, sums the signals and put the sum through an activation function. If the activation "fires", the neuron will send it as an output signal through the output links [Negnevitsky, 2005, p. 167-168].

2.1.1 Deep neural networks

As mentioned above, a single neuron is not very powerful, but many neurons connected together in several layers can solve quite complex tasks. This type of network is called *deep neural networks (DNN)* or *deep feedforward networks*. Figure 2.3 shows an example off a typical DNN. It consist of an input layer followed by many layers with nonlinear hidden units and lastly an output layer. The layers between the input and output layer is called hidden layers because the data does not show the desired output for each of these layers [Negnevitsky, 2005, p. 175-176].

The goal of a feedforward network is to approximate a function $f*$. The layers in the network can be viewed as a chain of functions that makes up the approximation of the entire function $f*$, such

[2]Deep neural network architecture: `https://www.mathworks.com/content/mathworks/www/en/discovery/deep-learning/jcr:content/mainParsys/band_2123350969_copy_1983242569/mainParsys/columns_1635259577/1/image_2128876021_cop_1731669336.adapt.full.high.svg/1508444613846.svg`

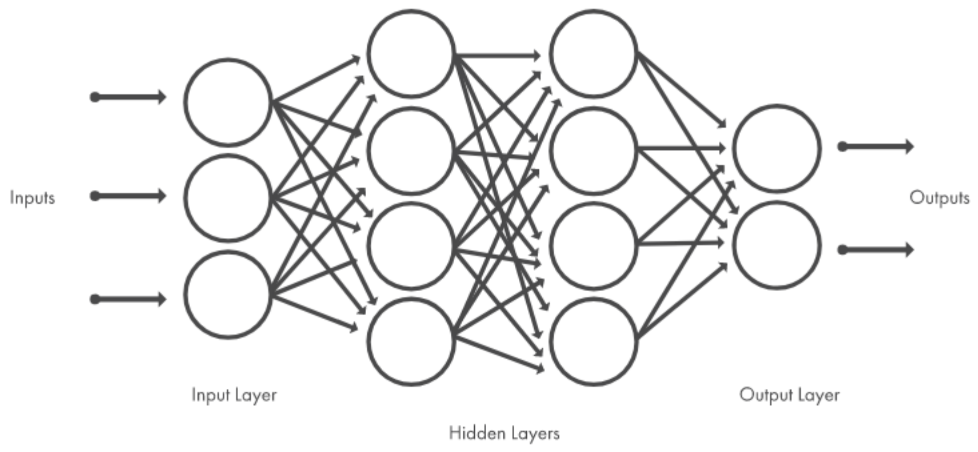


Figure 2.3: Typical architecture for deep neural networks[2].

as $f* = f^3(f^2(f^1(x)))$ [Goodfellow et al., 2016, p. 164].

In a feedforward neural network the input is propagated from the input layer to the hidden layers and in the end produces the output, this procedure is called forward propagation.

For the network to learn the function, the weights that connects the neurons in the network have to be adjusted. This is a straight forward process when there is only one layer, because there is only one set of weights and they are directly responsible for the output error. When the network have multiple layers with nonlinear functions it is not so clear how to adjust the weights.

Luckily there is an easy way to do this. The idea is that every neuron in the network is responsible for the error, and thus the weights have to be adjusted accordingly. This idea is more commonly known as the back-propagation algorithm. The algorithm back-propagate information through the network, and tries to minimize the error function in the weight space by using gradient descent. The information propagated backwards in the network is partial derivatives of the error or the cost function with respect to any weight w (or bias b) in the network [Chauvin and Rumelhart, 1995, p. 1-2][Goodfellow et al., 2016, p. 204-207][Rojas, 2013, p. 161]. Two examples of cost functions, also known as loss functions, are negative log loss (NLL), as seen in Equation 2.1 and binary cross entropy (BCE), as seen in Equation 2.2. In both of these equations, t is the target value, o is the predicted output value and N is number of examples in the minibatch. In Equation 2.1, M is the set of classes.

$$loss(o,t) = -\frac{1}{N}\sum_{i\in N}\sum_{j\in M} t_{ij} * log(o_{ij}) \tag{2.1}$$

$$loss(o,t) = -\frac{1}{N}\sum_{i\in N}(t(i) * log(o(i)) + (1 - t(i)) * log(1 - o(i)) \tag{2.2}$$

2.2 Recurrent neural network

Recurrent neural networks (RNN) is a specialized neural network that operates on time series and sequences of data. Unlike a feedforward neural network, an RNN have connections to itself as shown to the left in Figure 2.4. To the right, the same network is illustrated, only unfolded. It is called recurrent because it performs the same task on every element in the input sequence, and the output is depending on the previous computations [Martens and Sutskever, 2011][Goodfellow et al., 2016, p. 367].

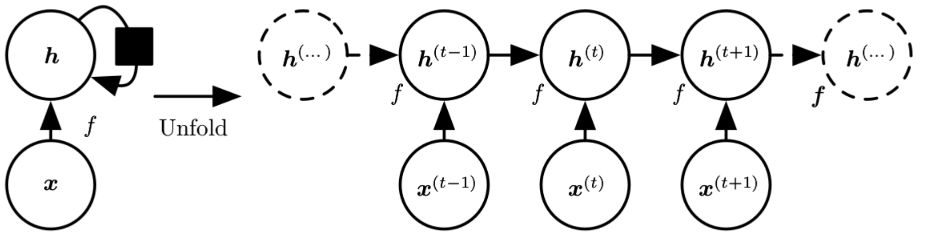


Figure 2.4: The architecture for RNN. The figure is retrieved from [Goodfellow et al., 2016, p. 370]

Since RNN's are doing the same task over and over again, they share the same parameters across time steps. This reduces the number of parameters the network have to learn, and thus more easily converges on the training data [Sutskever, 2013, p. 9-10] [Goodfellow et al., 2016, p. 367].

One advantage a RNN have over a normal feedforward network is that it can operate on different input lengths. This makes it more flexible and a good choice for tasks where the data varies in length, such as natural language processing (NLP) [Goodfellow et al., 2016, p. 367-369].

Another advantage is that the network can retain information from previous steps. Retaining information gives opportunity to exhibit dynamic temporal behavior. This can be thought of as if the network have a memory which captures what have been done so far. In theory a RNN can use this memory for arbitrarily length of sequences, but in practice, this is not the case [Goodfellow et al., 2016, p. 367-369].

Long-term dependencies happens often in RNN's because steps that are far apart can strongly depend on each other. This gives rise to a big problem for RNN's, which is called vanishing/exploding gradient problem. The reason this happens is because as the network receives long input sequences, the gradients propagated backwards often becomes very large or very small, meaning that either the gradients explodes or vanishes [Sutskever, 2013, p. 10-11] [Goodfellow et al., 2016, p. 396-398]. There are some common approaches to solve this issue, and a popular solution is to use long short-term memory (LSTM). This will be covered more in depth in section 2.2.1.

Back-propagation can also be used for training RNN's, but since the network uses the same set of parameters for each time step some adjustments have to be made. The gradient for each output is not only dependent of the current calculations, but also the previous steps. This means that to find the gradients for the parameters, the algorithm have to back-propagate gradients through time, and therefore it is called back-propagation through time (BPTT) [Sutskever, 2013, p. 23] [Goodfellow et al., 2016, p. 374-376].

2.2.1 Long short-term memory

As mentioned above, RNNs struggle with long-term dependencies, and LSTMs are a special kind of RNNs that are capable of dealing with these dependencies. Hochreiter & Schmidhuber [Hochreiter and Schmidhuber, 1997] were the first to introduce this idea in 1997.

LSTMs are designed to deal with these long-term dependencies which gives them the natural behavior of remembering information for long periods of time. LSTMs are similar to normal RNNs in that they both have a chain of repeating modules. The difference is that normal RNNs have only one single layer which gives rise to a very simple structure, but LSTMs have four layers interacting in a complex way[3]. Figure 2.5 illustrates a typical LSTM architecture, and compared to the RNN architecture in Figure 2.4, the differences are quite clear.

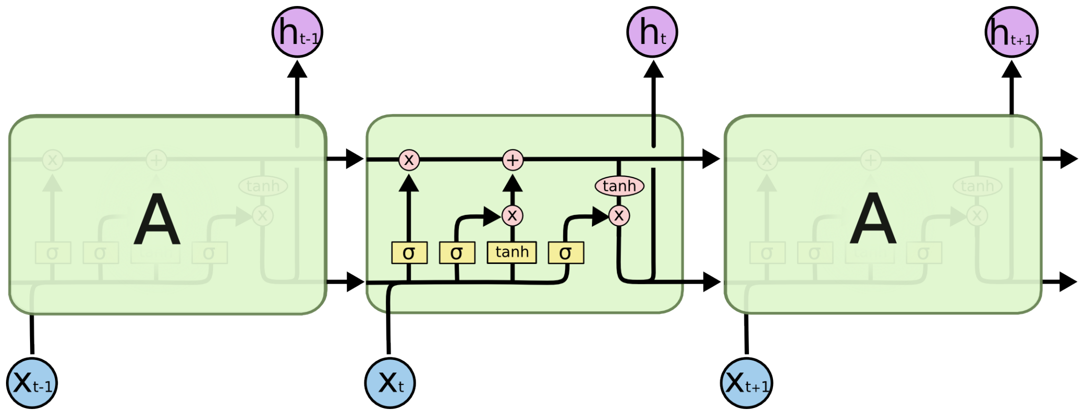


Figure 2.5: The architecture for a LSTM network[4].

The core idea of LSTM's is that they have a cell state. This state allow information to flow easy along the chain unchanged (the top line in the figure 2.5). The module uses gates to control how much information that should be allowed to get through to the next time step. Figure 2.6 illustrate the structure of a gate in a LSTM module. This is typically a sigmoid layer which outputs a number between 0 and 1. This number essentially tell the module how much information to let through, where 0 means nothing at all and 1 means everything [Yu et al., 2015][Gers et al., 1999].

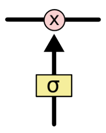


Figure 2.6: A gate in the LSTM architecture[5].

LSTM's have 3 different gates, where all of them are for controlling and protecting the information. The first gate, forget gate layer, is a simple sigmoid layer to decide what information to keep and what to throw away. The next gate, the input gate layer, is the layer which decide what new information that are going to be stored in the cell state. This gate have two parts, first

[3]Colah's blog about LSTMs: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`
[4]LSTM architecture: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-chain.png`
[5]A LSTM gate: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/img/LSTM3-gate.png`

a sigmoid layer that decides which values are going to be updated, and second, a tanh[6] layer that generates a candidate vector that could be added to the cell state. These two parts are then combined before a potential update to the state. Lastly is the output layer, which is based on the cell state. First, a sigmoid layer is used to decide what parts of the cell state is going to be output. Then, a tanh layer is used on the cell state to force the values in the range -1 to 1, and combines it with the sigmoid so the output gets filtered to what is desired [Yu et al., 2015][Gers et al., 1999].

One limitation of regular LSTM is that they are only using the previous context. Processing the data in both directions with two separate hidden layers by two different networks allows the model to access long-term dependencies in both directions. These hidden layers are provided to the same output layer. This approach is called Bidirectional LSTM (BLSTM). In situations where the whole input sequence is available from the beginning there is no reason to not exploit both future context and history context together [Graves and Schmidhuber, 2005][Graves et al., 2013][Yu et al., 2015]. Figure 2.7 gives a clear view of the BLSTM architecture and what information that is available at any given step.

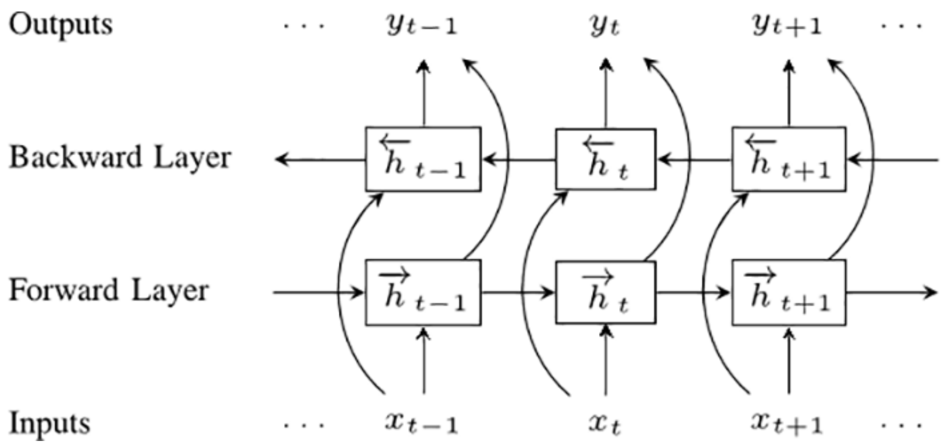

Outputs
$\cdots$ y_{t-1} y_t y_{t+1} $\cdots$
Backward Layer
$\overleftarrow{h}_{t-1}$ $\overleftarrow{h}_t$ $\overleftarrow{h}_{t+1}$
Forward Layer
$\overrightarrow{h}_{t-1}$ $\overrightarrow{h}_t$ $\overrightarrow{h}_{t+1}$
Inputs
$\cdots$ x_{t-1} x_t x_{t+1} $\cdots$

Figure 2.7: Typical architecture for a BLSTM[7].

Gated recurrent unit

Gated recurrent unit (GRU) was introduced by [Cho et al., 2014a], and is a variation of LSTM. The difference can be seen in Figure 2.8, it combines the forget gate and input gate into a single ”update gate”. The cell state and the hidden state is also merged together. This gives rise to a simpler model, and therefore have become a very popular variation to use [Goodfellow et al., 2016, p. 407-408].

[6]Hyperbolic tangent is a trigonometric function, with output values between -1 and 1. `https://en.wikipedia.org/wiki/Hyperbolic_function`

[7]BLSTM architecture: `https://cdn-images-1.medium.com/1%2AbACdupA0mGvIO2ijlUOTbA.png`

[8]The GRU architecture: `https://feature.engineering/difference-between-lstm-and-gru-for-rnns/`

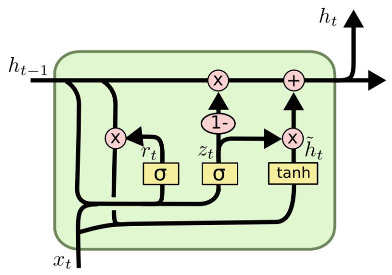


Figure 2.8: A illustration of the GRU architecture[8].

