CHALMERS

UNIVERSITY OF GOTHENBURG

Generating Headlines with Recurrent Neural Networks

Bachelor of Science Thesis in Computer Science and Engineering

ALEX EVERT, JACOB GENANDER, NICKLAS LALLO,
RICKARD LANTZ, FILIP NILSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2016

Bachelor of Science Thesis

Generating Headlines with Recurrent Neural Networks

ALEX EVERT
JACOB GENANDER
NICKLAS LALLO
RICKARD LANTZ
FILIP NILSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
University of Gothenburg

Göteborg, Sweden 2016

Generating Headlines with Recurrent Neural Networks

ALEX EVERT
JACOB GENANDER
NICKLAS LALLO
RICKARD LANTZ
FILIP NILSSON

Examiner: NIKLAS BROBERG

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Göteborg
Sweden
Telephone: +46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg 2016

Acknowledgments

First and foremost, we would like to thank our supervisor Mikael Kågebäck for giving us insightful feedback and input during the project. We would also like to thank him and the machine learning group (LAB) for letting us use their compute server. Finally, we would like to give acknowledgment to the people from the department for technical language, who gave us useful suggestions on the thesis.

Generating Headlines with Recurrent Neural Networks

ALEX EVERT
JACOB GENANDER
NICKLAS LALLO
RICKARD LANTZ
FILIP NILSSON
Department of Computer Science and Engineering,
Chalmers University of Technology
University of Gothenburg

Bachelor of Science Thesis

Abstract

This report describes the implementation and evaluation of two natural language models using the machine learning technique deep learning. More specifically, two different models describing recurrent artificial neural networks (RNNs) were implemented, capable of generating news article headlines. One model focused on the generation of random (unconditioned) headlines, and the other one on the generation of headlines based (conditioned) on a given news article. Both models were then trained and evaluated on a data set of approximately 500,000 pairs of news articles and their corresponding headlines. The task of summarizing large bodies of text into smaller ones, while maintaining the key points of the original text, has many applications. Quickly and automatically obtaining condensed versions of for example medical journals, scientific papers, and news articles can be of great value for the users of such content. The unconditioned model, implemented using a multi-layer RNN consisting of LSTM cells, was able to produce headlines of moderate plausibility, a majority being syntactically correct. The conditioned model was implemented using two RNNs consisting of GRU cells in an encoder-decoder-network with an attention mechanism, allowing the network to learn what words to focus on during headlining. Although the model managed to identify important keywords in the articles, it seldomly managed to produce meaningful sentences with them. We conclude that the techniques and models described in this report could be used to generate plausible news headlines. However, for the purpose of generating conditioned headlines, we think that additional modifications are needed to obtain satisfying results.

Keywords: LSTM, RNN, deep learning, NLP, generated headlines

Sammanfattning

Denna rapport beskriver implementationen och utvärderingen av två språkmodeller med maskininlärningstekniken *deep learning*. Två olika modeller av återkopplande artificiella neuronnät (RNNs) implementerades, båda kapabla att generera rubriker till tidningsartiklar. En av modellerna fokuserade på generering av slumpmässiga (obetingade) rubriker, och den andra på generering av rubriker baserade (betingade) på en given nyhetsartikel. Båda modellerna tränades på ett dataset med ca. 500,000 par av artiklar och deras tillhörande rubriker. Att sammanfatta stora textstycken till mindre sådana, där huvudpoängerna i originalet bevaras, har många tillämpningar. Att snabbt och automatiskt kunna få fram sammanfattade versioner av exempelvis medicinska journaler, vetenskapliga papper, och nyhetsartiklar kan vara av stort värde för dess användare. Den obetingade modellen implementerades med ett RNN organiserat i flera lager. Cellerna i nätverken var av LSTM-typ. Detta nätverk producerade måttligt trovärdiga rubriker. Den betingade modellen implementerades med två stycken RNNs bestående av GRU-celler ordnade i ett *encoder-decoder*-nätverk med *attention*. Detta ger nätverket möjlighet att lära sig vilka ord som är viktigast vid rubriksättningen. Modellen lyckades visserligen identifiera viktiga nyckelord i artiklarna, men hade svårt att sätta ihop dessa i vettiga meningar. Vår slutsats är att teknikerna som beskrivs i denna rapport skulle kunna användas till att generera trovärdiga nyhetsrubriker. Trots detta tror vi att det krävs ytterligare modifikationer för att kunna generera trovärdiga betingade rubriker på ett tillfredsställande sätt.

Keywords: LSTM, RNN, deep learning, NLP, generated headlines

Glossary

n-gram Word frequency based language model. 3

ANN Artificial Neural Network. 1, 9, 13

DMN Dynamic Memory Network. 40
dropcost Method for reducing cost variation in sequential output. III, 3, 21, 25, 37, 41

GRU Gated Recurrent Unit. vii, 12, 25

LSTM Long Short-Term Memory. v, vii, xv, 11, 12, 24, 29

NLP Natural Language Processing. 1, 41
NNLM Neural Network Language Model. 14

RNN Recurrent Neural Network. v, vii, xv, 2, 10, 11, 14–16, 40
RNNenc The basic encoder-decoder model. 14
RNNsearch Encoder-decoder with an attention mechanism. xv, 3, 14, 16, 17, 24, 40
RNNsearch* Encoder-decoder with an attention but with sentences as inputs. 3, 17, 40

SGD Stochastic Gradient Descent. 7, 44

Contents

List of Figures

List of Tables

1
Introduction

Creating a computer program that generates syntactically and semantically correct text is a difficult task. One approach is to, given some previous words in a sentence, estimate the conditional probability of each word in the language being the next in the sentence. It is then possible to choose one that seems likely, and repeat the process with the newly extended sentence. For example, the sentence ”The cat crossed the...” would seem to be more likely to end with ”street” or ”bridge” than ”sea” or ”introduce”. The question is what the probability is for each of these words, or any word in the language at hand for that matter, to be the next word in the sentence? Modeling relationships of this kind within a language is in the field of natural language processing (NLP) often referred to as language modeling. Traditionally, language models have been constructed using either n-gram models [2] or rule-based models [3] [4] [5]. A problem with n-gram models is that they only take the $n-1$ previous words into consideration when generating the next. This means that the end of a sentence may not be directly related to the beginning. Moreover, as each permutation of the $n-1$ words yields a probability, both memory and computational requirements increase exponentially with respect to n. A problem with rule-based models is that they require hand-crafted word type tagging [5], which takes time and would have to be done explicitly for each modeled language.

When solving the language modeling problem with artificial neural networks (ANN), less time is spent on the complex task of describing *how* to detect patterns and meaning in the text. Instead, a model describing a network of interconnected artificial neurons is designed, whose structure and properties are chosen as to facilitate the network's ability to *learn* the desired characteristics of the text by example. For instance, if the network is presented with example sentences where ”The cat crossed the” is frequently followed by ”bridge”, then the internal state of the network should be automatically adjusted towards outputting a higher probability for ”bridge” as the next word in that particular sequence. Such a task requires a lot less engineering [6], as the network will take care of most of the “thinking” itself. In addition, a well designed model requires few or no modifications, before it can be trained and used for similar tasks, e.g. to generate headlines for news articles or for medical health notes.

The network described is used in a type of machine learning called *deep learning*, which has risen in popularity lately. It is a method in which multiple neural networks are stacked in layers, and doing so enables the network to learn compositional representations of, for example, words [6]. This can be intuitively understood as having each consecutive layer learn more abstract representations of the words, by

analyzing the simpler representations received from the layers before it. By making it possible for the neurons to loop back information to earlier stages of the sequence, we get something similar to memory. These kinds of neural networks where information can persist is known as recurrent neural networks (RNNs), and is the implementation used throughout this project. In the context of RNNs, the notion of deepness in a network often refers to the number of steps in the sequence. The ability to remember things makes these networks especially interesting in language modeling, as we try to generate sentences based on previous words the network has seen. Hence, using RNNs as a language model seems to be appropriate, as words towards the end of a sentence often depend on previous words.

Furthermore, looking at natural language processing, many approaches to sentence-decomposition uses a tree-like structure [3], where words represented by the leaves of a tree, are grouped together to form a sentence at the root. The widespread use of such methods to model language [4] [7] [5] [8], also suggests that a layered architecture, although not tree-structured, might be a suitable approach for constructing our network models.

The generation of text using the previously mentioned methods has many applications. A statistical model can be used for tasks where the next word in a sequence is to be predicted, for example auto-completion in smartphone keyboards, whereas a conditioned model could be used for abstractive summarization tasks. The task of abstractive summarization aims at providing summaries of text, without extracting whole sentences from it [9]. This poses the problem of having to generate new sentences, which incorporate the most essential information from the original text. Some possible use-cases for such summarizations are medical journals, scientific papers, and news articles. In these cases, an automated approach might be preferable because of its speed, or due to a reduction of manual labor.