2.3 Convolutional neural network

Convolutional neural network (CNN) [LeCun et al., 1998] as the name suggest gets their name from convolution. A formal definition of convolutional neural networks is given as [Goodfellow et al., 2016, p. 326]:

> convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

CNN have led to many breakthroughs in DNN's in the last few years. CNN's exceed in domains with pattern recognition, such as time-series data, images and voice recognition. A simplistic view of CNN, is that it uses many identical copies of the same neurons. One advantage of having many copies of the same neuron is that large models can be trained while still keeping the actual number of parameters small.

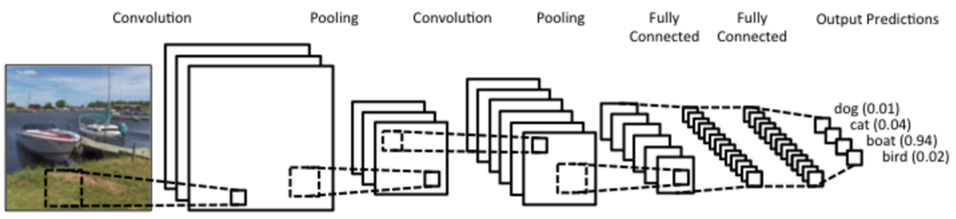


Figure 2.9: A typical convolutional neural network architecture [9].

The explanation above is very simple and leave out many details about the CNN. Figure 2.9 is a more correct view of a typical CNN. A CNN have one or more of the following four operators:

- Convolution
- Non Linearity function(Typically ReLU)
- Pooling or Sub Sampling
- Classification (A fully connected layer)

[9]Convolutional neural network architecture: `http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/`

These four operators are the building blocks of most CNN, thus a good understanding of these is important to truly understand CNN. The primary focus of the convolution step is to extract features from the input. It can be thought of as a fixed grid moving over the input, and doing the same operation everywhere. This operation is typically given like this:

$$s(t) = (x * w)(t) \tag{2.3}$$

The first argument, x, is often referred to as the input, and the second, w, is called the kernel. The output of this operation is often called a feature map [Goodfellow et al., 2016, p. 327-328]. Figure 2.10 gives a clear illustration of the Convolution step. The kernel, upper right in the figure, can be viewed as performing a dot product over the input data, upper left in the figure. Kernel moves over the input with stride steps, until it have covered the whole input data. The reason why the output is called "a feature map" is because it will only give output values if the kernel detects the features it is trained to detect. Thus it is common to use multiple kernels on the same input to detect different features. Figure 2.11 gives a clear understanding of this effect.

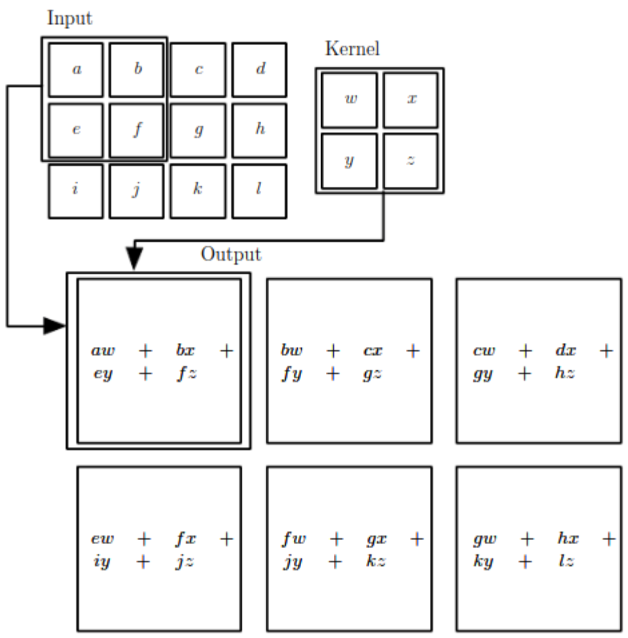
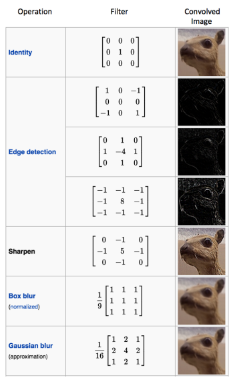


Figure 2.10: Illustrating a 2D convolution. Figure is retrieved from [Goodfellow et al., 2016, p. 330]

The next step in a CNN is to perform a non linear function on the feature maps. The purpose of this function is to introduce non linearity in the network. This is necessary because without a non linear function it is impossible to learn something that is not linear. Most things in the world are not linear, and therefore non linearity have to be introduced in the network. ReLU have been found to give good results because of it's nice derivative properties, which always is one for non negative numbers. Other activation functions could be used as well, such as sigmoid or tanh.

The third step is the pooling step. Spatial pooling reduces the dimensionality of each feature map while retaining the most important information. This can be done with many different functions, such as max, average, sum and so on [Goodfellow et al., 2016, p. 335-336].

[10] Filter detection: `https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/`



Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Figure 2.11: Features detected by different filters[10].

Lastly is the fully connected part. This typically consists of multiple layers with a softmax[11] activation on the output layer. As the name suggest, this layer have connections to every neuron in the previous layer. The neurons from the previous layers include high level features of the input, and the fully connected layer use this to classify the input. It can also be used to learn non linear combinations of the features from the previous layer.

The main motivation to use CNN is sparse interactions, parameter sharing and equivariant representation. It also provides a mean for working with variable sized input.

A traditional neural network will use matrix multiplication on a matrix of parameters, with a separate parameter describing the interaction between each unit. This means that every output unit interact with every input unit. A CNN typically have sparse connections. This is achieved by having a kernel size that is smaller than the input. One advantage of having sparse connections is that fewer parameters are required, reducing the memory requirements and improving the statistical efficiency for the model. The number of operations for computing the output is also reduced [Goodfellow et al., 2016, p. 330-331].

The meaning of parameter sharing is that parameters are used more than once in a model. As mentioned above, the kernel is used over the entire input. This way, the CNN use the same parameters several times in the model. CNN networks only learn one set of parameters instead of separate set of parameters for every location, like typical neural networks do. This will also reduce the memory requirements for the model. Therefore, CNN is several times more efficient than dense matrix multiplications, with respect to memory requirements and statistical efficiency [Goodfellow et al., 2016, p. 331-334].

This form of parameter sharing give the layers a property called *equivariance* to translation. This means that if the input changes, the output have to change in the same way. Thus, the model can produce the same representation of the input, independent of when it receives it in time series or if objects are moved in the input. However, this does not mean it can detect other transformations, such as rotation of objects or a change in scales for a picture [Goodfellow et al., 2016, p. 334-335].

[11]Softmax is a function that squashes the values into a vector with values between zero and one, that sums up to one.

2.4 Regularization

Machine learning problems usually train on a dataset that is a subset of the whole domain. The goal is to learn the representation of this dataset. Overfitting is a term that says a network learns a representation too well, and it exist another representation that performs better on the entire distribution of data [Mitchell, 1997, p. 67]. I.e. the network does not generalize well. Generalization is a term that says how well a network performs on unseen data.

To overcome the issue of overfitting, many strategies have been developed to improve the performance on unseen data. Strategies that aims to achieve this behavior is called regularization.

Goodfellow et al defines regularization in their *Deep Learning* book as [Goodfellow et al., 2016, p. 117]:

> Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

Some regularization strategies puts extra constraints and penalties on the machine learning model. These constraints and penalties might be designed to incorporate prior knowledge, or they can express a preference for simpler models. Another example of regularization is the use of parameter norm penalization [Goodfellow et al., 2016, p. 226]. This regularization approach aims at limiting the capacity for the model, adding a penalty $\Omega(\Theta)$ to the loss function, which leads to the new function displayed in equation 2.4:

$$\hat{L}(\Omega, C) = L(\Omega, C) + \alpha\Omega(\Theta)^{12} \tag{2.4}$$

L^1 and L^2 regularization is one of the most well-known methods of parameter norm penalization. They are very much alike, the difference is that L^2 is the sum of squared weights across the whole network while L^1 is the sum of absolute values [Goodfellow et al., 2016, p. 227-233].

Even though they seem very similar, they can have different effects. L^1 can have a sparsity effect, because it drives many of the weights to zero. This happens if most of the weights have small absolute values. L^2 can have a more stable update scheme, because the penalty scales linearly with the weights [Goodfellow et al., 2016, p. 227-233].

2.4.1 Dropout

Dropout is also a well-known regularization strategy. It provides a computationally efficient, but yet a powerful method to regularize a model. One way to look at it, is that it works as a bagging method. Bagging is an ensemble method that combines multiple hypotheses to achieve better performance on new inputs. Bagging does this by training multiple models and combine these models (usually by majority vote) to get a prediction on each test example. Dropout effectively removes some units of the network by multiplying the output with zero [Goodfellow et al., 2016, p. 255]. This will in a way result in training multiple networks, as illustrated in Figure 2.12.

The term *dropout* refers to dropping out some units in the network. In the section above it was done by multiplying the output with zero, but any operation that drops the unit can be used. For

[12] Slightly altered equation from [Goodfellow et al., 2016, p. 226] for easier understanding. regularized loss function ($\hat{L}$), L = loss function, C = cases, Θ = parameters of the estimator, Ω = penalty function and α = penalty scaling factor

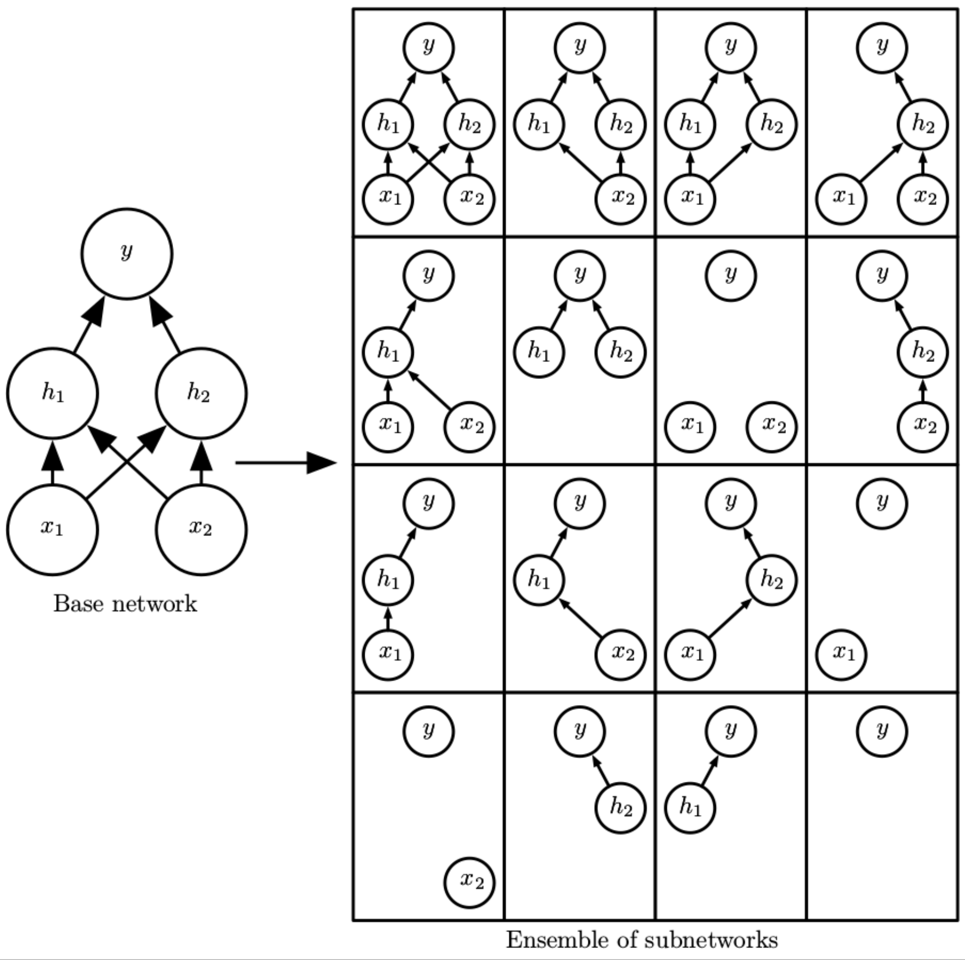


Figure 2.12: Visualization of the dropout functionality. Retrieved from [Goodfellow et al., 2016, p. 256].

every run, which units that get dropped is random, and is usually determined by a probability, p. Dropout is not exactly like bagging, because in bagging you train several independent models, but dropout shares the same parameters. Parameter sharing makes it possible to represent a large number of models with reasonable amount of memory [Goodfellow et al., 2016, p. 255-257] [Srivastava et al.,].

There exist multiple other regularization techniques, such as data augmentation, multitask learning, early stopping, sparse representations and adversarial training, but for this project, only a general understanding of the regularization concept and the methods mentioned in this chapter will suffice.

Chapter 3

State Of The Art

In this chapter we will be looking into state of the art regarding headline generation and controllable text generation, as well as multilabel classification, as our baseline, proposed model and results are based on this.

3.1 Headline generation

There is a lot of work done in the field of abstractive text summarization, but we will mainly focus on sequence to sequence models used in NLP tasks as they seem to have given the best results regarding headline generation thus far.

Deep neural networks has traditionally done very well on both regression tasks as well as classification tasks, but could not be used in the traditional way to map one sequence to another sequence. To cope with this problem [Sutskever et al., 2014] proposed[1] a model which usually is called *Sequence 2 Sequence* or *seq2seq*. This model uses a multilayered LSTM to map a input sequence to a vector of fixed dimentionality (referred to as the *Encoder*), and then another multilayered LSTM to decode the target sequence from this vector (referred to as the *Decoder*). [Sutskever et al., 2014] achieved state of the art performance on the machine translation task using this model. The standard seq2seq model is shown in Figure 3.1. The model reads an input sentence ”ABC” and produces the output sentence ”WXYZ”. The model stops predicting new words after outputting the <EOS> token.

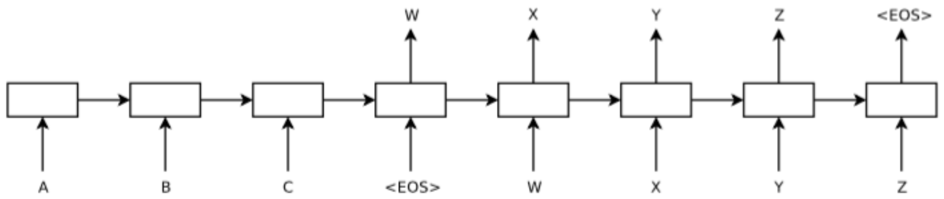


Figure 3.1: The seq2seq model as presented in [Sutskever et al., 2014]

[1]It seems multiple research groups had the same idea about the same time, [Cho et al., 2014b], among others, also used a similar architecture

The seq2seq model was further improved by [Bahdanau et al., 2014] using what is known as an *attention mechanism*.

The most important difference of this approach compared to the basic seq2seq model is that it does not attempt to encode the entire input sequence into a single fixed-length vector. Instead, it encodes the input sequence into a sequence of vectors and chooses a subset of these vectors to focus on while decoding the output. The attention mechanism basically allows the decoder to search for a set of positions in the source sequence where the most relevant information is located.

The calculations behind the attention mechanism are quite straight forward. We start by calculating a set of attention weights. This is done with a standard feed forward layer, using the decoder's input as well as the hidden state of the decoder as inputs, followed by a softmax. These weights are then multiplied by the encoder output vectors to create a weighted combination. The attention mechanism is illustrated in Figure 3.2.

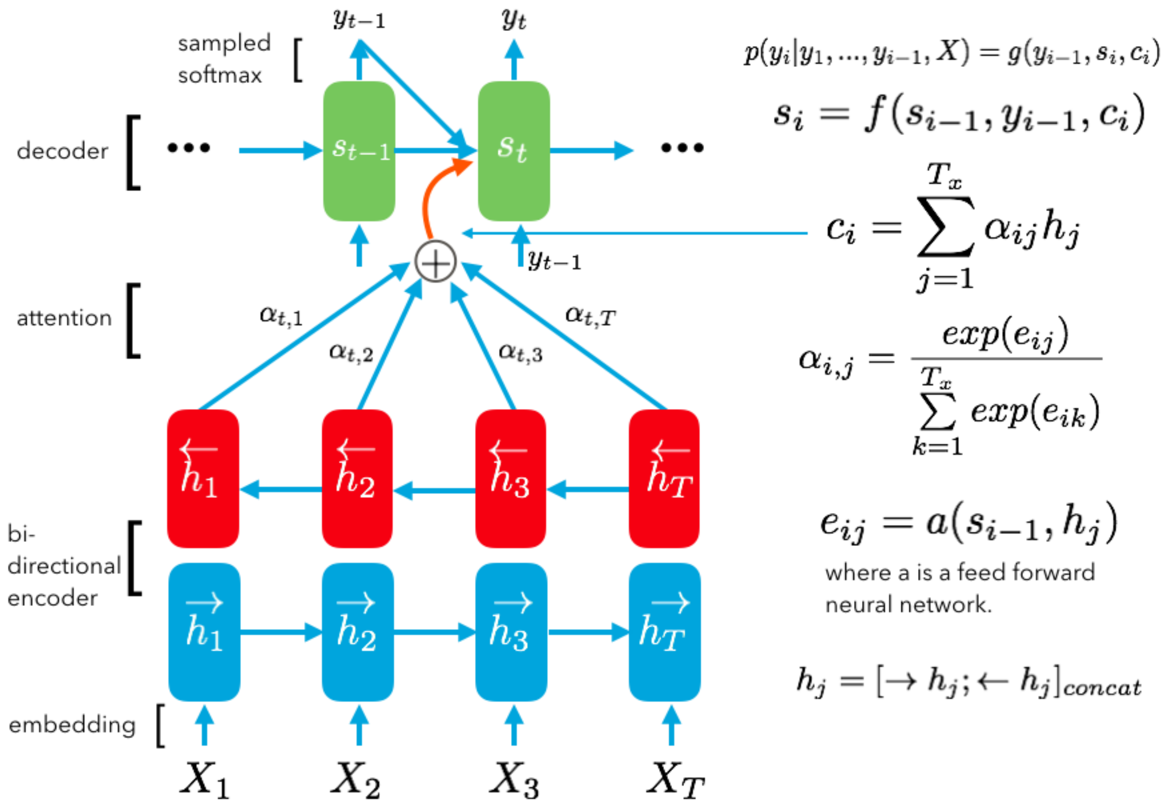

sampled softmax
decoder
attention
bi-directional encoder
embedding
y_{t-1}
y_t
s_{t-1}
s_t
y_{t-1}
$\alpha_{t,1}$
$\alpha_{t,2}$
$\alpha_{t,3}$
$\alpha_{t,T}$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
X_1 X_2 X_3 X_T
$p(y_i|y_1,...,y_{i-1},X) = g(y_{i-1},s_i,c_i)$
$s_i = f(s_{i-1},y_{i-1},c_i)$
$c_i = \sum_{j=1}^{T_x} \alpha_{ij}h_j$
$\alpha_{i,j} = \frac{exp(e_{ij})}{\sum_{k=1}^{T_x} exp(e_{ik})}$
$e_{ij} = a(s_{i-1},h_j)$
where a is a feed forward neural network.
$h_j = [\rightarrow h_j; \leftarrow h_j]_{concat}$

Figure 3.2: The attention mechanism[2]

The seq2seq model was mainly used for machine translation, and the same yields for the attention mechanism. However, [Lopyrev, 2015] used the seq2seq model to generate headlines, achieving very promising results on the GIGAWORD[3] dataset when only keeping articles below 50 words.

[Paulus et al., 2017] showed how this method could be applied to longer sequences by combining

[2]Attentional interfaces: `https://theneuralperspective.com/2016/11/20/recurrent-neural-network-rnn-part-4-attentional-interfaces/`

[3]GIGAWORD: `https://catalog.ldc.upenn.edu/ldc2012t21`

standard supervised word prediction and reinforcement learning (RL), producing state of the art results on abstractive summarization. [Paulus et al., 2017] also uses intra-temporal attention, preventing the model to attend to the same parts of the input on different steps during decoding. This is shown by [Nallapati et al., 2016a] to reduce the amount of repetitions when using attention to generate summaries on long documents.

Both [Lopyrev, 2015] and [Paulus et al., 2017] used what is known as *teacher forcing* to speed up the training process. Instead of generating a word and using that word as input when generating the next word, we use the word from the target headline as input during training. This process is known to speed up convergence, but can also lead to a disconnect between training and testing. To overcome this disconnect, we can choose a ratio as to how often we want to use the target word vs. the generated word. This approach generally seems to work well. In [Lopyrev, 2015], they used a 0.9 ratio (using the target word 90% of the time, and the generated word 10% of the time).

3.2 Controllable text generation

There exist a lot of work involving generation of text, and results are pretty decent. The problem with todays solution is that they only generate good sentences with correct context, but there is no way to control the generation. In many scenarios one might wish to provide some sentiment or other attribute to control the overall attitude or direction for the text generation. This is often mentioned as *controlled generation of text*, and is very challenging. The main goal is to generate realistic and natural sentences, where attributes can serve as a guide for the generation.

In [Hu et al., 2017] they propose a neural generative model which combines variational auto-encoders (VAEs) and holistic attribute discriminators, as shown in the Figure 3.3. The model produces sentences with desired attributes of sentiment and tenses.

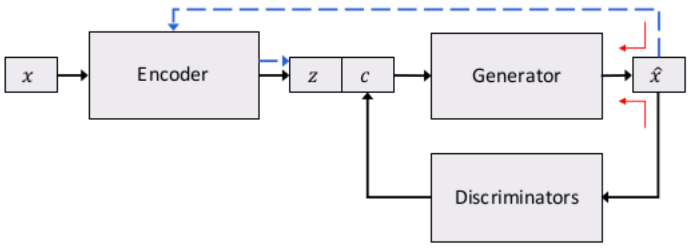


Figure 3.3: The combined model, where z is the context vector produced by the encoder and c is structured code representing attributes that helps controlling the text generation. [Hu et al., 2017]

They first train their base VAE on a large corpus of unlabeled sentences, with a latent code c. The full model is then trained by alternating the optimization of the generator and the discriminator. The effective combination enables discrete latent code, holistic discriminator metrics and efficient mutual bootstrapping.

Discriminators for different attributes can be trained independently on separate labeled sets. I.e, the model does not require a sentence to be annotated with all attributes, but instead needs only independent labeled data for each attribute. They show that a little supervision is sufficient for learning structured representations.

Using this approach, they managed to get meaningful generation with restricted sentence length and improved accuracy on sentiment and tense attributes. Table 3.1 shows some good examples of how this simple change of attribute alters the generation.

Varying the code of tense	
i thought the movie was too bland and too much	this was one of the outstanding thrillers of the last decade
i guess the movie is too bland and too much	this is one of the outstanding thrillers of the all time
i guess the film will have been too bland	this will be one of the great thrillers of the all time

Table 3.1: An example of generated sentences by varying the tense code that is injected to the context vector [Hu et al., 2017]

They also show that their model provides an interface that connects the end-to-end neural model with conventional structured methods. For instance they could encode structured constraints on the latent code to incorporate human intentions.

In [Li et al., 2016], they experiment with injecting more information into the context vector. They inject the target with a speaker-level vector representation. This method can be thought of as teacher forcing, but they use it to ”force” the text generation towards the speaker-level behavior. Figure 3.4 shows how they inject the speaker vector into the seq2seq model. It also shows a *word embedding* graph for time step 2. It illustrates that ”Rob”, which is the persona vector for this example, is clustered around people who often mentioned England in the training data, thus the generation of token *england* is more likely than *u.s.* This shows how they control the generation by inject the model with some extra information.

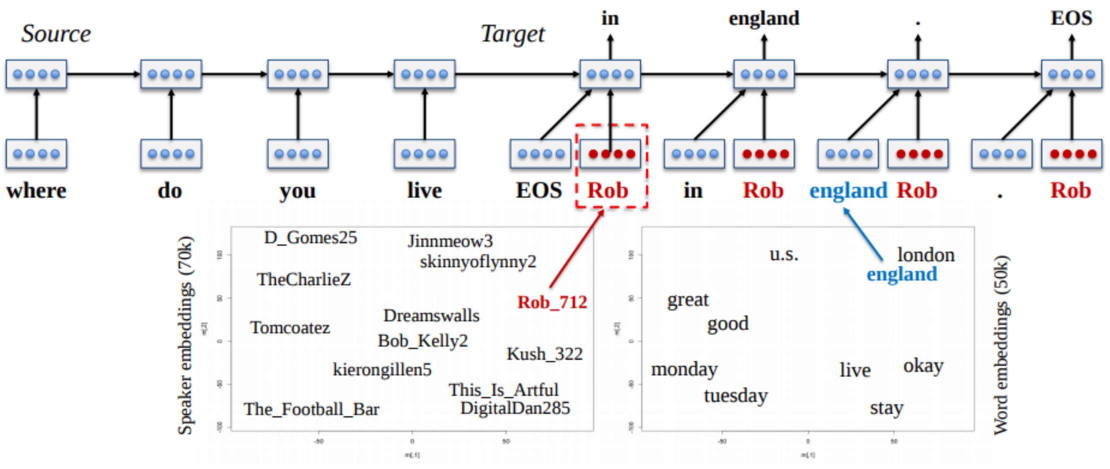


Figure 3.4: Illustrating the vector injection and how it affects the network. [Li et al., 2016].

Their proposed model operates as follows: It tries to predict how speaker A would respond to a message produced by speaker B. They try to aid this prediction, by first encode message S into a vector representation $H(s)$ using the encoder. Then for each step in the decoder, hidden units are obtained by combining the representation produced by the decoder at the previous time step, the word representations at the current time step, and the speaker embedding. In this way, speaker information is encoded and injected into the hidden layer at each time step and thus helps predict personalized responses throughout the generation process.

Their proposed models show some improvement over the standard seq2seq model in terms of BLEU, perplexity, and human judgments of speaker consistency. It is worth mentioning that there is no extra information added to the loss calculation (at least not mentioned in the paper). This illustrates how the text generation can be improved by only injecting some extra information to the context vector.

3.3 Multilabel classification

Multilabel classification is a generalization of multiclass classification. The problem is to map the inputs X into a fixed-length vector Y, where every element in Y is either 0 or 1 depending on whether or not X fits that particular label.

Traditional approaches to this problem includes boosting, k-nearest neighbours, decision trees, kernel methods and neural networks (BP-MLL[4]). Regarding sentence level multilabel classification, [Kim, 2014] has achieved state of the art results using pretrained word vectors and CNNs. The model is especially attractive because of its simplicity[5].

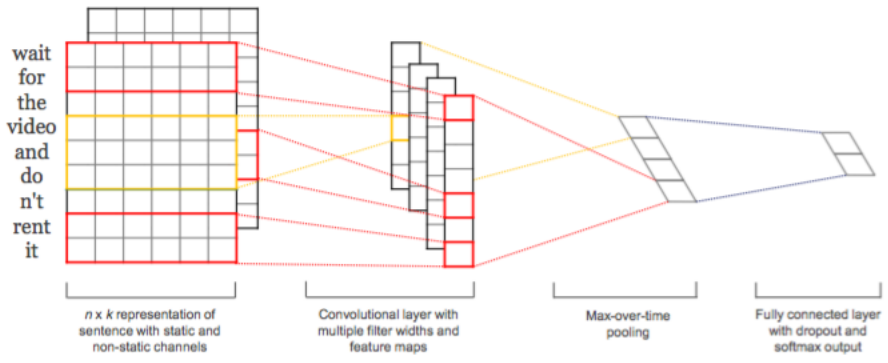

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 3.5: CNN classification with two channels for an example sentence as presented in [Kim, 2014]

The model, as shown in Figure 3.5, takes an input X that is a concatenation of the words in the sentence. These words are embedded using some word-embedding scheme[6], and put through a convolution layer with multiple filter widths, producing multiple feature maps. A max-pooling layer is used to locate the most important features. Finally, a fully connected softmax layer is used to output the probability distribution over the possible labels.