1.1 Purpose

In this paper we implement two artificial neural networks, examining the advantages of using memory-capable cells and multiple network layers, to try and solve two natural language tasks involving text generation:

1. Generation of random news headlines
2. Generation of news headlines, based on some input text (e.g. part of an article)

Regarding the first goal, the headlines should be random in the sense that they are not based on some input text, but have the style of news headlines in general (e.g. formulated as statements or questions and not too long).

In the second goal, each headline will be conditioned on a news article. This will be done using an attention model so that the network can choose, on its own, which parts of the article that are most relevant for the headline.

1.2 Scope

The project will only involve training the networks on English text, since adding an additional language would make the task more complex. Learning multiple languages, the training time of the networks would probably be longer as well. Further, the networks will only be trained to generate headlines for articles, not for books or other written media. We aim to train the networks to recognize the specific style of headlines. To mix this style with e.g. book titles could lead to headlines that do not seem natural to find among articles.

The networks approximate an unknown conditional distribution that changes dramatically over time, i.e. the style of newspaper articles from the 19th century may be very different from those written in the past 10 years. Articles stemming from different sources may also be very different in style. This could potentially be a problem when measuring how good the networks generate an appropriate headline for a given article. Therefore, we will narrow down the time span for publication dates of the articles.

1.3 Related Work

Early attempts to approximate the propability of a sequence w of words, $P(w)$, relied on n-grams, that is, sequences of n consecutive words. The assumption is that languages obey the $n-1$:th order Markov property, i.e. that words in a text depend only on their index and the $n-1$ words before them. Using this assumption, one could instead aim to approximate the probability of a word w_t occuring at time step t, given all the previous words in the sequence, by calculating the probability of a word w_t occuring at time step t, given only the $n-1$ previous words. That is, $P(w_t|w_1^{t-1}) = P(w_t|w_{t-n+1}^{t-1})$, where the right hand side could be found using a simple frequency count. This gave

$$P(w_t|w_{t-n+1}^{t-1}) \approx \frac{C(w_t)}{\sum_{w_i \in \mathcal{V}} C(w_i)}$$

where $C(w_j)$ is the number of occurrences of the word w_j after the sequence w_{t-n+1}^{t-1} in the corpus. The subscript and superscript in w_{t-n+1}^{t-1} denote the first and last element respectively in the sequence w that are taken into concideration. These models have problems with scaling, considering that when n grows, the frequencies become very low. The assumption that languages have this Markov property is also trivially false for small n. Extending the models to cluster words [10] did however manage to achieve respectable perplexities on large corpuses using only 3-gram models.

1.4 Contributions

In this thesis we propose the objective modification technique dropcost as described in 4.3. We also propose a variation of RNNsearch, RNNsearch*, detailed in 3.4.

This variation has, to the best of our knowledge, never been implemented. Both of these contributions needs further study before any conclusions can be drawn.

1.5 Structure

The report begins with a background consisting of several chapters explaining the principles behind artificial neural networks, language models, and different training concepts. This is followed by a chapter describing the methods used, and details about the implementation. After that, there is a chapter describing the results from the project and finally a chapter discussing these results.

2

Artificial Neural Networks

There are many concepts related to the field of machine learning. In the following sections, we will describe the essential parts of the techniques used in this project. The explanations are not exhaustive, but should rather give the reader an understanding of how they work and what their purposes are.

2.1 Feed-Forward Neural Networks

Artificial neural networks model complex functions by learning from experience. No code is needed to tell the network explicitly what to do. Instead, it is a process of tuning certain parameters and looking at the result as data is input.

The fundamental building block of any neural network is the neuron. The neuron takes a number of inputs, processes them and outputs a result that decides its activation. What the output will be depends on the neuron's state. In its simplest form, the state consists only of a number of weights, one for each input. The weighted inputs are added together and put into an activation function. An activation function is a function that gives a smooth transition between two distinct values, like 0 and 1. The two extremes can be seen as two modes, on or off, for the neuron. In many cases, a bias is also added together with the weighted inputs. The bias can be thought of as a number telling us how easy it is to activate a certain neuron. If a bias is included, it is also part of the state.

To construct a network, we order several neurons in interconnected layers, where the activation from one layer is the input to the next. Layers have different names depending on where in the order they are placed. The first layer is the input layer, then come the hidden layers and lastly the output layer. The input layer can take a number of input values simultaneously, which is referred to as feeding the network. In our case, this will be a vector representation of a word where each element will be treated as a separate input value.

A more mathematical definition of the input of a neuron and its activation can be written as follows.

Let

$$z_k = \sum_j w_{jk} x_j + b_k \tag{2.1}$$

be the input of a neuron k (other than in the input layer), where the sum is over all neurons j in the previous layer, w_{jk} the weight from j to k, x_j the output of j and b_k the bias term of k. The activation of the neuron k can then be written as:

$$g(z_k) = \frac{1}{1 + \mathrm{e}^{-z_k}} \tag{2.2}$$

2.1.1 The Softmax Function and Sampled Softmax

The final layer in the network should output a probability distribution over the different classes. In our models, the classes are words and the distribution shows the probabilities for the next word. This distribution is calculated using a special activation function called the softmax function. It is a multidimensional generalization of the logistic function (see Equation 2.2) and very common for classification problems.

Let $\sigma(z)_j$ denote the softmax function, where z is the input and j denotes a specific class for which the softmax value is to be calculated.

$$\sigma(z)_j = \frac{\mathrm{e}^{z_j}}{\sum_{k=1}^{K} \mathrm{e}^{z_k}}, \text{ for all } k = 1, 2, ..., K \tag{2.3}$$

z_j is in our case the input to the neuron in the final layer, corresponding to word-number j in the vocabulary. The sum in the denominator is over all words k in the vocabulary, which consequently normalizes the sum of all softmax values in the final layer to 1.

As a consequence of using large vocabularies, the training complexity increases [11], in part due to the amount of parameters in the network. Using neural networks to translate text from one language to another, it has been shown that decreasing the size of the vocabulary also makes the translation performance decrease rapidly[11], i.e. the quality of the output probability distribution degrades. A possible remedy to this problem is to use a technique called sampled softmax.

Sampled softmax uses only a subset of all words in the vocabulary to calculate how the probabilities of words should be updated in each iteration during training. Since only a fixed amount of word probabilities are adjusted, the computational complexity is kept constant with respect to the vocabulary size [11].
Using sampled softmax, since only a fixed amount of classes are adjusted, regardless of vocabulary size, the computational complexity is kept constant with respect to the vocabulary size.

2.2 Training

The parameters of the network has to be tuned to produce the desired output. This is done like a classical optimization problem. Since we know what the desired outputs are, we can compare these with the actual output. The difference is formulated as a cost function and we minimize this by changing the weights and biases. Training a network this way is known as supervised learning [6].

2.2.1 Negative Log Likelihood

To get a measure of how well our model made its predictions, we are going to use the negative log likelihood for the target words.

$$\text{cost} = -\frac{1}{N}\sum_{\text{i=1}}^{N} \ln p_{\text{target}_i} \tag{2.4}$$

Instead of just seeing how far from 100% the probability was for a specific word, this more sophisticated function will improve the learning process when used together with the softmax function [12]. One can get an intuitive feel for the function by understanding that all output word probabilities will be between 0 and 1 and that the cost will decrease as the probabilities for certain words go closer to 1. Notice that we take the average of several predictions since the training is done in batches.

2.2.2 Stochastic Gradient Descent

During training, the task is to minimize the cost of the output as a function of the parameters, namely the weights and biases. An analytic way of finding the minima of a function is to differentiate it and look for its extreme points. But since the system is going to be overdetermined, this has to be done numerically. In our case, a method known as gradient descent will be used.

The graph of the cost function can be thought of as a multidimensional landscape, where each parameter corresponds to a dimension. At each iteration of the optimization process, a step is taken in the opposite direction of the gradient. The gradient is a vector of the partial derivatives of the cost function, indicating in what direction the function is increasing the most. Therefore, we want to move in the opposite direction of the gradient. How long we keep going before calculating the gradient anew is decided by the learning rate. Increasing the learning rate might make the learning process faster, but there is also the risk of diverging from a local minimum. However, this may in some cases be desired, since a local minimum not necessarily is the global minimum. After a fixed number of steps or when the results do not seem to improve, the learning process is over.

Calculating the cost over the whole data set each training iteration is computationally heavy [13]. To speed up the process, the data will instead be divided into several smaller batches. The network is then trained on one batch at a time. This updates the gradient more quickly but with less accuracy, since only a portion of the data set is considered. Doing so will most likely not generate the shortest path to a local minimum, but has the advantage of stochasticity. As the updates are somewhat off compared to the gradient of the whole data set, it is possible to escape local minima [14]. Using gradient descent with smaller batches (sometimes called mini-batches) of the data set is known as stochastic gradient descent (SGD).