[4]BP-MLL is a back-propagation algorithm for multilabel learning [Zhang and Zhou, 2006]
[5]There are already several implementations of the architecture in the paper
[6]In the paper, they use Word2Vec `https://code.google.com/archive/p/word2vec/`

Chapter 4

Models

In this chapter we explain our baseline model, which is based on the state of the art regarding headline generation. We also elaborate on our proposed model for controlling the generation, using categories as additional input, as well as using an additional loss function in our network.

4.1 Baseline

The baseline is the first model used in our experiments to generate headlines. It is important to get some proper results here, before we can dig into actually controlling the text. When we go further, it will be with some kind of modification and/or extension to the baseline. The baseline is also useful for comparing with other state of the art results when it comes to abstract text summarization and headline generation.

The baseline for our experiments is based on a seq2seq encoder-decoder architecture with attention, as described in Section 3.1.

To calculate the loss, we use the negative log loss function averaged over a minibatch. We use Adam[1] as our chosen optimizer with 0.001 learning rate and a 10^{-5} L2 penalty.

Instead of using LSTMs we chose to use GRUs, as they generally have the same behaviour, but are faster to compute because of the shared gates and states as described in section 2.2.1. The encoder is also bidirectional, providing even more information to the decoder and the attention mechanism.

In the näive implementation of this architecture, the top1[2]-scoring word from the softmax output is appended to the sequence at each timestep. To hopefully generate better sequences, we implemented beam-search when decoding the headline during evaluation (not during training). This is also done in [Sutskever et al., 2014] and [Lopyrev, 2015].

Our beam-search implementation was quite standard. We chose 3 parameters: k_1, k_2 and k_3. (k_1) How many beams should be created from the previous beams. (k_2) How many beams should be kept for each iteration. (k_3) How many beams should we finally output. The beams were extended with the k_1 top scoring words from the softmax function after running the current beam as input

[1]Adam is an adaptive learning rate optimization algorithm, which computes individual adaptive learning rates for different parameters, using estimates of first and second moments of the gradients [Kingma and Ba, 2014].

[2]Hereafter, we will use *topX* to refer to the topX-scoring word(s) from the softmax output, where $X \in \mathbb{N}$.

to the decoder. We then pruned these beams keeping only the k_2 top scoring beams, where the score was computed as the average sum of the logs of the softmax outputs for each word in the sequence this far (as shown in Eq. 4.1), giving us an average score per beam. We finally output the k_3 top scoring beams when all beams had either reached an <EOS>[3] token or maximum length. In our experiments we chose k_1 = 3, k_2 = 20 and k_3 = 5. We chose these parameters based on computational efficiency, while providing a sufficient search space for the generation.

$$\frac{1}{N}\sum_{word_i}^{N} ln\left(score\left(word_i\right)\right) \tag{4.1}$$

The qualitative results shown in Chapter 5 are presented with the top 5 scoring sequences.

4.2 Controlling the text

Generating headlines and abstract summaries from text has been researched for quite a while, but controlling the generation of text is something that has not been done as much. In the interest of providing different outputs from our generation as well as the possibility to improve the quality of the generated sequences, we here present some extensions to the baseline model. Inspirations are taken from [Li et al., 2016] and [Hu et al., 2017].

In short, we add two additional components to our baseline model. First, we allow an additional binary encoded attribute as input to our network, which we feed to the decoder at every timestep. Secondly, we take the entire sequence produced from our network and feed it to a pretrained multilabel classifier and calculate the loss between the input attributes and the outputs from the classifier. Here, we assume that the pretrained classifier is able to discriminate between the different input attributes in any sentence. The proposed model is presented in Figure 4.1, and is thoroughly explained below.

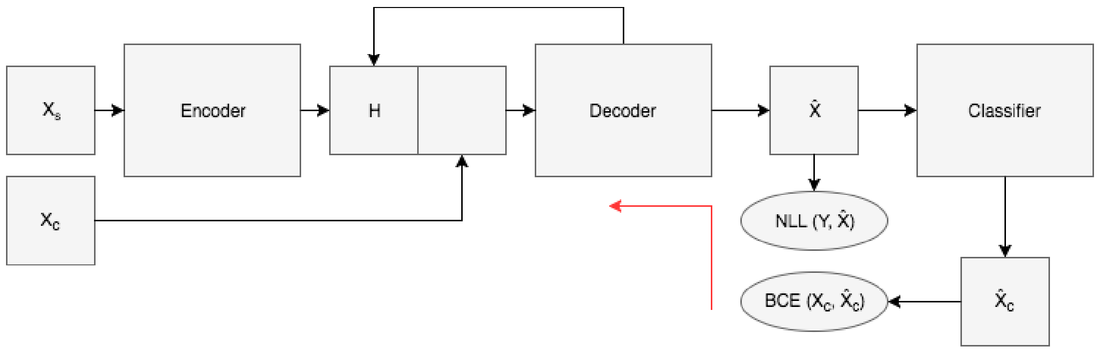


Figure 4.1: Our proposed model. X_s is the input sequence and X_c is the corresponding attribute. The combination of the two losses, NLL (Y, $\hat{X}$) and BCE (X_c, $\hat{X}_c$), is not included in the figure. The red arrow denote gradient propagation.

Specifically, the additional input to the network can be any set of attributes, e.g. Sentiment (positive vs. negative), Categories (Sports, Politics, Finance) or Quality (Good vs. Bad). The only requirement is that we are able to pretrain a classifier to discriminate between the attributes

[3]Token for End of Sentence

in any sentence with good precision. This additional input is fed to the decoder at each timestep, merging with the current input and hidden state. This concatenation is similar to what is done in [Li et al., 2016]. The attention weights are calculated as in Figure 3.2, except that we do the concatenation before the linear layer, making the additional input influence the attention weights, as shown in equation 4.2. H_{i-1} is the previous decoder hidden state, $\hat{X}_{i-1}$ is the previous output from the decoder and X_c is the input attribute.

$$e_{ij} = a(concat(H_{i-1}, \hat{X}_{i-1}, X_c)) \tag{4.2}$$

The decoder produces a token each timestep, ending with a <EOS> token. During training, this decoded sequence (without the <EOS> token), which is our suggested headline, will be used as input to the pretrained classifier. The classifier will output a score for each label, describing the confidence that the headline belongs to this label. We further compare the classifications with the input attributes for the same text using a binary cross entropy loss function. This new classification loss is used to scale the original loss from the generation of the headline, propagating even more information through the network. In Chapter 5 and 6 we experiment with linear-scaling, and in Section 6.3, we briefly mention how the classifier could potentially be used in a reinforcement learning framework.

The proposed model could be viewed as a simplification of the model proposed in [Hu et al., 2017]. Given that their model is very complex[4], this model could work as an alternative to achieving some of the same desired results. The proposed model is also different to the model proposed in [Hu et al., 2017] in the sense that it directly trains the network to maximize a target sentence as well as an additional function, as a form of multi-task learning, without relying on a structured non-entangled code C.

[4]The model presented in [Hu et al., 2017] has a lot of details, without being described fully in the paper, and there is no open source code available

Chapter 5

Experiments and Results

In this chapter we will present our experiments and results. We start by explaining the details behind the dataset that we use, and what kind of preprocessing we do with this dataset. Following is a brief overview of the experimental setup. Finally, we present our obtained results.

5.1 Dataset

The dataset we use is a selection of $\sim$132 000 articles from NTB[1]. These are articles labeled with one or more of the following categories:

- Sport
- Økonomi og næringsliv
- Politikk
- Kriminalitet og rettsvesen
- Ulykker og naturkatastrofer

We lower cased all letters and removed all kinds of symbols except letters between a and z, numbers and the symbols [.,?!æøå]. Numbers were replaced with the symbol "#" per number. We further tokenized all the words, separating punctations from the text so that each punctation count as a single token.

We used a minimum article length of 25 tokens, a minimum headline length of 4 tokens, and a maximum article length of 80 tokens, where we cropped the articles at 80 tokens and backtracked until the last end of sentence. This seems reasonable as most of the content relevant for the headline seems to be captured in the first few sentences in most of the articles. Example (after cropping and tokenization):

Headline: regjeringen vil videreføre dagens ulvebestand

[1]NTB: https://www.ntb.no/

Article: regjeringen foreslår å beholde dagens bestandsmål for ulv og vil bare gjøre mindre endringer i ulvesonen . det går fram av stortingsmeldingen om ulv som klima og miljøminister vidar helgesen h la fram fredag . dagens mål er å ha tre helnorske ulvekull årlig innenfor ulvesonen , og regjeringen foreslår å videreføre dette , men legger også fram et alternativ .

Removed part: det innebærer et mål om fem til åtte familiegrupper , hvor også ulver som lever i grenseområdene mellom norge og sverige er inkludert . regjeringen foreslår at deler av områdene vest for glomma i dagens ulvesone tas ut , mens et område nord for dagens sone inkluderes . samlet sett reduseres sonen med #,# prosent . helgesen sier norge er forpliktet til å ha ulv gjennom internasjonale konvensjoner . dette er et svært vanskelig spørsmål der vi må kombinere hensynet til beitenæringen og hensynet til rovvilt , sier helgesen i en pressemelding .

It is easy to verify with this simple example that the cropped part has next to no influence on the headline, and can be disregarded completely. Similar cropping is done in [Chen et al.,], [Sutskever et al., 2014] and [Lopyrev, 2015]. This cropping reduces the computation time significantly as well as reducing the long term dependencies which often is problematic with long sequences, as discussed in Section 2.2.

The vocabulary of the entire set of articles was above 200 000 when using one-hot vectors[2]. This includes both miss-spelling and dialect. This makes the softmax computations very slow and will also have impact on the learning, as the tokens that are rare would not get many updates as to when they should be used. To cope with the huge vocabulary size, we use a common approach of replacing rare words with a special token <UNK>[3]. We then removed articles consisting of more than 10% <UNK> tokens. The remaining vocabulary was 28480 tokens.

One minor problem with the NTB dataset is that roughly 20% of the articles are duplicates in nynorsk. We could look at this as some sort of augmentation, but in reality it means the total amount of data will be less diverse, and may reduce the generalization quality.

Example of two headlines where 1) is in bokmål and 2) is in nynorsk from the dataset:

1. får ikke lov av staten til å si opp prester
2. får ikkje lov av staten til å seie opp prestar

There were also some cases of non-construtive articles, which we simply removed (These are typically very similar). Example:

Headline: tippetips fra ntb spilleomgang søndag , ## fakta

Article: denne søndagens kupong spilleomgang ## formtabeller #. divisjon menn engelsk #. divisjon fem siste kamper toppscorere #. divisjon menn mads stokkelien , stabæk ## oddbjørn skartun , bryne ## jo sondre aas , ranheim ## jean alassane mendy , kristiansund ## robert stene , fredrikstad # abdurahim laajab , bodø glimt # alexander tveter , follo # ole kristian langås , ull .

We removed the copyright ending of the articles as well as the start of the article when it included information about who had written the article and where, as these attributes has no value in our experiments.

[2]A one-hot vector is a group of bits where only one bit is "hot" (1), e.g. "00100".

[3]In our implementation, we actually replaced the rare words with <UNK+first letter of the word>, so that it would make slightly more sense when reading and evaluating, as well as reducing the bias towards the word <UNK> a bit, as there now will be about 30 different UNK tokens

To make our model able to conclude when to not generate more words we appended an end of sentence token (<EOS>) to the articles and headlines. To make it possible to train in batches we also appended padding tokens (<PAD>) to the data to make all the articles have the same length and all the headlines have the same length. This would not seem to be a problem for the loss function, as it would generally not add extra loss, but rather just decrease the average loss across the sequence, which can be compensated for by using a higher learning rate or training for more epochs.

When running our experiments we split the data into a training set, throwaway set, and a evaluation set. We split it to about 90% for training and 10% for evaluation. The size of the throwaway set was set to 1000 articles (1 week of data) and was used to remove possibly similar articles in the training and evaluation set[4]. We chose these 1000 articles as the 1000 articles between the training and evaluation set sorted by timestamp, where we use the newest articles in the evaluation set.

The categories included in the dataset are used as input attributes for our proposed model. The categories are encoded as a binary vector. E.g. the following article has the category "Ulykker og naturkatastrofer", which produces the binary vector "00001". If the article also included the category "Sport", the binary vector would be "10001".

Article: minst ## personer omkom da en skolebuss kolliderte med en søppelbil i pakistan onsdag . ## av ofrene er barn . ulykken skjedde nær byen <UNKn> , som ligger rundt ## mil nord for karachi . barna hadde vært på en <UNKs> og var på vei tilbake da <UNKs> og <UNKs> kolliderte front mot front . ## er døde , og ## av de er barn .

5.2 Infrastructure

We run our experiments on a single Tesla P100-PCIE-16GB that is provided by Telenor AI Lab NTNU. All the different models presented in this project is written in PyTorch[5]. PyTorch is a deep learning framework for Python. We also used Tensorboard [6] to log all our experiment results. Tensorboard is a visualization tool provided by tensorflow, and can be used to visualize all sorts of metrics.

5.3 Proceedings

We will start by presenting results from our baseline, followed by results obtained with the proposed model. The baseline results are presented with training and evaluation loss, BLEU score, accuracy and attention weights, as well as qualitative results. The proposed model is presented with the same measures, as well as some additional results where we change the input attribute on the same article. In addition we will present accuracy, recall, precision and F1 scores for the classifier used in the proposed model.

All measures presented below is done on the same evaluation corpus, and all values is an

[4]Articles produced in a close timespan could be about the same events, giving a possibility of overlap in training and evaluation set [Lopyrev, 2015]

[5]PyTorch homepage: `http://pytorch.org/`

[6]TensorBoard homepage: `https://www.tensorflow.org/get_started/summaries_and_tensorboard`

average over the whole evaluation corpus. Accuracy measures how close the predicted set is to the true set of labels. Precision and recall are standard information retrieval measures [7]. Precision says how many of the retrieved instances are relevant. It is given as: $Precision = \frac{|(Relevantset) \cap (Retrievedset)|}{|(Relevantset)|}$. Recall says how many of the relevant instances are successfully retrieved. It is given as: $Recall = \frac{|(Relevantset) \cap (Retrievedset)|}{|(Retrievedset)|}$. F1 score is commonly used as test accuracy when dealing with binary classification [8]. It uses both precision and recall to calculate an average harmonic score, formally given as: $F1 = 2\frac{Precision * Recall}{Precision + Recall}$. Accuracy, precision, recall and F1 results presented in this project is acquired with the *sklearn metric* API[9].

Bilingual evaluation understudy (BLEU) [Papineni et al., 2002] is an algorithm used for evaluating machine translations. The central idea behind BLEU score is to measure how close the machine translation (candidate) is to a professional. Therefore, one measure the translation against one or multiple human references. The metric tries to match n-grams of the candidate with n-grams from the references. It only count matches, so the n-grams are position independent, and more counts it gets the better score is given to the candidate. These scores are then averaged over the whole test corpus, to give an estimate for the overall quality of the translation. All BLEU calculation given below is done with *nltk translation BLEU score* API, which is based on the work of [Papineni et al., 2002]. To calculate the BLEU score, we divide the article into multiple sequences (one per sentence), and use them as individual references, as well as using the actual headline as a reference. This way we manage to create multiple relevant references per generated headline.

5.4 Hyperparameters

We experimented with a lot of different hyperparameters, as well doing changes to the model. Some changes had minor impact, and some had a larger impact. Our findings are briefly summarized and described here.

Using a bidirectional encoder, as well as switching to the Adam optimizer instead of using standard stochastic gradient descent had a pretty large impact on the results. We saw that the model converged faster and generated more meaningful sentences in about half the number of epochs combining these two techniques. We can also note that using multiple layers in the encoder and decoder actually had a visible impact on the results before changing to a bidirectional encoder and using Adam, but after doing this change, there were generally no improvements when using multiple layers.

The learning rate was very important when using the Adam optimizer. If we had a learning rate above 0.005 it would not even start converging, and using a learning rate that was lower than 0.0005 would make the optimization very slow. The best results were seen with 0.001 learning rate.

The batch size, weight decay, dropout and teacher forcing ratio had quite low impact in themselves. Generally, when optimized, all of these parameters would improve the results slightly, although, mostly on the convergence rate. When having all of them optimized together, we could see quite

[7]Precision and recall formulas: `https://en.wikipedia.org/wiki/Precision_and_recall`

[8]F1 score: `https://en.wikipedia.org/wiki/F1_score`

[9]Sklearn metric documentations: `http://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics`

large improvements. We can also note that when using a batch size of 1, the model would generally not converge very well, and in many cases the training loss would start to increase a lot after some epochs.

The hidden size did matter quite a bit. With a very large hidden size (e.g. larger than 512), the model starts to fit the data all too well, and the results on the evaluation set are a lot worse compared to results on a smaller hidden size. Generally, everything between 64 and 512 seems to work pretty well with the size of our training set, where 128 and 256 seemed to generalize best.

The number of epochs did not seem to matter too much. We can see a steady decrease in loss for the training data, even after 40 epochs, but after about 20 epochs, it does not seem to generate any better results on the evaluation set.

5.5 Baseline results

In this section, we will present the results from our baseline. The results presented use the hyperparameters shown in Table 5.1. Training on the infrastructure mentioned in section 5.2 with the parameters from Table 5.1 for 20 epochs takes between 2 to 4 hours, depending on whether or not the server (specifically, the GPU) is busy with other tasks at the same time.

Hyperparameter	value
Batch size	32
Learning rate	0.001
Weight decay	0.00001
Dropout	0.1
Hidden size	128
Teacher forcing ratio	0.9
Epochs	20

Table 5.1: Hyperparameters

Figure 5.1 illustrates both training and evaluation loss development during the training. The training loss is calculated as an average for each epoch, and the evaluation loss is calculated over the entire evaluation data set after each epoch.

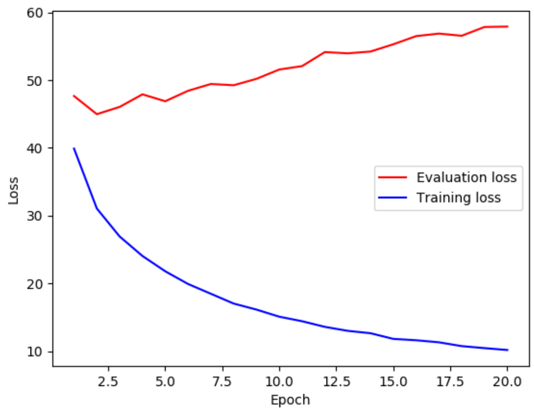


Figure 5.1: Both training and evaluation loss for the baseline.

Table 5.2 shows the BLEU scores for our baseline. It is presented with 4 different kinds of smoothing methods. A smoothing method is used because it's possible that the baseline generates words that are not in the article. Instead of giving these words a score equal 0, smoothing allow us to give a much smoother scoring. Resulting a better picture of how good the generated headlines are. *Smoothing 0* is the actual BLEU score without any smoothing. *Smoothing 1* adds a epsilon value (0.1) to the score. *Smoothing 2* uses the longest common subsequence and skip-bigram statistics. *Smoothing 3* is computed by giving a score of $\frac{1}{(2^k)}$, instead of 0, for each precision score whose matching n-gram count is null. k is 1 for the first 'n' value for which the n-gram match count is null, and increases by 1 for each n-gram that is null. The last method is *Smoothing 4*. Instead of scaling to $\frac{1}{(2^k)}$, it divides by $\frac{1}{ln(len(T))}$, where T is the length of the translation. The *Smoothing 0* method have naturally a larger value then the other methods, because it simply ignores cases with null instances of n-grams. Thus, it gets a shorter length and a larger average BLEU score.

	Smoothing 0	Smoothing 1	Smoothing 2	Smoothing 3	Smoothing 4
BLEU score	0,5637	0,1019	0,2653	0,1572	0,2433

Table 5.2: BLEU scores for the baseline.

Table 5.3 shows the accuracy score for each category and the overall accuracy. All these accuracy scores have been calculated based on the generated headlines from the evaluation set.

	Sport	Økonomi og næringsliv	Politikk	Kriminalitet og rettsvesen	Ulykker og naturkatastrofer	Total
Accuracy	0.9365	0.8520	0.8620	0.9065	0.9580	0.6651

Table 5.3: Accuracy score for baseline

Figure 5.2 illustrates the attention weights for two examples from the test corpus. The x-axis is labeled with the words in each article example, and the y-axis is labeled with the generated words. Each row shows which words the decoder pays attention to when it generates the labeled word seen on the y-axis for the same row. This is illustrated with either black or white squares, or values between 0 and 1. Black means that the corresponding word on the x-axis is not important and white squares are important for the generation[10].

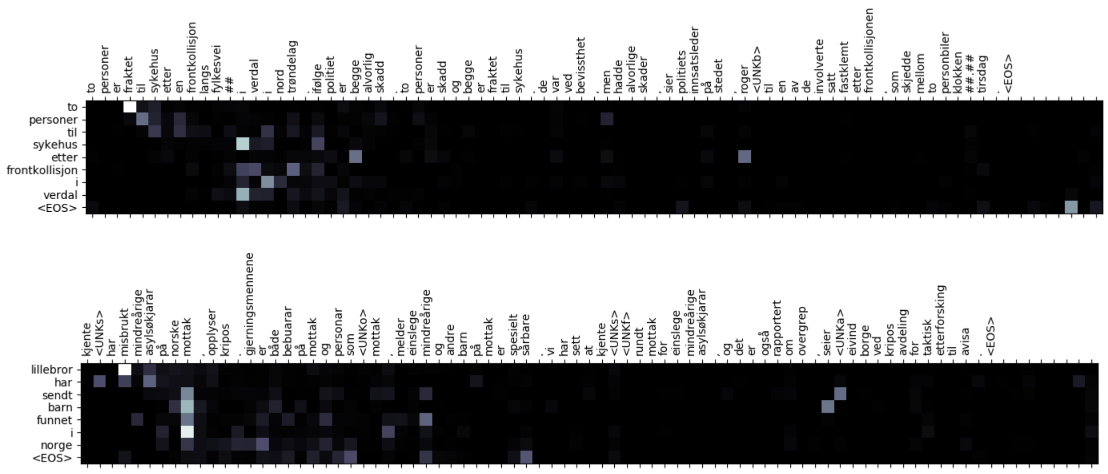


Figure 5.2: The first figure illustrates a good generated headline, with seemingly irrelevant attention weights. The second figure illustrates bad generation, as well as seemingly irrelevant attention weights.

5.5.1 Qualitative baseline results

Below we will present tables with qualitative results for the baseline. We will show both some good and bad examples. Each example is illustrated in a table where first row is the input article, the second row is the ground truth for that article and the final row is five generated headlines from our baseline based on our beam search method. The score in these examples (the number in front of the generated headlines) are the average softmax outputs, as described in Section 4.1, and can be viewed as a measure of how good the model itself believes the headline is. Table 5.4 and 5.5 are two good examples and Table 5.6 and 5.7 are two bad examples.

hovedindeksen på oslo børs gikk opp #,# prosent til ###,## poeng torsdag . <UNKo> bidro til oppgangen . statoil var som vanlig mest omsatt , og kursen gikk opp #,# prosent . vinnerne var pgs , seadrill og subsea #, som gikk opp henholdsvis ##,#, ##,# og #,# prosent . marine harvest var blant taperne , og kursen falt # prosent .
god dag på oslo børs
-0.1270915985107422 oppgang på oslo børs -0.5674104690551758 positiv start på oslo børs -0.6263462702433268 oppgang på oslo børs opp -0.6570791516985212 oppgang på oslo børs i pluss -0.7769292195638021 positiv utvikling på oslo børs

Table 5.4: A good baseline example.

[10]The scale from black to white varies. Black is always 0, while white is a relative number between 0 and 1, depending on the output distribution, where white is the highest output value.

en ## år gammel mann er sendt til sykehus med uavklarte skader etter å ha blitt knivstukket i arendal . politiet har pågrepet en ## år gammel mann i saken . politiet fikk beskjed om hendelsen klokka #.## natt til mandag , etter at vitner hadde ringt etter ambulanse . den skadde ## åringen ble sendt til sykehus med uavklarte skader . foreløpig vet vi ikke mye om omstendighetene .
mann knivstukket i arendal
-0.19716720581054686 mann knivstukket i arendal -0.22423013051350912 ## åring knivstukket i arendal -0.6355757713317871 gutt ## knivstukket i arendal -0.6499004364013672 ## åring knivstukket knivstukket i arendal -0.7058693567911783 ## åring knivstukket i stjørdal

Table 5.5: Another good baseline example.

to kvinner fra kristiansand er dømt for å ha satt fyr på en hytte i <UNKå> i vest agder sommeren ####. ifølge tiltalen bar de to kvinnene ut en tv og noen <UNKj> , før den ene av dem helte <UNKt> på hytta og tente på .
to kvinner dømt for å ha satt fyr på hytte
-0.8095567491319444 to helikoptre på jakt på et halvt år -0.8121957778930664 to helikoptre på jakt på <UNKg> i vest -0.8993104696273804 to helikoptre stuper på <UNKg> i vest -0.9026612175835503 to helikoptre på jakt etter <UNKg> i vest -0.9071855545043945 to års fengsel for vest i vest agder

Table 5.6: A bad baseline example.

britiske myndigheter vil forby offentlige institusjoner å boikotte varer produsert av <UNKu> selskap eller <UNKo> som israel . i forslaget fra regjeringen , som ventes å bli offentliggjort når <UNKc> <UNKo> minister matt hancock besøker israel senere i uka , trues universiteter og andre offentlige institusjoner og myndigheter med streng straff dersom de vedtar lokale boikotter . lokale <UNKb> kan . . . hindre britisk eksport og skade internasjonale forbindelser , heter det i en kunngjøring fra <UNKh> kontor .
britene forbyr boikott av israelske varer
-2.181590270996094 tilsyn av palestina iran -2.28331724802653 tilsyn med iran om flyktninger -2.3549728393554688 tilsyn med iran om israel reagerer på, -2.382332611083984 tilsyn med israels utestengelse -2.403347969055176 tilsyn av iran stjerne

Table 5.7: Another bad baseline example.

5.6 Classifier

Before showing result for our proposed models, we will present some result for our classifier. As described in section 4.2 we propose a new model that include some additional loss, and to find this loss we need a classifier. The classifier used in our experiments is trained on the same training and evaluation sets as mentioned in section 5.1. Hyperparameters used for training our classifier is presented in Table 5.8. These parameters is motivated by [Kim, 2014], with small adjustments. The learning rate is the standard value for the Adam optimizer, which also is different from what [Kim, 2014] uses, who use the Adadelta optimizer.

Hyperparameter	Value
Learning rate	0.001
Hidden size	64
Dropout	0.7
Kernels	100
Kernel size	[2, 3, 4]
Batch size	16

Table 5.8: Hyperparameters used to train our classifier.

The classification loss for the training set and the evaluation set is shown in Figure 5.3. These are averages per epoch. Precision, recall, F1 and accuracy for the actual headlines from the evaluation set is shown in Table 5.9.

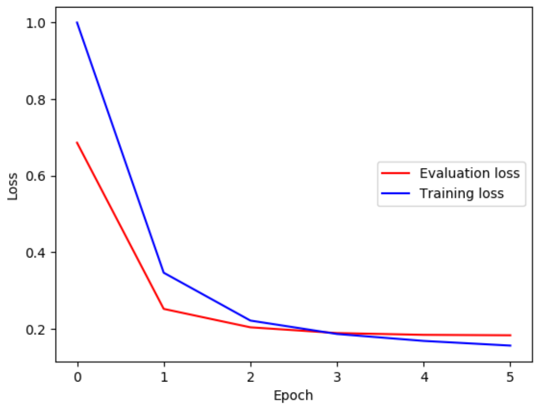


Figure 5.3: Training and evaluation loss for the classifier.

	Sport	Økonomi og næringsliv	Politikk	Kriminalitet og rettsvesen	Ulykker og naturkatastrofer	Total
Precision	0.951	0.796	0.896	0.907	0.886	-
Recall	0.95	0.708	0.746	0.695	0.765	-
F1	0.95	0.749	0.814	0.787	0.821	-
Accuracy	0.965	0.893	0.895	0.928	0.972	0.74

Table 5.9: Classifier measurements

5.7 Proposed model results

In this section, we will go through the results obtained by our proposed model presented in Section 4.2. We first present results where we make a simplification to the proposed model, only concatenating the context vector with the articles categories. Then, we present results from the full model, where the additional loss is included as well. Training these models have also been done on the infrastructure mentioned in section 5.2 with the parameters from Table 5.1. We will also here present Loss, BLEU score, attention weights, accuracy and qualitative results, the same way as we did for the baseline.