2.2.3 Adam Optimizer

The Adam optimizer (Adaptive Moment Estimation) [15] is a gradient decent optimization algorithm. Its goal is to speed up the training while still being accurate. Since some parameters of the input data change more drastically and frequently than others from input to input, the value of a single fixed learning rate may be suitable for some parameters but not to others. Being close to the minimum in one dimension of parameter space would suggest that slow moves towards it are suitable (as explained in subsection 2.2.2), while on the contrary, some other parameters may be farther away from their respective minimum, suggesting that a faster learning rate would be suitable for them. Instead of using a single learning rate for all parameters or predetermined ones for each of them, the Adam optimizer computes individual adaptive learning rates for each of them. It incorporates both the exponentially moving average of the objectives' gradient and the squared gradient in order to estimate moments (mean and uncentered variance) of the gradient. The effect of the moments can be thought of as applying momentum to the gradient, preserving velocity longer when moving downhill in parameter space and reducing velocity quicker when moving uphill.

2.2.4 Backpropagation

Calculating the partial derivatives needed for the gradient can get quite complex because of all the different variables the partial derivatives depend on. This is especially a problem in large networks with multiple layers, as it would require calculating the output of the network as many times as there are weights and biases in it (using the definition of the derivative)[12]. One method to solve this is the backpropagation algorithm. This algorithm makes use of previously calculated values in order to calculate the new ones. It is based on the chain rule, and here follows an example, which shows how to calculate the partial derivatives needed to update the weights of the network.

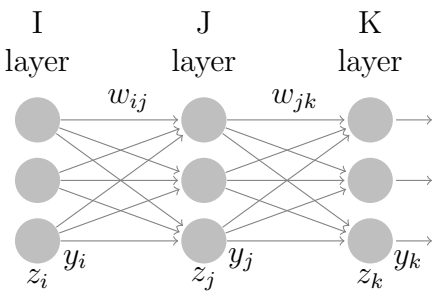

Figure 2.1: Three layer feed-forward neural network. Each layer labeled by its receptive index variable with inputs z, outputs y and weights w.

Let's say we have a feed-forward ANN as in Figure 2.1 [16] with layers I, J and K, where K is the output layer of the network. Let w_{jk} be the weight from neuron j in layer J to neuron k in layer K, y_j the output of j and $z_k = \sum_j w_{jk} \cdot y_j$ the input of k. Also, let $\frac{\partial E}{\partial y_k} = r_k$ be the partial derivative of the error E, with respect to the output y_k (i.e. the partial derivative of the error function). In our case, the error function is defined by the cost function in Equation 2.4

In order to update the weight w_{jk}, we need to calculate the partial derivative $\frac{\partial E}{\partial w_{jk}}$. We can start by noting that $\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \cdot \frac{\mathrm{d}y_k}{\mathrm{d}z_k}$. The partial derivative of the output y_k with respect to the input z_k is the derivative of the activation function $f(z_k)$ (w.r.t. z_k), so

$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k} \cdot \frac{\mathrm{d}y_k}{\mathrm{d}z_k} = r_k \cdot f'(z_k)$$

This is why the activation function must be differentiable. Here we made use of r_k, which is done only for the output layer K. When propagating the error backwards further into the network, we would like to use the equivalent value of the partial derivative of the error E with respect to the output y in any layer other than K. Since the forward-propagated signal branches onto several neurons, we sum over these neurons. For the next layer, J, we obtain

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial z_k} \frac{\partial z_k}{\partial y_j}$$

As $\frac{\partial z_k}{\partial y_j}$ is the change in input of k with respect to the output of j, it can be substituted by the corresponding weight, w_{jk}, and we obtain

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial y_k} w_{jk} \tag{2.5}$$

Now, we can continue by writing $\frac{\partial E}{\partial w_{jk}}$ as

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial z_k} \cdot \frac{\partial z_k}{\partial w_{jk}}$$

Notice how $\frac{\partial z_k}{\partial w_{jk}} = y_j$. This might seem a bit puzzling, but can be motivated by asking the question "how does the input of neuron k, z_k, change with respect to the weight w_{jk}?". Since $z_k = \sum_j w_{jk} \cdot y_j$, the change is y_j, and we write

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial z_k} \cdot y_j \tag{2.6}$$

Equation 2.6 shows how to calculate the desired partial derivative specifically for a weight w_{jk} from layer J to K. The resulting value can be used to update the weight, as shown in Equation 2.7

$$w_{jk_{\text{new}}} = w_{jk_{\text{old}}} - \eta \frac{\partial E}{\partial w_{jk}} \tag{2.7}$$

where $w_{jk_{\text{new}}}$ and $w_{jk_{\text{old}}}$ refer to the updated and old value of w_{jk} respectively, η is the learning rate and the minus sign before it indicates that we update in the opposite direction of the gradient. This is done for all weights w_{jk}. Now we continue backwards in the network by repeating the process. We can calculate the partial derivative of the error with respect to the weights in the next layer, w_{ij}, by

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E}{\partial z_j} \cdot y_i$$

similarly to what we did in 2.6. For the purpose of explaining the algorithm, it suffices to say that $\frac{\partial E}{\partial z_j}$ can be found by using the previously calculated value in 2.5. For the same reason, biases of the neurons were not taken into consideration.

2.2.5 The Vanishing Gradient Problem

As explained in subsection 2.2.4, calculating gradients of weights near the input layer requires calculating gradients of the weights between subsequent layers. This comes with a problem, namely the vanishing gradient problem. It becomes especially evident for deep [12] and recurrent networks [17], the latter being discussed in section 2.3.

Each layer in the network works as a function of its input. The output layer of the network can therefor be considered to be a function of some other functions, corresponding to the previous layers. Using a logistic function as activation function, the derivative of a neuron's output (w.r.t. to its input) lies in the range $(0, \frac{1}{4}]$ [12]. When many of these small terms are multiplied, as when calculating the gradient in a deep or recurrent network, the product becomes smaller and smaller for each multiplication. In effect, the gradient of the error w.r.t. the weights near the input layer can become impractically small when training [17] - the gradients vanish. As the gradient is calculated using weights as well, the opposite may happen for large enough values of the weights, making the gradient explode [12].

One solution to the vanishing gradient problem is to use a so called *constant error carousel* with gates, suggested by [17]. The carousel ensures that the gradient of the activation remains constant, while the gates ensure that only relevant information is corrected for during the backpropagation.

2.3 Recurrent Neural Networks

Recurrent neural networks (RNN) is a type of neural networks that are used for modeling time sequences, such as words in a sentence. By remembering their previous state, s_{t-1}, they can make better predictions of what comes at time t. The problem with simply looping back information is that the network quickly forgets what it has just seen . Longer time sequences will thus be harder to model. A solution is to introduce a number of gates which control what the network should remember and forget. This helps with the vanishing gradient problem discussed in 2.2.5, and to

this end, the Long Short-Term Memory cell was invented [17]. The output of both gated and plain RNNs will, for input x_t, be referred to as RNN(s_{t-1}, x_t).

2.3.1 Long Short-Term Memory

Long Short-Term Memory, LSTM, is a special network architecture that has the ability to remember certain values for longer time, while discarding others immediately. This is accomplished by using four so called gates, which are trained to control the flow of information within the cell, sometimes also called the hidden unit [17]. The cell has a memory called the cell state that contains the information from previous steps, see Figure 2.2. The gates incorporate parts of the old state when calculating the new. Each iteration part of the state is either retrieved, overwritten or kept unchanged for the future. We can describe the LSTM cell and its state as C_t^l, where t is the time step, and l is the layer that it resides in.

The cell takes two inputs. The first is its own cell state from the previous time step, and the second is the output from the previous layer at the current time step, concatenated with its own output from the previous time step. In the case where the cell is in the first layer, the input is the word vector itself. It will then use these inputs to calculate the output as well as to update its own state for the next time step. In our case, each time step will be used for calculating one word vector.