5.7.1 Including the categories

The first proposed model (the simplified version of the proposed model) is simply just concatenating the context vector with the articles categories. Figures and Tables below will present our findings for this simple adjustment to the model.

Figure 5.4 illustrated the loss development during training time, and Table 5.10 show the corresponding BLEU scores for this proposed model. The BLEU scores have been calculated with respect to the generated headlines from articles in the evaluation set.

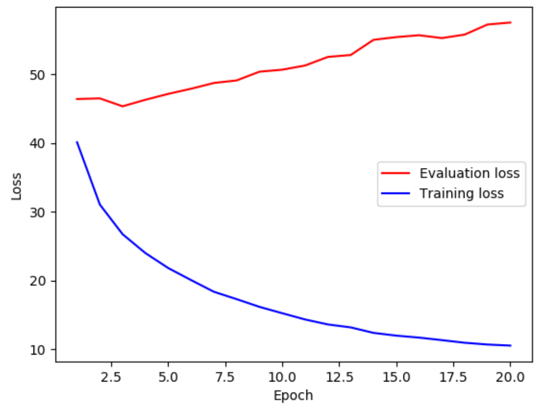


Figure 5.4: Both training and evaluation loss when categories are included

	Smoothing 0	Smoothing 1	Smoothing 2	Smoothing 3	Smoothing 4
BLEU score	0.5640	0.1044	0.2693	0.1609	0.2457

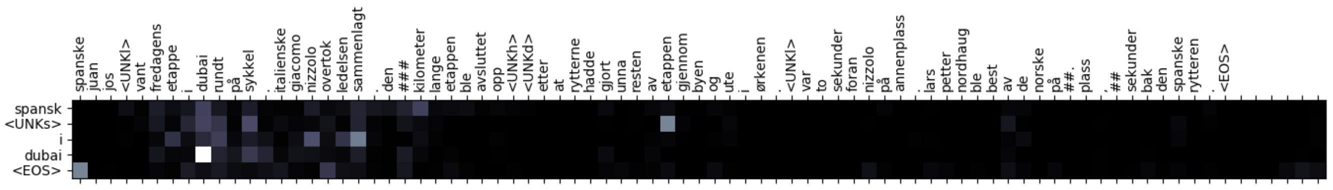
Table 5.10: BLEU scores when categories are included

Table 5.11 shows the accuracy score for each category and the total accuracy for the proposed model with categories concatenated with the context vector. All these accuracy scores have been calculated based on the generated headlines from the evaluation set.

	Sport	Økonomi og næringsliv	Politikk	Kriminalitet og rettsvesen	Ulykker og naturkatastrofer	Total
Accuracy	0.9460	0.8542	0.8623	0.9023	0.9562	0.6700

Table 5.11: Accuracy score when categories are included

Figure 5.5 illustrates the differences in attention between the baseline and the proposed model with only concatenating the categories.



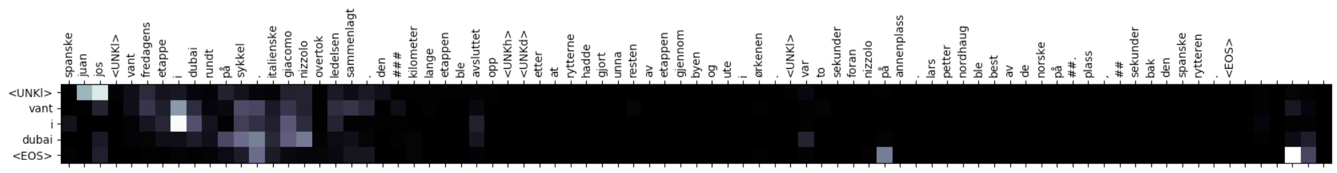


Figure 5.5: First image is the attention weights for the baseline, and the second image is attention weights when the sport category is included

Attention experiments

In this section we will illustrate a few different experiments to see how the attention responds and changes it focus. Figure 5.6 and 5.5 illustrate the changes in attention weights by only changing the category injected to the context vector. This effect is also shown in Figure 5.7. Figure 5.8 shows the same effect, but here a hidden size of 64 is used. The last attention illustration shown in Figure 5.9 is a constructed example where we have selected and merged together two different sentences from the corpus. The reason behind this constructed example is to see how the model view the input and where the attention is focused.

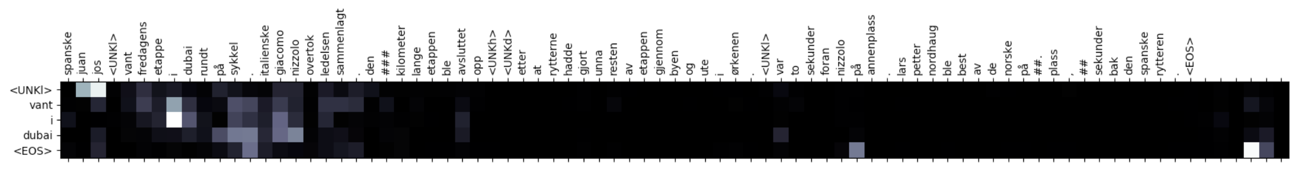


Figure 5.6: Attention weights when the category is changed to politics

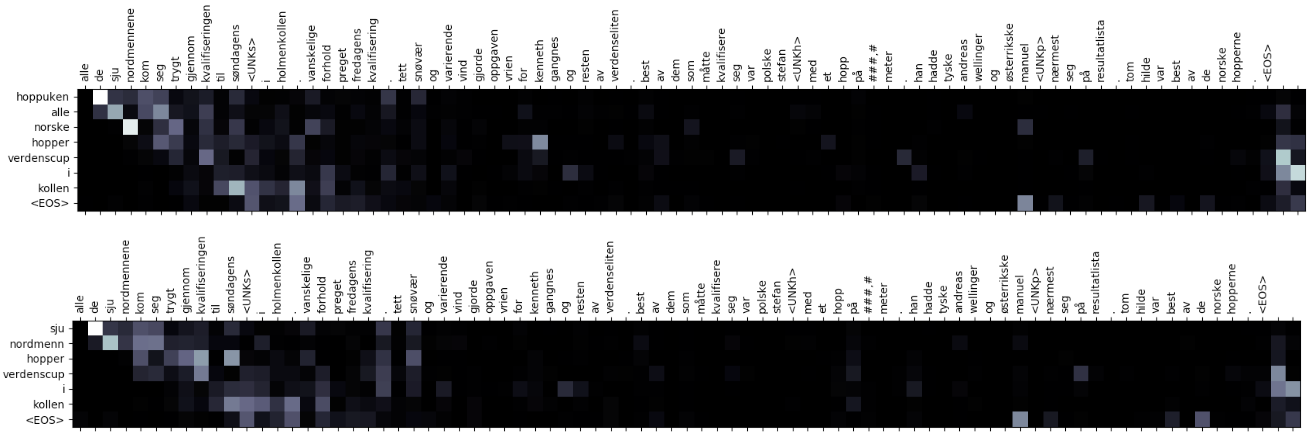


Figure 5.7: Attention weights with difference at hidden size 128. First image use the sports category. Second image uses the politics category.

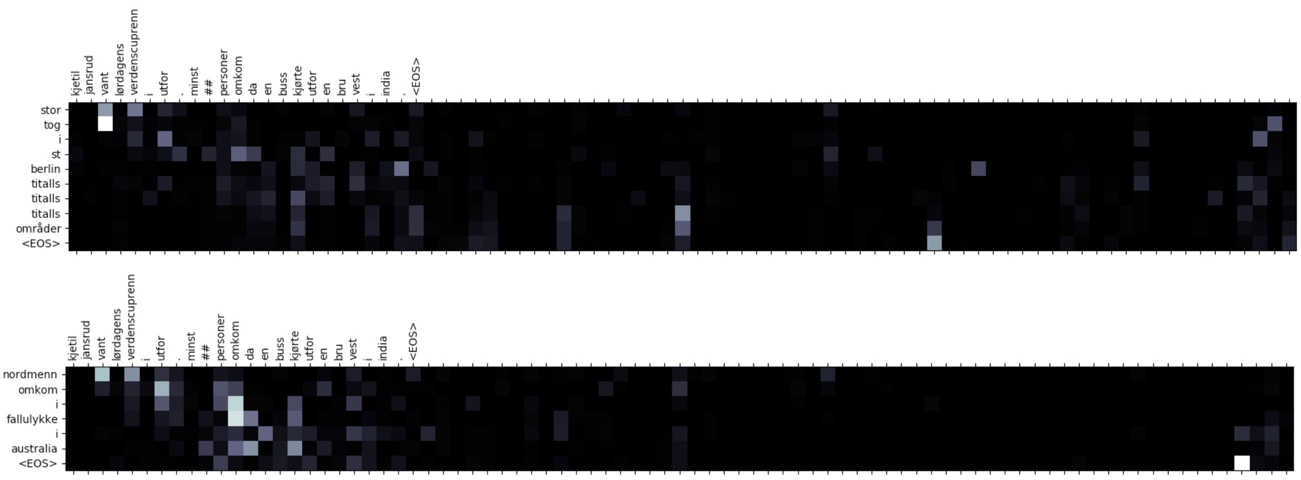


Figure 5.8: Attention weights for hidden size 64. First image use the sports category. Second image use the politics category

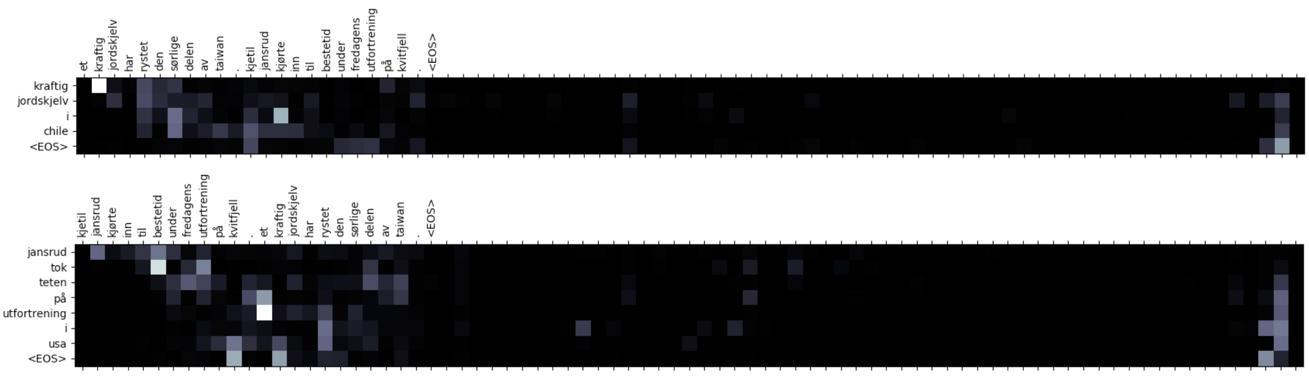


Figure 5.9: Attention weights for a constructed article. Second image has the sentences flipped.

Qualitative results for the proposed model

Below we will present some qualitative results for our proposed model. The tables presented in this section is built up in the same way as described in section 5.5.1. Table 5.12 and 5.13 shows qualitative results for the same input article, but with different categories. Table 5.14 and 5.15 shows qualitative results for the constructed example and the effect of flipping the input sentences. In the last two tables, there is only two rows because they are constructed examples without a actual headline.

alle de sju nordmennene kom seg trygt gjennom kvalifiseringen til søndagens <UNKs> i holmenkollen . vanskelige forhold preget fredagens kvalifisering . tett snøvær og varierende vind gjorde oppgaven vrien for kenneth gangnes og resten av verdenseliten . best av dem som måtte kvalifisere seg var polske stefan <UNKh>med et hopp på ###,# meter . han hadde tyske andreas wellinger og østerrikske manuel <UNKp>nærmest seg på resultatlista . tom hilde var best av de norske hopperne .
samtlige nordmenn klare for kollen renn
-0.7962527275085449 hoppuken alle norske hopper verdenscup i kollen -0.8302636827741351 sju nordmenn hopper verdenscup i kollen -0.8912453651428223 hoppuken alle alle norske hopper i kollen -0.8985360009329659 hoppuken alle norske hopper i kollen -0.9145355224609375 alle verste hopper verdenscup i kollen

Table 5.12: Qualitative results when the category is sport

alle de sju nordmennene kom seg trygt gjennom kvalifiseringen til søndagens <UNKs> i holmenkollen . vanskelige forhold preget fredagens kvalifisering . tett snøvær og varierende vind gjorde oppgaven vrien for kenneth gangnes og resten av verdenseliten . best av dem som måtte kvalifisere seg var polske stefan <UNKh>med et hopp på ###,# meter . han hadde tyske andreas wellinger og østerrikske manuel <UNKp>nærmest seg på resultatlista . tom hilde var best av de norske hopperne .
samtlige nordmenn klare for kollen renn
-0.7649854932512555 sju nordmenn hopper verdenscup i kollen
-0.8500755627950033 sju hopper verdenscup i kollen
-0.9290696552821568 alle verste hopper verdenscup i kollen
-0.9397819836934408 alle verste hopper i kollen
-0.9545952479044596 sju nordmenn hopper i kollen

Table 5.13: Qualitative results when the category is politics

et kraftig jordskjelv har rystet den sørlige delen av taiwan . kjetil jansrud kjørte inn til bestetid under fredagens utfortrening på kvitfjell .
-0.5483936309814453 kraftig jordskjelv i chile
-0.652642822265625 kraftig jordskjelv utenfor indonesia
-0.6975198745727539 kraftig jordskjelv i indonesia
-0.7229232788085938 kraftig jordskjelv utenfor mexico
-0.754338128226144 kraftig jordskjelv i chile etter ferien

Table 5.14: Beam search results for a constructed article

kjetil jansrud kjørte inn til bestetid under fredagens utfortrening på kvitfjell . et kraftig jordskjelv har rystet den sørlige delen av taiwan .
-0.9284952878952026 jansrud tok teten på utfortrening i usa
-0.9417125384012858 jansrud ligger til konkurrentene igjen
-0.9762239456176758 jansrud begynner å ta opp på utfortrening
-0.978644847869873 jansrud ligger til konkurrentene , jansrud minst to poeng
-0.9971389770507812 jansrud tok teten , men varm på utfortrening

Table 5.15: Beam search results for a constructed article where the input is flipped

5.7.2 Additional loss

This is our second variation to the proposed model where we include the additional loss gained from the classifier. Below we will present the loss (Figure 5.10), BLEU score (Table 5.16) and accuracy (Table 5.17) to compare the results with the baseline and the proposed model where the category is used as input only.

This model is the complete model explained in Section 4.2, where we use the categories associated with the articles as additional input, as well as using the classifier explained in Section 5.6 to provide an additional loss used to scale the original loss gained from our original model. The scaling factor is calculated as an average classification loss on the generated sequences for an entire minibatch. This factor is multiplied directly with the original loss.

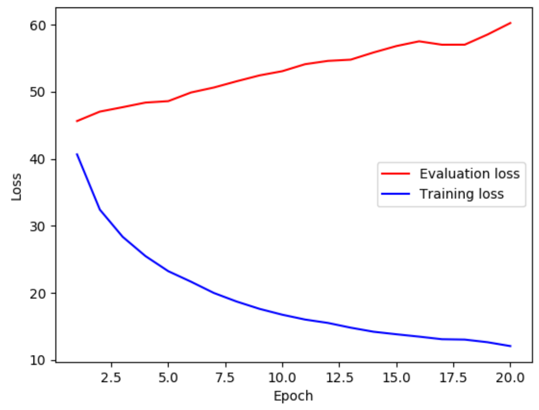


Figure 5.10: Evaluation and training loss when additional loss is included in the model.

	Smoothing 0	Smoothing 1	Smoothing 2	Smoothing 3	Smoothing 4
BLEU score	0.5669	0.0968	0.2589	0.1508	0.2409

Table 5.16: BLEU scores when additional loss is included.

	Sport	Økonomi og næringsliv	Politikk	Kriminalitet og rettsvesen	Ulykker og naturkatastrofer	Total
Accuracy	0.9388	0.8571	0.8586	0.8980	0.9537	0.6652

Table 5.17: Accuracy score when additional loss is included

Figure 5.11 shows a smoothed[11] graph of the average classification loss per batch. Our assumption is that, if the classification loss is very stable, the effect of the additional loss will not be too great, considering that it will only scale the gradients, which is similar to scaling the learning rate.

Table 5.18 shows the absolute sums of the expected average gradients stored in the Adam optimizer for 6 random layers (groups) in the model, 3 from the encoder and 3 from the decoder, after 2 epochs. The first column indicates a scale multiplier, where 0 means that we do not include the scale factor, 1 is the normal scaling and 100 is a additional scale factor (multiply the classification loss by 100) to see if the trend persists at higher scales. The experiments were conducted by following these steps:

- Saved the model at initialization
- Removed shuffling of training samples per epoch
- Set dropout to 0.0
- Set teacher forcing to 1.0

[11]The graph in Figure 5.11 is smoothed with a gaussian filter with sigma=2.0 to remove special case outliers

These steps were done to remove randomization when looking at the gradients. We ran the experiments multiple times to make sure that the same parameters generated the exact same gradients each time.

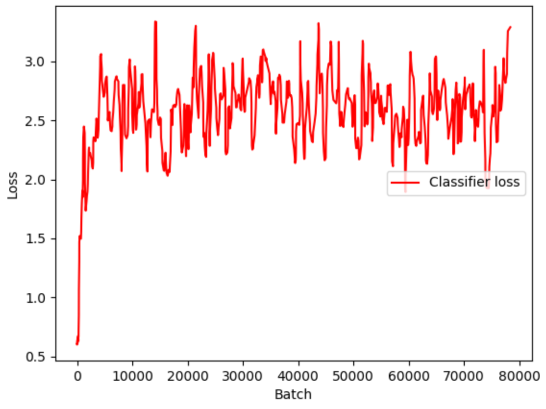


Figure 5.11: Classifier loss for each batch.

	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6
0	64.83	33.39	14.21	86.81	34.28	0.10
1	169.47	109.50	43.03	212.86	98.23	0.27
100	14797.30	14637.30	4850.05	18255.30	11837.00	34.37

Table 5.18: Absolute sums of expected average gradient per parameter group in the Adam optimizer. The first column indicates the scale multiplier.

Table 5.19 shows an example where the loss is calculated for two different sets of input categories for the same article. The table includes the article, the generated headline, the scores predicted by the classifier, and the two losses calculated for the category sets (Sport, Økonomi og næringsliv) and (Sport). The label is classified as positive if the prediction score is positive, and vice-versa. This example shows that the classification loss can be very sensitive to the input categories.

Article	tv # jobber hardt for å sikre seg rettighetene til det norske <UNKf>em og vm kvalifiseringskamper for #### og ####. det er discovery som i øyeblikket har rettighetene til <UNKl>. <UNKr>det amerikansk eide konsernet sitter på avsluttes imidlertid med kvalifiseringen til vm sluttspillet i #### i russland . nå ønsker tv # å løfte <UNKl>over på sine plattformer .
Generated headline	tv # i for em em
Prediction scores	35.37 -23.66 -17.36 -21.04 -25.14
Category(1, 1, 0, 0, 0)	loss = 4.73
Category(1, 0, 0, 0, 0)	loss = 0.00

Table 5.19: The loss from binary cross entropy on the predictions generated from the classifier for two different sets of input categories on the same article and generated headline.

Chapter 6

Discussion and Conclusion

In this chapter we will evaluate and discuss our results, findings and other observations. We will summarize and conclude our project, and finally, in the future works section, we will discuss possible improvements and extensions to our research.

6.1 Discussion

In this section we will evaluate and discuss general trends, key findings and other observations. We will compare the baseline model with the two versions of our proposed model, as well as comparing our work to the current state of the art. Finally, we will answer the research questions stated in Section 1.2.

6.1.1 General trends

Figure 5.1 and 5.4 illustrate a very similar loss development during training. This is as expected because the category concatenation alone will not be a big enough force to drive it to new discoveries. We can also see in both figures that evaluation loss quickly starts to increase. One might think early stopping would be beneficial, but since we only measure against one ground truth we believe that it is better to overfit the training data. This seems to be a good hypothesis, since our model creates decent headlines that are grammatically and syntactically correct even though it seems to not generalize well.

Table 5.2 and 5.10 shows the BLEU scores for the baseline and the simplified proposed model. As we can see, there is not much difference between the models. One can argue that the proposed model seems to have a slight increase in the score. This increase is only a sign of the right direction of our ideas, but it is clear that alone it does not achieve our goal. Comparing results to the state of the art have proven to be very difficult. This is because most of them only report one score, and do not mention if they have used any smoothing or any other information. That is why we have only chosen to compare them to each other too see the relative changes. Another reason why BLEU is not an optimal way to measure the quality of a headline is the fact that a very limited amount of references is used to do the computation, while a particular article might have a lot higher number of possible good headlines. Many state of the art have also presented a different machine translation score called *ROUGH*. This method seemed much easier to compare with our result, given that we had used the same datasets, but since we could not find any legit version of ROUGH we had to drop this comparison[1].

We can see in Figure 5.2 that the generated headlines and the attention weights does not seem to have much in common in our baseline model. This might be because we overfit the training data too much and it has seen some similar inputs before, or it can just be that the model have focus on different parts of the input than a human would have. One can clearly see that the headline in the first image is pretty decent for the input, so this might be the case. Our simplified proposed model have cases where we can see higher correlation between the generated headlines and the attention weights. This can be seen in Figure 5.7. Here, some of the generated words have attention on similar words from the input. Thus, we believe that our proposed model do generalize better than the baseline model, to some extent.

Both our proposed models and our baseline produces some good headlines and some bad ones. Some good examples are illustrated in Table 5.4 and 5.5 for the baseline. Here, we can see that both models believes they produce good headlines with scores like "-0.1", and that the headlines captures both the theme and the main points in the article. On the other hand, in Table 5.6 and 5.7 for the baseline we can see some bad examples. Here, we see that the headlines are either composed of seemingly random words and do not make any sense, or they can be proper sentences, but totally wrong for the input article. We can also see that some times the models thinks it produces some decent sentence and have actually a score close to zero. This suggest that the model struggles to capture the subject of the input.

Table 5.3 and Table 5.11 shows the accuracy for the baseline and the simplified proposed model when input category is included, respectively. In this case we can note that the accuracy is quite high for every single category (>0.85), and the total accuracy is also somewhat high for both (>0.66). Even if a total accuracy of 0.66 is not a perfect score (which would be 1.00), it is still relatively close to the ground truth[2] as shown in Table 5.9, which is 0.74. This is very promising, and functions as a great addition to the other quantitative measures used, especially since both loss and BLEU score is very biased towards the actual headline, while there could exist a lot of different "good" headlines for one particular article.

Our results show the tendency, but some of the results that seem to be different between models could be because of randomization in the model (initial parameters, corpus shuffling between epochs, dropout and teacher forcing). Two different runs with the same parameters can produce very different top5 results in general because of small variations. Still, the main trends are the same for the same parameters, I.e the loss, BLEU score, accuracy and the quality of the generated headlines are very similar.

6.1.2 Key findings

Looking at Figure 5.9, we can note that the attention weights and the generated sentence mainly focuses on the first part of the input text. This is also observable in all the other attention figures presented. The majority of the focus always lies in the first sentence or the first couple of sentences.

[1]It would still be very hard to compare results. E.g. we would have to use the same dataset, and the score from this measurement would still only be computed based on one (or just a few) reference headline(s), which is the main issue with BLEU as well.

[2]In this case, ground truth refers the score produced from our assumed perfect classifier on the actual headlines.

As mentioned in Section 5.1, most articles summarize the essence of the text in the first couple of sentences, meaning that our model might learn that the first part of the input text is most important. This is perhaps as expected, but it also suggests that the input articles are not pruned enough. In this regard, it would be interesting to see if the model could optimize better if we removed some redundant parameters by cutting the input articles at e.g. 40 words instead of 80.

One of our main findings when investigating our models was that the beam search itself is a very important aspect. In almost all cases, the top5 from our beamsearch will not even include the näive top1 result, but rather include much better qualitative results. This also begs the question, could we further improve this mechanism? This is further discussed in Section 6.3.

Figure 5.7, as well as Table 5.12 and Table 5.13 shows an example using the same input article with two different input categories when the hidden size is 128. We can see from Figure 5.7 that the attention weights are changed slightly, but not that much. The trend is pretty much the same. We can note that the produced sentence in this example is slightly different, as also seen in Table 5.12 and Table 5.13. Looking at a similar experiment in Figure 5.8, where the hidden size is 64, we can see that the difference in attention weights is higher, and the produced sentence is actually very different. These results tells us that the hidden size does seem to matter when it comes to the impact of the input category. However, as can be seen in this example, the quality is worse than that of hidden size 128. Figure 5.5 and Figure 5.6 shows a typical example of hidden size 128 comparing two different input categories, as well as the baseline results for the same article. In this case we can clearly see that the attention weights are very similar when using different categories, but still very different from the baseline. This seems to be the general trend. What we can note from these results is that, generally, using the input category does seem to have an impact on the generation, and even a higher impact for a lower hidden size. However, changing the category does not seem to matter that much, and this might be because of the fact that the article reflects the category pretty well already, making it unlikely to produce large changes when changing the input category.

6.1.3 Single character tokens

To compensate for the huge vocabulary using one-hot vectors based on tokenized words, one thing we tried was to use a vocabulary with single character tokens, where we fed the network one character at a time and generated similarly one character at a time. This, however, turned out to be very slow.

Even if the vocabulary is under 40 characters (compared to, say, 30 000) it still doubles or triples the time to run X iterations, mostly because of the fact that the sequences (input and output) are now so large, e.g. 100 tokens (words) in a sequence would become about 1000 tokens (characters) which is fed into the network. This means that computing the forward values as well as computing the gradients takes a lot more time. It also adds a lot of overhead on the long term dependencies, which will make the LSTMs perform worse.

After seeing poor performance as well as poor results, we decided to scrap this idea pretty fast without testing or developing it further. After all, using standard one-hot vectors based on tokenized words provided good results, even if it does take some time to compute the softmax accross huge vocabularies.

6.1.4 Classifier

The classifier is not the main focus in this report, but it is an important part of our proposed model and future work. Hence, it is of importance to us that the performance is decent. Table 5.9 shows the performance of our trained classifier. Compared to [Kim, 2014], we see that our total accuracy comes very close to their performance. They present results on multiple datasets, but only two of them have a similar structure to ours. The first one, called *SST-1*, have 5 classes which are none-overlapping. On this dataset, [Kim, 2014] achieves an accuracy equal to 48.0. The second one, called *TREC*, have 6 classes, which also are non-overlapping. On this dataset, [Kim, 2014] achieves an accuracy equal to 93.6. None of these datasets can be compared exactly to ours, but it gives some insight that our model performs quite well (total accuracy of 0.74 on 5 overlapping classes).

6.1.5 Additional loss

As we can see from Table 5.16, the BLEU scores are on average a bit worse for this model than the model where we do not use the scaling additional loss. The same can be seen in Figure 5.10, where the loss is slightly higher for both the training corpus and the evaluation corpus. Accuracy, as seen in Table 5.17, is also on average a bit lower.

Figure 5.11 shows the classification loss for every batch that is run through the network. We can see that the classification loss is very stable. It starts at about 0.6, goes up to about 2.5 and is quite steady in the range between 2.2 to 2.9. The average classification loss is about 2.7. When using this additional loss, it will have the function as a scaling factor for the gradient (or, equivalently, the learning rate). This can be verified by looking at Table 5.18, where the expected average gradients are shown after 2 epochs using a multiplier of 0, 1 and 100 to the classification loss. We can clearly see that multiplying the 0 multiplier values with 2.7 gives us the results shown on the second line, and multiplying by 270 gives us the last line (approximately).

Considering that we found the learning rate to be optimal at 0.001 for our experiments, and that the additional loss used in this particular manner will only have the effect of scaling the learning rate, it will generally, as seen, only reduce the generalization.