The forget gate layer is there to decide what to keep in the cell state and what to forget. It uses a sigmoid layer, i.e. a layer with the logistic function as activation function. We will call the output from the forget gate $\mathbf{f}$. Let's also use $\mathbf{b}$ for bias vectors, $\mathbf{x}_t$ for the cell's input, and $\mathbf{W}$ for weight matrices inside gates.

$$\mathbf{f} = \sigma(\mathbf{W}_f \cdot [\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_f) \tag{2.8}$$

The input gate layer uses both a tanh layer to calculate new inputs into the cell state and a filtering sigmoid layer to decide what to save. Like a sigmoid layer, the tanh layer is correspondingly a layer that uses tanh as activation function. Let's call the output of the tanh layer $\mathbf{g}$ and that of the sigmoid layer $\mathbf{i}$, we now get:

$$\mathbf{i} = \sigma(\mathbf{W}_i \cdot [\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_i) \tag{2.9}$$

$$\mathbf{g} = \tanh(\mathbf{W}_g \cdot [\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_g) \tag{2.10}$$

$$\mathbf{C}_t^l = \mathbf{f} \odot \mathbf{C}_{t-1}^l + i \odot \mathbf{g} \tag{2.11}$$

where $\odot$ denotes element-wise multiplication. After calculating the new cell state, there is a final output layer to decide what to output from the cell state. It uses a tanh neural net layer on the cell state as well as a sigmoid layer on the input to filter the information from the tanh layer. The output $\mathbf{h}_t^l$ can then be calculated. The filtered output, $\mathbf{o}$, from the sigmoid layer is obtained by:

$$\mathbf{o} = \sigma(\mathbf{W}_o \cdot [\mathbf{x}_t, \mathbf{h}_{t-1}] + \mathbf{b}_o) \tag{2.12}$$

$$\mathbf{h}_t^l = \mathbf{o} \odot \tanh(\mathbf{C}_t^l) \tag{2.13}$$

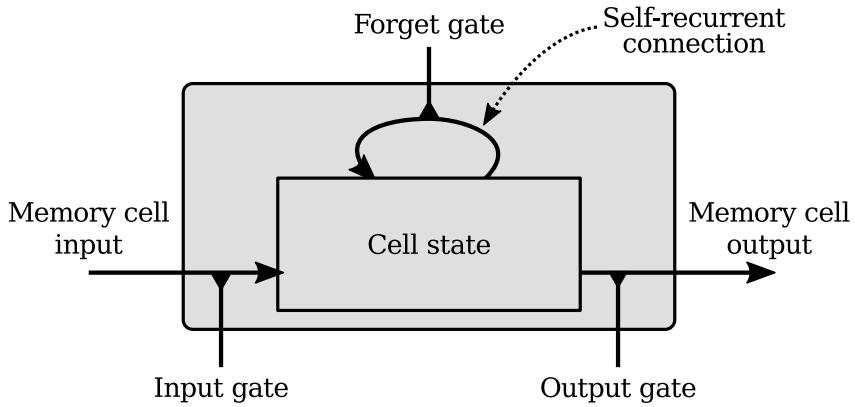

Figure 2.2: Schematic drawing of an LSTM cell with its input, output, gates and cell state. Notice that the cell state has a recurrent connection.

2.3.2 Gated Recurrent Unit

Another type of cell or hidden unit that can be used in networks is the Gated Recurrent Unit (GRU), introduced as a simpler version of the LSTM [18]. While the LSTM cell has four gates to control the flow of information, this model instead only has two gates; a reset gate as well as an update gate. Despite being far less computationally heavy than the LSTM cell, empirical studies have shown it to perform comparably [19].

2.4 Deep Neural Networks

A minimal network would consist of only one input layer and one output layer. Using multiple hidden layers in between are referred to as a deep neural network, hence the name deep learning. Having multiple layers, the network may gain performance. However, there seems to be no precise explanation of when this is true, i.e. no precise relationship between performance and the number of layers. Testing multiple setups of networks is therefore common practice.

3
Language Models

The two main goals of the project are to create a program that generates random headlines and another that generates headlines based on the beginning of a news article. To this end, two different language models are used, namely an unconditioned language model for the first goal and a conditioned one for the second goal. The following subsections will describe each model and their corresponding approaches to solving the problems.

3.1 Word Embeddings

Unlike images and sound, words in a language do not carry any information, but instead act merely as tokens. Thus, a space of objects represented as a sequence of words have no natural metric, and the dimensionality of the space will grow exponentially with the length of the sequence. Trying to approximate some probability mass function on such a space is notoriously hard, simply because the probability for any one element will be close to zero. This is known as *the curse of dimensionality* [20].

To remedy this problem, [20] introduced the concept of distributed representations of words, or word embeddings, where words are mapped to elements in $\mathbb{R}^m$. Typically $m \in \mathbb{N}$ is significantly smaller than the vocabulary in the language, and the vectors are tuned to make some problem well-conditioned. In the contexts of ANN, it is common to represent words as $w \in \{0,1\}^{|V|}$ with $|w| = 1$ and to insert a simple linear embedding layer that multiplies w with $W \in \mathbb{R}^{|V| \times m}$. The weights W are then trained together with the network. The vector w will be refered to as a *sparse word representation*.

W can be initialized in many ways, either randomly using some sensible distribution or by first training an ANN to minimize some problem based on the *distributional hypothesis in lingustics* introduced by [21] and using these weights in a more interesting problem. Notable such pretraining models have been constructed by [22] and [23].

3.2 The Unconditioned Language Model

The starting point of language models is assigning probabilities to sequences of words. Once this is achieved, one can use this distribution to either draw sentences

or judge existing sentences. If w is a sequence of words, w_i the i:th element of that sequence and w_i^j a sub-sequence from i to j the probabilities can be computed by using the chain rule on the prefixes in the sequence, i.e.

$$P(w) = \prod_t P(w_t | w_1^{t-1})$$

To estimate the conditional probabilities $P(w_t|w_1^{t-1})$ as a multinomial distribution over the vocabulary $\mathcal{V}$, an RNN is trained to minimize the negative log likelihood of a softmax output.

The standard model introduced by [20] used a feed-forward neural network and is often referred to as NNLM. This model was later extended by [22] to use an RNN instead. In its most basic version, the model is given by the following equations

$$\begin{aligned} s_t &= f\left(\mathbf{U}w_t + \mathbf{W}s_{t-1}\right) \\ y_t &= \sigma\left(\mathbf{V}s_t\right) \end{aligned}$$

where $U \in \mathbb{R}^{|V| \times m}$ is a word embedding, $W \in \mathbb{R}^{m \times m}$ simply some trainable weights, σ is the softmax function, $V \in \mathbb{R}^{m \times |\mathcal{V}|}$ a linear mapping and f some activation function. Of course, this simple activation RNN can be replaced by an arbitrary recurrent cell which output can than be mapped onto the vocabulary and given to the softmax function.

3.3 The Conditioned Language Model

Many tasks can not be solved by a simple language model and require the generated sentence to be conditioned on some input sequence. Even though RNNs are well suited for processing sequential data, they have problems taking sequences as inputs since they require the input dimension to be known and fixed. In statistical machine translation, this problem has been successfully solved by different variations of *encoder-decoder* networks [18][24].

These models train two RNNs together — one that encodes an input sequence to a fixed sized vector and one that decodes that vector to a target sequence — using standard gradient based optimization through the whole system. These basic models will be referred to as RNNenc. Later these models have been extended using an attention mechanism introduced by [25] where instead of trying to encode the original sequence in a single vector, the encoder gives a list of vectors relevant to different parts of the original sequence. The decoder is trained to decide which vectors are relevant for the next word given the currently generated sequence. This model will be referred to as RNNsearch.

3.3.1 The Basic Encoder-Decoder Model

In RNNenc, the context c representing the input sequence $x_1, \ldots, x_{T_x}$ is typically encoded as the internal state of an RNN that has had the input fed through it,

either forward as in [18] or backwards as in [24].

The conditioned probability that the word $w \in \mathcal{V}$ should appear next in the generated sequence $y_1, \ldots, y_t$ is then modeled as

$$P(y_t = w|y_1^{t-1}, x_1^{T_x}) \approx \sigma\left(V \text{ RNN}\left(s_{t-1}, \begin{bmatrix} y_{t-1} \\ c \end{bmatrix}\right)\right) \cdot w'$$

where $V \in \mathbb{R}^{m \times |\mathcal{V}|}$, s_{t-1} is the internal state of the RNN after the model has generated distributions for the words $y_1, \ldots, y_{t-1}$ and w' is a sparse word representation as described in 3.1. Note that the context vector c has been concatenated to the input. Further, s_0 is $0 \in \mathbb{R}^m$ and y_0 is a word g not appearing in the target vocabulary.

This is essentially the same as the unconditioned model only that the input vectors have been extended to not only carry information about the word, y_i, but also a condensed version , c , of the input sequence. During training the target words y_i are not picked from the output but instead read from the target sequence, when generating sequences the output distribution is used to pick the next word to give to the model.

Sometimes when discussing implementation details, the block of the weight matrices that deals with the concatenated context is factored out, as is done in [25].

3.3.2 Encoder-Decoder with Attention

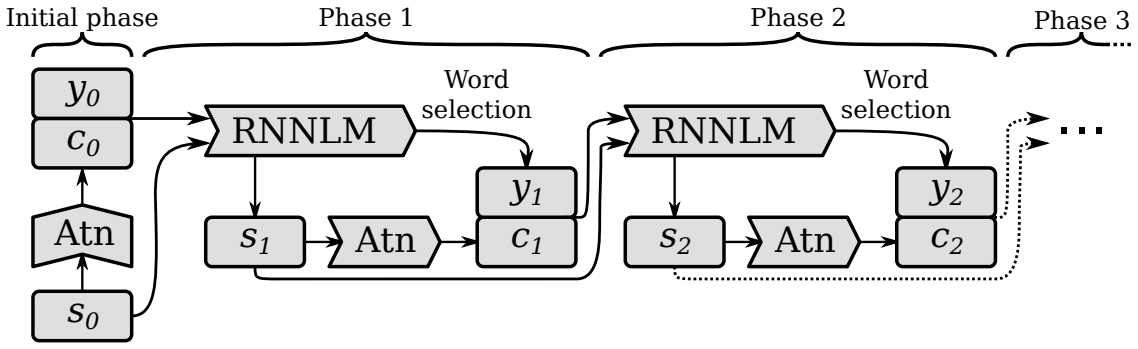

Figure 3.1: Schematic description of the data flow, *RNNLM* refers to the unconditioned language model and *Atn* to the attention model. The phases are the steps of the model and the word selection is done from the softmax output during generation and the target sequence during training.

The encoder in RNNSearch instead produces T_x annotation vectors, h_i, consisting of the concatenation of the internal states of one RNN run forward on the input sequence up to word i and one run backwards to the same word. Alignments e_{ij} between x_i and y_j are then calculated and later, after a softmax layer, used to

approximate the likelihood of the word y_j aligning to annotation h_i, i.e

$$\begin{aligned} P(y_j \text{ aligns to } h_j) &\approx \alpha_{ji} \\ \alpha_j &= \sigma(e_i) \\ e_{ij} &= a(s_{i-1}, h_j) \end{aligned}$$

where $a(s_i, h_j)$ is some trained alignment model. A context c_j is then computed as the expected annotation

$$c_j = \alpha_j \cdot h$$

and used in a decoder entirely analogous with the decoder in RNNenc. An overview of the data flow is given in Figure 3.1 where *Atn* refers to the attention model described here. The data flow of the attention model can be viewed in Figure 3.2.

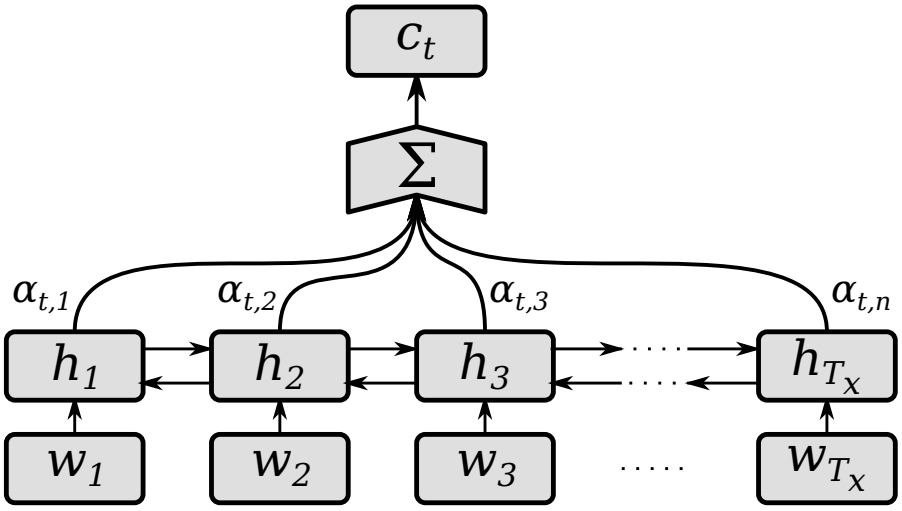

Figure 3.2: Data flow for computing the output of the decoding RNN in RNNsearch. The summation is the scalar product used to calculate the expected annotation c_t.

3.4 A Variation on the Conditioned Language Model

Conditioning the model as in 3.3 introduces problems with scaling [26][27], partly because the increased amount of attention contexts and time steps. It might also be because of the underlying assumption of RNNsearch that generated words align to one specific word.

When setting a title on a text it might thus be useful to try to reduce both the number of time steps and attention contexts by considering the text not as a sequence of words but instead as a sequence of sentences. This also helps motivate the plausibility of the assumption, i.e it's more plausible that words in the title aligns to particular article sentences rather than particular article words.

Sentences can be effectivly represented as vectors [28] and when that has been done, those vectors can be fed to RNNsearch with no modifications. This is, to the best of our knowledge, a novel model which we introduce and call RNNsearch*. Further, the whole model can still be trained by gradient based optimization algorithms via backpropagation.

4
Training Concepts

There are many concepts in the context of training ANNs, both regarding the training data and the elements of the network. All of these concepts affect the final performance and the following subsections will explain the key ideas behind those used in this project.

4.1 Hyperparameters

Some parameters cannot be learned, but have to be set manually. These so called hyperparameters are mainly related to the specific learning algorithm and structure of the network. Tuning these is done not only to improve training results, but also to generalize the model so it can make better predictions on cases it has not seen before. Learning the training set too closely is known as "overfitting"and choosing the right hyperparameters can prevent this. Following is a more detailed explanation of the different hyperparameters and their effect on the model.

4.1.1 Layer Size

When choosing the number of hidden units in each layer, it is important to choose a size that is big enough [29]. Larger than optimal values will not hurt performance, but it will increase the time it takes to train the network. A larger network also means more parameters available for tuning which can lead to overfitting [12].

4.1.2 Training/Validation Data Ratio

To acquire good results, it is essential that the network has high-quality and extensive data to learn from. We also need data which can be used to validate the network's performance. One hyperparameter is then how we should distribute our data set between training and evaluation. It is important to have a validation set that is disjoint from the training set, as it gives an indication of the networks ability to generalize. If the validation was to be done on the training set, it would have been difficult to tell whether the network was overfitting or not.

4.1.3 Number of Epochs

One epoch is one run through the data set and running several epochs could yield better results. Still, we do not want the model to learn the data set too closely. The number of epochs will also affect how long the training takes.

4.1.4 Learning Rate

As discussed in subsection 2.2.2, it is possible to increase the learning rate to make more radical changes to the variables each optimization step. A low learning rate will not improve the cost as much as we want, but a too big step might never converge to a minimum. One way of choosing a fitting learning rate is to look at the graph of the cost function, and see whether it decreases satisfactorily each time step. It is also common to decrease the learning rate as the training progresses. In that case, when to begin the decrease and by how much are also parameters to decide.

4.1.5 Word Dimensionality

Another hyperparameter is the input size of the network, i.e. the dimensionality of the word embeddings. If pre-trained word vectors are used, this hyperparameter is equivalent to their dimensionality.

4.2 Regularization

The process of reducing overfitting while maintaining the size of the network is called regularization and the following paragraphs will explain the two main regularization techniques used in this project.

4.2.1 Dropout

Dropout is a technique described in [30] that aims to prevent overfitting. By removing some neurons and their connections within the network during training (with a probability p), it can be seen as training random subsets of the whole network. When the training has ended, no dropout is applied, meaning that all neurons and connections are present. The weights may then be scaled by the probability p, which makes the output of a neuron equal to the expected output during training with dropout. The result is a network consisting of many sub-networks that have been trained differently from each other. This has been shown to increase performance on various machine learning tasks. While capable of increasing the effectiveness of the network, dropout also increases the training time, since only a fraction of the whole network is trained each iteration. The dropout probability p is a hyperparameter.

4.2.2 Early Stopping

One way to detect overfitting is to compare the performance of the network on the training set with that of the validation set. In the beginning of the training process, the network should improve on both sets. However, after some training, the performance on the validation set may become worse, while that of the training set keeps on becoming better. This could indicate that the network is learning the training set too specifically and cannot generalize to the validation set. To prevent this, we can simply stop training when the performance of the two sets begins to diverge. However, it might be difficult to determine when to stop, as divergence

could be temporary. One way to circumvent this is to save the network at different checkpoints during training and then pick the one which has best performance on the validation set.

4.3 Dropcost

When optimizing sequence-generating models using the average negative log likelihood (see subsection 2.2.1) of the sequence, there's a risk that the average cost can be lowered by diverging from the target sequence in some systematic manner. To penalize this we propose a variation of the regularization technique *dropword* introduced by [31]. We will refer to this variation as *dropcost*. While *dropword* hides input words from the model trying to penalize over dependence on part of the input, *dropcost* instead waives costs in the output sequence, benefiting models with low variation in the cost for the generated outputs. The expected fraction of the words in a target sequence with wavered costs during training will be referred to as the *dropcost factor*.