One problem here, is the high value stability of the classification loss. One of the reasons for the stability is from having multiple classes and being multi-label at the same time, and combining this with a binary cross entropy loss function. Generally, even if our classifier does quite well on real data, it does not necessary do as well on noisy data, such as the produced sequences during training. An example is shown in row four in Table 5.19, where the loss is 4.7, while it actually gets 4 out of 5 labels correct. Here, we can see that the model predicts one of the two categories contained in the input correctly, but the second category prediction being totally wrong makes the loss very high. In row five, the second label is removed, and we can see that the loss in this case is 0.0. If a minibatch of length two contained these two articles, the average classification loss for this minibatch would still be 2.35, which is quite high considering that the model is actually getting 9 out of 10 correct label predictions.

We might have gotten very different results if we used attributes with only, say, 2 classes that are non-overlapping. If it is easier to discriminate between the classes, the model would most likely produce better feedback. In this case, the scaling factor introduced would not have been so stable (at a high value), and could be used to increase the gradients for the miss-classified cases more than for those that are correctly classified, possibly making the model generalize better. Using a different form of loss function should also be tested.

6.1.6 Preprocessing

One of the most important aspects that helped us achieve the results presented here has been preprocessing. As mentioned in Section 5.1, preprocessing of the dataset includes many techniques that we have fine tuned over many iterations, but is probably still far from perfect. Some of the problems with the dataset that is hard to get quite right includes:

- Removing redundant close-to-duplicates.
- Removing noisy data, e.g. articles with tables and seemingly random labels.
- Dialect and misspelling challenges, creating a lot of low-frequency words in our corpus.

Generally, there are a lot of different techniques that can be applied when preprocessing the data. For example, one could use different schemes for tokenizing low-frequency words (UNK tokens), numbers, punctations, etc. However, we are quite happy with how we managed to preprocess the dataset, which helped us achieve our results.

6.1.7 State of the art

As mentioned above, comparing our results to other state of the art results has proven challenging, and made even harder when there is two very different datasets used. However, comparing results between our different models has been quite helpful and provides a lot of insight in itself. We can also note that these results are, to the best of our knowledge, the first results in this research direction on a Norwegian dataset.

[Lopyrev, 2015], as mentioned in Section 3.1, also generates news headlines, and is the most similar work we have seen, compared to ours. The generalization is very visible in their work, seeing that the loss for both the training and the hold-out set starts at about 70, and ends at about 35 after 9 epochs. This is very different compared to our baseline results, where we start at about 47 training and evaluation loss and end at about 9 training loss and 58 evaluation loss after 20 epochs. The main advantage they have that makes generalization easier is that they train on the GIGAWORD corpus, using 5.5 million training examples, compared to our 125 thousand training examples. Their qualitative results seem to be more accurate, compared to ours, especially when predicting headlines from the same newspapers as their model was trained on. However, we still see a lot of the same trends as in our examples. Even if there are a few good examples produced, there are always some bad ones as well. One of the example predicted headlines they show in their paper is ”United set for test test”, which resembles the results we see in Table 5.6.

Comparing our results to the results in [Hu et al., 2017], we can note that our results are much less clear and visible. One of the main reasons is that in [Hu et al., 2017], they generate movie reviews, which can be viewed as generating random text based on some attributes (in their case: Tense and sentiment), while we generate headlines, which is a summarization task. Generally, a summarization task is much less free to generate random samples, making the attributes used to control the generation less important. There is also the question if using categories in our case is very optimal at all. As mentioned above, the categories are already represented by the content, making it less probable to steer away from the main topic when the category is changed. We further discuss the possibilities to use other attributes in Section 6.3. Compared to the results in [Li et al., 2016], the same arguments applies. In their case, they use twitter conversations, which similarly to movie reviews, is more free in terms of possible generated samples.

6.1.8 Goals and research questions

Regarding the goal and research questions in Section 1.2, we conclude that we managed to *generate meaningful headlines based on the content of an article* (Examples include Table 5.4 and Table 5.5).

In response to research question 1, our conclusion is a yes, the generation is controllable through the use of an input category, to some extent. We understand that our approach might not be the best way to do it, and that the number of parameters (hidden nodes) plays a huge role for the possible influence. What we can note is that the effect of using a particular category as input to our proposed model, is that it will change the preference for some words (I.e. the score given by the final softmax output layer), but by small margins. The small margins means that in general, the difference will not be that high, as already mentioned.

Regarding research question 2, we can not fully conclude anything, and emphasize that further research is required. However, we can note that, in the case that we tried, using linear scaling of the loss, as described in Section 5.7.2, the generation (final results) are worse than for the baseline and the simplified model. The convergence trend during training is similar to the one of the simplified model and the baseline, but is slightly slower, ending at about 11.9 training loss and 60.2 evaluation loss after 20 epochs, compared to 9.6 training loss and 57.5 evaluation loss for the simplified model, and 9.5 training loss and 57.8 evaluation loss for the baseline.

In terms of how well we managed to fulfill the goal of *Controlling the generation of headlines based on the content of an article and a given category*, we conclude that we do quite well on the first part, I.e. general headline generation based on the content, but that more work is required when it comes to controlling the generation. There are definitely still many things to try, and some of these are discussed in Section 6.3.

6.2 Conclusion

In this project, we have explored the seq2seq model with attention and how this can be used to generate headlines for articles. This model serves as our baseline, and we propose two extensions to this basic model that further aims to control the generation. In one of the models, we do this by injecting categories into the context vector, and in the other model, we extend this idea by scaling the original loss with a classification score from the generated sequence.

The results are generally promising, showing how such models can generate qualitatively good headlines, even with limited amounts of data. When injecting the category into the context vector, the generalization seemed to increase, compared to the baseline. However, the model using the classification score from the generated sequence seemed to perform worse in comparison to the other models. We also looked into what goes on in the attention mechanism, noting some interesting findings that should be further explored in future studies.

The problem at hand is both complex and challenging, and even thought we are quite happy with our results, we realize there are a lot of things left to explore.

6.3 Future work

This section will give some insight into how our model can be used as a framework for future work. This will consist of some new approaches, ideas and possible solutions to the current problems.

Dataset and preprocessing

The dataset used in our experiments contains a relative small number of examples ($\sim$132 000). Compared to similar research, the *Gigaword* corpus[3] which is the most used in the state of the art (E.g. [Lopyrev, 2015] and [Nallapati et al., 2016b]) contains some million examples. The *CNN* dataset[4] used in [Nallapati et al., 2016b], [Hermann et al., 2015] and [Chen et al.,] contains about 290 000 examples. Including more data when training deep learning models would increase the generalization in most cases. The *Gigaword* corpus costs 3000$, but both the *CNN* dataset, and *The Signal Media One-Million News Articles Dataset*[5] are open to anyone, and would be useful to explore, both to see if the generalization gets better and also to be able to compare results quantitatively with the current state of the art which have used the same dataset.

As mentioned in Section 6.1.2, cutting the articles earlier (<80 words, e.g. 40) would be interesting to see how the attention weights would change. I.e. would we see the attention weights spread more evenly across the entire input instead of just at the beginning? It would also be interesting to see if the generalization would improve.

Input attributes

A straight forward extension to our work is to try different attributes (e.g. sentiment). One could also inject multiple attributes concatenated together to the context vector (e.g ”football” and ”negative”). So far, we have only tried categories, and it seems like the model actually captures this feature quite well already, providing a very limited gain. It would be interesting to see how different attributes would affect the generation. Both [Li et al., 2016] and [Hu et al., 2017] shows some promising results when including input attributes to their models. We believe that using some attributes that is not that easy for the model to capture by the content could give some richness to the headline generation, and the generation might possibly be more easily controllable.

[3]Gigaword corpus: `https://catalog.ldc.upenn.edu/ldc2003t05`

[4]CNN dataset: `https://cs.nyu.edu/~kcho/DMQA/`

[5]The Signal Media One-Million News Articles Dataset: `http://research.signalmedia.co/newsir16/signal-dataset.html`

Beam search

The beam search mechanism presented in Section 4.1 proved to be of high importance for our results. Experimenting further with this mechanism would be highly relevant.

Optimally tuning the parameters mentioned in Section 4.1 for this mechanism would be relevant to see if it is possible to gain even better results from it. However, this task is not as straight forward as it may look. One problem when tuning these parameters is that we keep beams based on the score, and even bad sequences can get high scores in some cases. E.g. if we choose one top20 word, that initially has a low score, the words following that word can still get quite high scores, and the average becomes high. This can happen even if the sequence is not too great. The parameters generally needs to be adjusted to not include too many ”false positive” (irrelevant sequences that the model believes are relevant) sequences during the search.

Incorporating the classification of the sequence in the beam search would be relevant to further steer the generation in a certain direction. I.e. classifying the current sequence in the beam search and use a score scheme where hitting the desired category affects the average score positively and not hitting the desired category would affect the average score negatively. Naturally, this would increase the accuracy of our results, as well as having a possibility to increase the difference in generation when the category is changed for the same article.

The beam search mechanism could also be used to restrict the length of the generated sequence. I.e. only keeping beams in a certain length range. The results from such an experiment would be interesting in themselves, and also be useful in a real-world scenario.

Reinforcement learning

Reinforcement learning have had much success lately, because of the simple idea behind it. Let the model interact with the environment and learn by using rewards. This approach can lead to new and better understanding of the environment. [Hu et al., 2017] uses a semi-supervised framework in their work, showing one example of how such a framework can be formulated. It would definately be interesting to incorporate some reinforcement learning ideas into our model, and especially in the proposed model, where we could potentially use reinforcement learning to do something better than simple linear scaling. One example is to use rewards from the classification, e.g a +1 reward for a correct classification, and a -1 for a missclassification. There is a lot of current research in reinforcement learning, and many approaches are feasible. This is indeed a interesting direction to go.

Bibliography

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473. Retrieved October 16, 2017.

[Chauvin and Rumelhart, 1995] Chauvin, Y. and Rumelhart, D. E. (1995). *Backpropagation: theory, architectures, and applications*. Retrieved October 21, 2017.

[Chen et al.,] Chen, V., Montaño, E. T., and Puzon, L. An examination of the cnn/dailymail neural summarization task. Retrieved November 22, 2017.

[Cho et al., 2014a] Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014a). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*. Retrieved November 17, 2017.

[Cho et al., 2014b] Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078. Retrieved October 15, 2017.

[Gers et al., 1999] Gers, F. A., Schmidhuber, J., and Cummins, F. (1999). Learning to forget: Continual prediction with lstm. Retrieved October 25, 2017.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep learning*. MIT press. Retrieved October 21, 2017.

[Graves et al., 2013] Graves, A., Jaitly, N., and Mohamed, A.-r. (2013). Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE. Retrieved November 13, 2017.

[Graves and Schmidhuber, 2005] Graves, A. and Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610. Retrieved November 13, 2017.

[Hermann et al., 2015] Hermann, K. M., Kocisky, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems*, pages 1693–1701. Retrieved November 22, 2017.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780. Retrieved October 27, 2017.

[Hu et al., 2017] Hu, Z., Yang, Z., Liang, X., Salakhutdinov, R., and Xing, E. P. (2017). Toward controlled generation of text. In *International Conference on Machine Learning*, pages 1587–1596. Retrieved November 14, 2017.

[Kim, 2014] Kim, Y. (2014). Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882. Retrieved October 16, 2017.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. Retrieved December 12, 2017.

[LeCun et al., 1998] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324. Retrieved November 3, 2017.

[Li et al., 2016] Li, J., Galley, M., Brockett, C., Spithourakis, G. P., Gao, J., and Dolan, B. (2016). A persona-based neural conversation model. *arXiv preprint arXiv:1603.06155*. Retrieved November 14, 2017.

[Lopyrev, 2015] Lopyrev, K. (2015). Generating news headlines with recurrent neural networks. *CoRR*, abs/1512.01712. Retrieved October 16, 2017.

[Martens and Sutskever, 2011] Martens, J. and Sutskever, I. (2011). Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040. Retrieved October 23, 2017.

[Mitchell, 1997] Mitchell, T. M. (1997). Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37). Retrieved November 15, 2017.

[Moratanch and Chitrakala, 2017] Moratanch, N. and Chitrakala, S. (2017). A survey on extractive text summarization. In *Computer, Communication and Signal Processing (ICCCSP), 2017 International Conference on*, pages 1–6. IEEE. Retrieved November 14, 2017.

[Nallapati et al., 2016a] Nallapati, R., Xiang, B., and Zhou, B. (2016a). Sequence-to-sequence rnns for text summarization. *CoRR*, abs/1602.06023. Retrieved October 14, 2017.

[Nallapati et al., 2016b] Nallapati, R., Zhou, B., Gulcehre, C., Xiang, B., et al. (2016b). Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*. Retrieved December 10, 2017.

[Negnevitsky, 2005] Negnevitsky, M. (2005). *Artificial intelligence: a guide to intelligent systems*. Pearson Education. Retrieved October 16, 2017.

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics. Retrieved November 20, 2017.

[Paulus et al., 2017] Paulus, R., Xiong, C., and Socher, R. (2017). A deep reinforced model for abstractive summarization. *CoRR*, abs/1705.04304. Retrieved October 14, 2017.

[Rojas, 2013] Rojas, R. (2013). *Neural networks: a systematic introduction*. Springer Science & Business Media. Retrieved October 21, 2017.

[Srivastava et al.,] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958. Retrieved November 15, 2017.

[Sutskever, 2013] Sutskever, I. (2013). Training recurrent neural networks. *University of Toronto, Toronto, Ont., Canada*. Retrieved October 23, 2017.

[Sutskever et al., 2014] Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215. Retrieved October 15, 2017.

[Yu et al., 2015] Yu, Z., Ramanarayanan, V., Suendermann-Oeft, D., Wang, X., Zechner, K., Chen, L., Tao, J., Ivanou, A., and Qian, Y. (2015). Using bidirectional lstm recurrent neural networks to learn high-level abstractions of sequential features for automated scoring of non-native spontaneous speech. In *Automatic Speech Recognition and Understanding (ASRU), 2015 IEEE Workshop on*, pages 338–345. IEEE. Retrieved November 13, 2017.

[Zhang and Zhou, 2006] Zhang, M.-L. and Zhou, Z.-H. (2006). Multilabel neural networks with applications to functional genomics and text categorization. *IEEE transactions on Knowledge and Data Engineering*, 18(10):1338–1351. Retrieved October 16, 2017.