4.4 Mini-Batch Size

Choosing to train on one word pair each iteration would adjust the parameters towards better predicting the next word, given the first word in particular. On the other hand, training on the whole data set each iteration would adjust the parameters towards better predicting the next word for all words in the set at once. The first approach has the advantage of stochasticity, meaning that the parameters can escape local minima more easily, while the latter approach speeds up the process by enabling parallel calculations of several words. A commonly used technique is to train on a so called mini-batch each iteration, which contains a subset of the whole data set. Choosing the size of the mini-batch is a balance between efficiency and effectiveness.

5
Method

This section describes how the proposed models were implemented, what tools were used for doing so and how the validation of the outcome was conducted.

5.1 Data Sets

A set of over 540,000 articles with corresponding headlines was sampled for training and validation data in both models. The data set was produced by Congle Zhang and Daniel S. Weld, by querying the Bing news search engine for news articles based on the titles in RSS news seeds. This was done using daily news seeds from multiple newspapers between January 1 to February 22, 2013 [32].

5.1.1 Preprocessing the Data Sets

The data set was pre-processed before tokenization to reduce the amount of unique tokens generated from it. Also, some incomplete entries (titles without articles etc.) were removed. All text went through the following processing before being tokenized and used in the models:

lower-casing All text was converted to lower-case.

number-normalization All digits were replaced by zeros.

5.2 The Unconditioned Model

This section describes the implementation of the language model from section 3.2.

5.2.1 Data processing

Only titles from the data set were used, and tokenized using a tokenizer from the Natural Language Toolkit (NLTK). This tokenizer provides multiple tools such as separating shortened word combinations, for example ”couldn’t” becomes two tokens: ”could” and ”n’t”. A vocabulary was then created, containing only tokens occurring more than 5 times. Less frequently occurring tokens were replaced with the ”unknown-token”, ”_UNK”.

5.2.2 Implementation

Figure 5.1 shows the final architecture of the unconditioned language model. A word vector is put through one or several LSTM layers and the output from the last one will be used as input to the softmax function. The softmax function gives a probability distribution over the dictionary for what the next word might be. When generating the headlines, the program will choose a word from this distribution and then use it as input at the next time step. Also, at each time step the LSTM cells' states are updated.

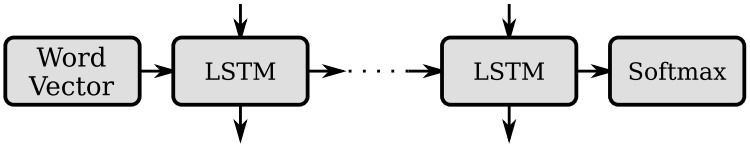

Figure 5.1: Final architecture of the unconditioned language model. A word vector propagates through several LSTM layers and finally through the softmax function, giving a distribution of what the next word might be. The vertical arrows indicates how the LSTM cells' states move between time steps.

The different layers are connected with weights, as described in 2.1. These are the parameters that get tuned during the optimization step together with the word embeddings. It is worth noting that the model trains on several sequences at once and calculates the average cost. To accelerate the training process, the word embeddings are initialized with pretrained word2vec vectors [1]. These have been trained on part of Google News' data set.

5.3 The Conditioned Model

This section describes the implementation of the RNNsearch language model from subsection 3.3.2.

5.3.1 Data processing

In the conditioned model, all articles and titles were first converted into tokens using a simple white-space tokenizer. Two different vocabularies were created, consisting of the 40,000 most common words[1] found in the articles and titles respectively. Each article-title-pair was then truncated to a fixed length of 200 tokens for articles and 50 tokens for titles.

5.3.2 Implementation

The conditional language model is implemented using the seq2seq-library from Tensorflow The model is an implementation of an encoder-decoder network described

[1] A fixed vocabulary-limit was used to allow more direct control of the memory requirements

in subsection 3.3.1 with the addition of attention as described in subsection 3.3.2. Just like in [25], the implementation uses a single feed-forward tanh layer as the alignment model.

When training the network, the article-title-pairs were fed through the encoder-decoder network in batches of 16 or 32 pairs, depending on the other hyper parameters' impact on RAM usage. The network was implemented with GRU-cells, ranging from 512 to 1024 cells in 2 or 4 layers. For each batch, a loss was calculated, and network parameters (vector representations and annotations) adjusted using the Adam [15] optimizer. Every 100 batches, the perplexity was noted and a checkpoint of the model was saved.

5.3.3 Dropcost

To evaluate dropcost two networks identical to the conditioned network with 1024 cells and 2 layers but with dropcost factors of 0.15 and 0.30 respectively were trained. Then 100 articles from the validation set where sampled and headlines generated by all three networks. The 10 articles on which the network without dropcost was judged to have performed the best were selected and the networks performance on these 10 articles where then qualitatively compared.

5.4 Environment

The models described in this section have been implemented using the machine learning framework TensorFlow [33] by Google. Data processing and other book-keeping tasks have been implemented in Python 2. Bash scripts were used to manage the execution of simultaneous jobs on the compute server.

The training and evaluation of the models were done either on a GPU-equipped workstation (when not limited by the RAM of the GPU), or on Chalmers' Machine Learning group's Compute server. Specifications for these machines can be found in Table 5.1. Notable is that the NVIDIA GTX 970 GPU can be described as a more consumer oriented product, both regarding price and performance, while the NVIDIA Titan X is oriented more towards enthusiasts and scientific calculations.

Table 5.1: Machine specifications

Specification	Workstation	Compute server
CPU	Intel i7-4790K ($8 \cdot 4$ GHz)	Intel i7-5930k ($12 \cdot 3.5$ GHz)
GPU	NVIDIA GTX 970 (4 GB)	4x NVIDIA Titan X (12 GB)
RAM	16 GB DDR3	32 GB DDR4

5.5 Optimizing Hyperparameters

There are many parameters that can be tuned manually to increase the performance of the models. Getting all the parameters right can take a lot of time and because of that we are going to focus on the ones that clearly increase performance and are easiest to tune. The process can best be described as trial and error, where a couple of models are trained with different values and their results compared. However, there are some heuristics to follow.

The size of a network can affect its performance, where a larger network can learn more sophisticated patterns than smaller ones. A large network also has higher probability of overfitting and therefore often has to be compensated with increased dropout. One way to see if the model is overfitting is to compare the cost for the training and validation set. These should both be decreasing when training and not differ too much from each other.

Besides network size and dropout ratio, the most important parameter to tune is the learning rate. Typical values for our case is in the range of less than 1 and greater than 10^{-6} [29]. When trying different learning rates it is often more constructive to test different multiples of the original learning rate than just increasing it with some decimal value. The graph of the cost function can be inspected to see whether the training progresses as desired or not. A too steep cost function often indicates a too big learning rate and a cost function that looks more linear often means that the learning rate is too small.

When it comes to mini-batch size and number of training epochs, these parameters mainly affect the program's running time and will not be crucial to the end results. Together with network size, they will however decide how long it takes to go through the data. Therefore, they are still important since time is limited.

5.6 Evaluation

In order to evaluate the performance of the models implemented for the project aims, the following two measures were considered.

5.6.1 Perplexity

Perplexity is a measurement used to evaluate how well a model predicts a sample. Many research papers use this measurement, so it is useful if we want to compare our work with that done by others in similar projects. In our case we calculate the per-word perplexity with the formula $e^{\frac{\text{cost}}{\text{words}}}$.

5.6.2 Turing-like Test

To further evaluate the the first goal (generating random news headlines), a Turing-like test was used. In 1950, Alan Turing devised a test for validating how well

machines can imitate humans, by letting a human try to determine if a conversation is held with a computer or another human [34]. The test used in this project consisted of 100 random generated headlines and 100 random non-generated headlines from the training set, presented as 20 smaller tests with five generated and five non-generated headlines in each. A smaller test would be randomly chosen and presented to the participant, who were then asked to, for each headline, indicate how certain they were that a given headline was generated or non-generated, using an integer scale from 1 (”non-generated”) to 5 (”generated”). This way, the plausibility of the headlines could be evaluated in finer detail than when having to choose between ”generated” or ”non-generated” only. The answer values for each headline would then be summed and averaged. An average value of 3 would indicate that a headline was as plausible as unplausible. The best result achievable for a generated headline would be 1, where as the worst would be 5, with respect to the project aims.

There were no restrictions made on who could participate in the test or how many smaller tests could be taken by the same person. This was justified by not showing the correct answers in the end of the test. Each of the smaller tests consisted of the same headlines every time it was presented, but in random order, as this may contribute to less bias when answering.

6
Results

After implementing and training the models, they could be evaluated using the techniques described in Evaluation. The following sections describe how the different models ultimately performed. In order to give an indication of their performance, some examples of generated headlines are shown as well.

6.1 The Unconditioned Language Model

The unconditioned model was trained on an Nvidia GTX 970 graphics card. The biggest model able to fit on one of theses cards had about 45,3 million parameters. This value represents all adjustable weights and biases: LSTM cell states, word embeddings, weights between the different layers in the model and the neurons' biases. Three separate models were trained with roughly the maximum number of parameters, differing only in number of layers and layer size. The different configurations can be seen in Table 6.1.

Table 6.1: Model configurations and number of parameters

Number of Layers	Layer Size	Parameters
1	925	45362000
2	800	45309700
3	720	45239940

To reduce memory requirements the Adam optimizer was used [15]. Each model was trained for 25 epochs with a batch size of 50, and the learning rate was kept constant at 0.002. 10% of the data set was used for evaluation. Using the method described in subsection 5.2.1, the vocabulary ended up with a size of 33300 words.

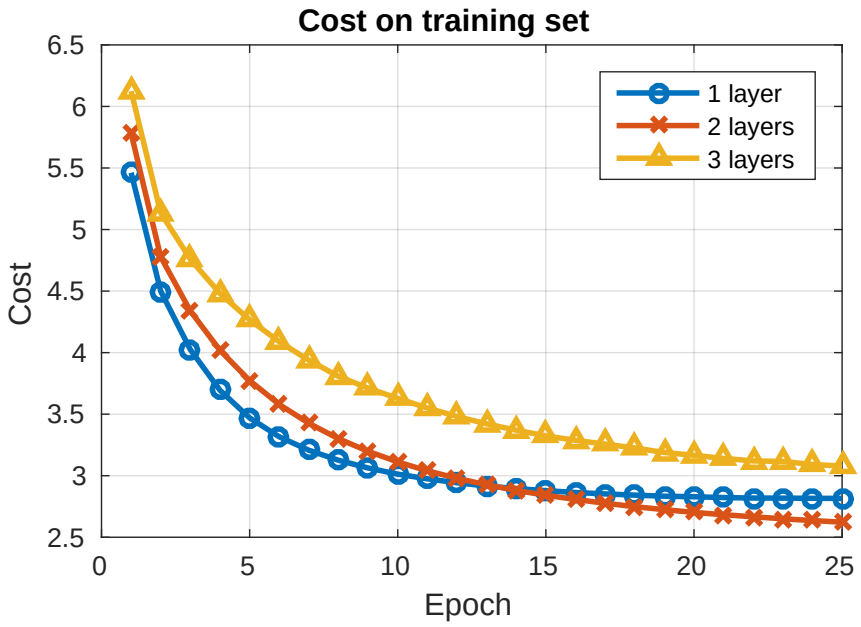

Figure 6.1: Cost on training set during the optimization process. Three different models were evaluated with roughly the same number of parameters, differing only in number of layers and layer size.

Figure 6.1 shows how performance on the training set changed as training prolonged. Looking at the results, it can be seen that the 2-layer model yields the lowest cost in the end, though not far from the cost of the 1-layer model. The 3-layer model does not seem to be optimal in this case, even though it probably would benefit from longer training time. More important is however the performance on the evaluation sets where the models show if their knowledge can be generalized.

To get comparable results for the validation set, 50% dropout was added to avoid extreme overfitting. Apart from this, the models were exactly the same as above.

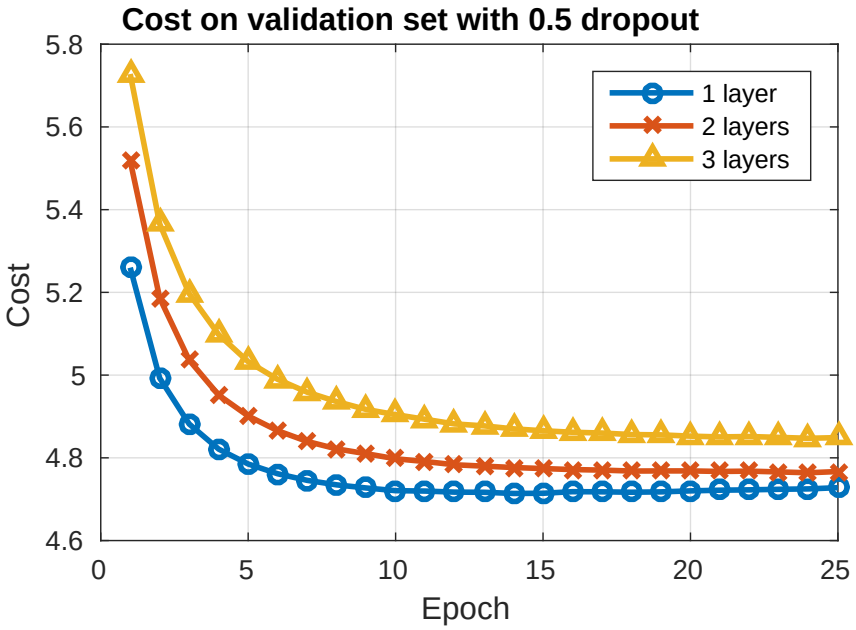

Figure 6.2: Cost on validation set during the optimization process. Three models with different number of layers were compared against each other. All three models had 50% dropout added to avoid overfitting.

In Figure 6.2, the clear winner seems to be the 1-layer model. Better performance would probably be achievable if more tuning had been done, but we did not want to over-engineer anything since these results could have been used as a baseline when evaluating the sequence-to-sequence model. Lastly, in Table 6.2, we show the perplexity at the epoch with lowest validation cost.

Table 6.2: Validation and test perplexity at epoch with lowest cost

Number of Layers	Epoch	Validation Perplexity	Test Perplexity
1	14	111.48	109.67
2	24	117.21	115.54
3	24	127.39	125.51

6.1.1 Pretrained Word Vectors

Our word embeddings were initialized with 300-dimensional vectors pretrained on about 100 billion words from different news articles [1]. Figure 6.3 shows what impact this had on our 1-layer model.

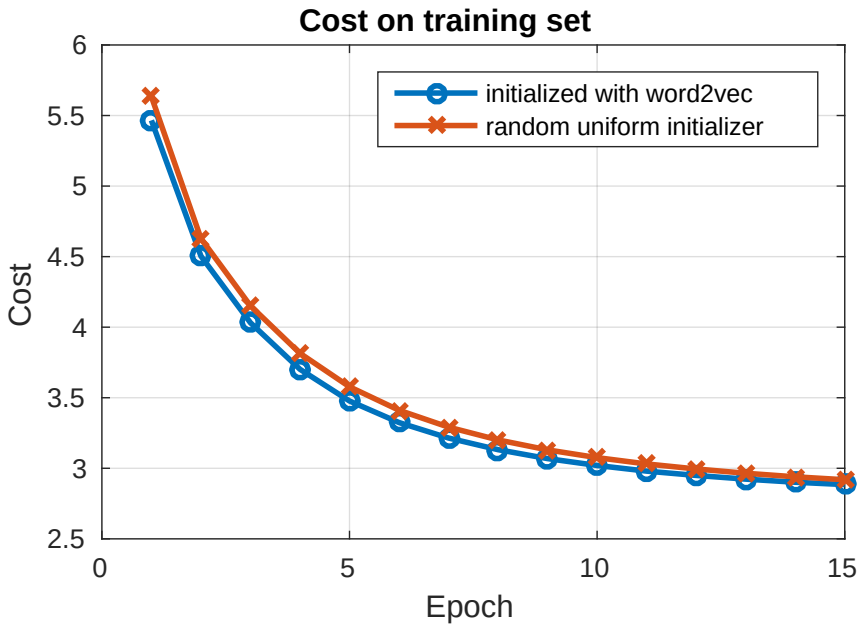

Figure 6.3: Cost on training set when using different ways of initiating the word vectors in a 1-layer model. Pretrained word2vec [1] vectors were compared against a random uniform initializer between -0.1 and 0.1

Hardly any improvement at all was gained by using the pretrained vectors. A more elaborate discussion of why this might be will take place in section 7.1.

6.1.2 Evolution of Samples While Training

It is interesting to see how the sampled headlines change while the model trains. Following samples were taken from the 1-layer model that achieved the lowest perplexity.

Training for just 1 epoch gives the following result:

Epoch 1

```
wondered cemented obama scientist
sens straw blue jays again john mayer as i agree on permanent beat
psg crowned mainstream' pickup in delhi
'fiscal cliff' wins big picture as corporate each and worst of...
who was prince harry & this co-founder? '' says retirement to not...
```

The model strings together some words but most of the headlines are still quite random. After a few more epochs the structure seems to be getting better:

Epoch 3

```
worst of the year
mad men who battles breast cancer loss
vodafone building deals soar on playstation 0 spreads to more than...
preparations for pope benedict to resign as pontiff
sun profits beat #00: unemployment drops but remain in panel
```

The headlines have become shorter and perhaps even started to make some sense. Further training keeps improving this:

Epoch 10

```
hindsight
new mario lopez who speak at rutgers value
uk detective sheds light on horsemeat scandal
azarenka reaches second round
beware slow weather by jones' wins (from the bolton news)
```

After the 10th epoch the grammar is better and we are getting a few headlines that do not seem to be generated. None of the headlines are taken directly from the data set, but a few of the words commonly occur together in the same order. Training for more than 10 epochs does not improve the quality of the samples remarkably in our case.

6.1.3 Turing-like Test Results of Unconditioned Model

The Turing-like test was taken 121 times in total. Figure 6.4 shows the average score for the generated and non-generated headlines in each smaller test. Test number 20 had no participants and is therefore not included in the figure. The total average score for the whole test, i.e. all 19 smaller tests that were participated in, was 3.64 for the generated and 2.70 for the non-generated. The lowest average score for a single generated headline was 1.20 and the highest average score was 5.00. For a more thorough analysis, see the Discussion section.

6.2 The Conditioned Model

The conditioned model was trained using two different sets of hyper parameters, effectively creating two models. Each model was trained for about 11,000 steps with the parameters shown in Table 6.3. Both models had the same amount of cells, but arranged in different number of layers.

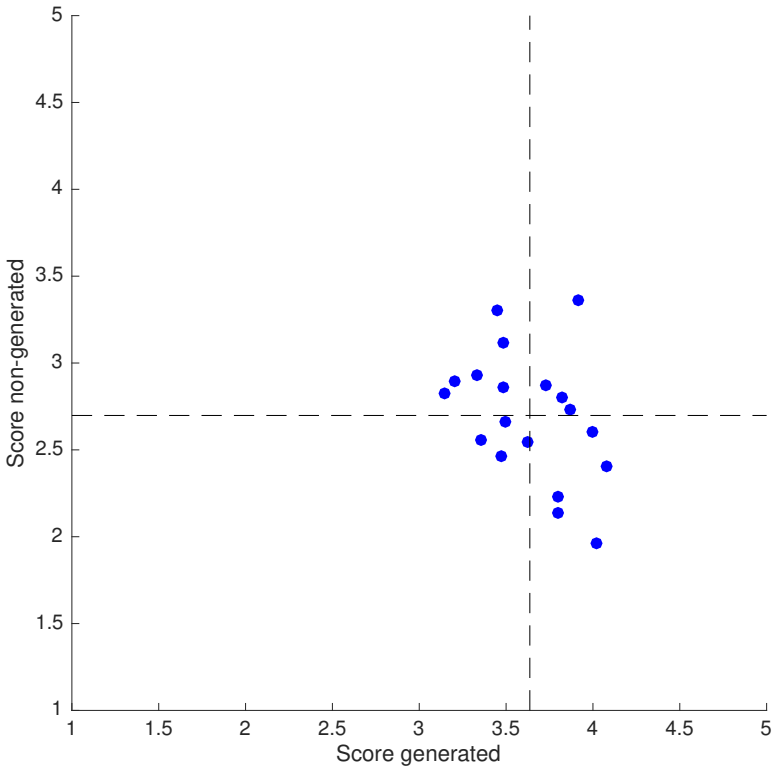

Figure 6.4: Average score for generated and non-generated headlines in each test, indicated by the dots. Total average score for generated and non-generated headlines are indicated by the vertical and horizontal dashed lines respectively. This shows how the average score for generated headlines is higher than for non-generated, which is undesired when trying to generating plausible headlines.

Since the models were run with different batch sizes, they were trained on 352,000 and 176,000 article-title-pars respectively, or 70% and 35% of the training set. The perplexity as a function of training set completion is shown in Figure 6.5

6.2.1 Evaluation on Different Checkpoints

Following are some excerpts from the generated headlines at different checkpoints (steps). An ellipsis, '...', has been inserted where the generated sequence consisted of repeated words. Each line starts with the amount of steps, the perplexity in parenthesis, followed by the generated title.
The articles have been picked as to show (by inspection) results that reflect the networks performance when it performs average, above average, and poorly.
Note that the shallow model has a batch size double that of the deep one (table 6.3), meaning that for the same amount of steps, it has read twice as much data from the training set. In addition, the lowest perplexity reached with deep model (step 11,000), was reached by the shallow one at step 2000.

Table 6.3: Parameters for the second model

Model	Layers	Layer size	Batch size	Article vocabulary	Title vocabulary
Shallow	2	1024	32	40,000 tokens	40,000 tokens
Deep	4	512	16	40,000 tokens	40,000 tokens

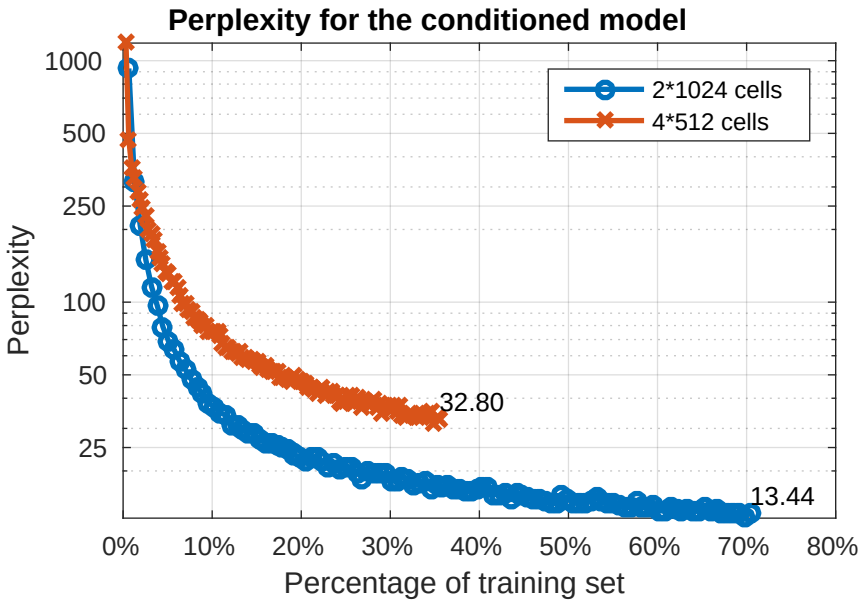

Figure 6.5: The perplexity for the conditioned model on two different sets of parameters

Article 1

The article has the following title: "chinese hackers have been hacking the new york times for the past 4 months"

Results for the shallow model

```
1000  (  232): china : china ' s china ' s china of the hackers
2000  (  347): hackers hit hackers
3000  (17912): traditions traditions ...
4000  (  169): hackers hackers hackers say hackers hackers ...
5000  (  357): hackers hackers breached
6000  (  864): hackers targeted cyber attacks
7000  (  335): chinese hackers hacking journal journal ...
8000  (21363): wines wines wines wines wines muniz muniz ...
9000  (10411): mohave mohave mohave mohave mohave einhorn einhorn ...
10000 (  454): chinese attacks hack hack ...
11000 (  246): chinese security attacks
```

the shallow model has clearly identified some keywords. However, it is not clear that further training improves the results, as checkpoints 3000, 8000 and 9000 show no correlation to the title or article (none of the words in those headlines occur in

the article).

Results for the deep model

```
2000  (662): us : us _UNK
5000  (384): chinese chinese chinese chinese cyber
8000  (397): china report : china report
11000 (307): chinese chinese hacking
```

The deep model did not perform as well as the shallow one. The word ”china” is found in some titles, but many more titles are unrelated to the article. (Steps that showed no correlation to the article or title are removed)

Article 2

The article has the following title: ”streisand to receive lincoln center award”
Results for the shallow model

```
1000  (   16): jj tribute to receive award
2000  (   14): receive award to receive award award
3000  (10835): junkie junkie ...
4000  (    3): award award to center center ...
5000  (    3): streisand to award award ...
6000  (    2): streisand to receive award award ...
7000  (    1): streisand to receive lincoln award prize prize prize
8000  ( 2604): appropriate appropriate ...
9000  ( 4071): all-white all-white ...
10000 (    7): broadway to perform
11000 (    2): streisand to receive center award
```

Again, the shallow model yields results with more correlation to the title, steps 7000 and 11,000 showing a difference of only one word (not counting repeated words). However, sequences that show no correlation are present at other steps.
Note that the perplexity for step 7000 is extremely low, even though (or rather because) words are repeated.
Results for the deep model

```
5000  (10): oscar to oscar oscar
8000  ( 9): local to receive _UNK at 0000 awards
11000 ( 6): lincoln to perform 00th lincoln awards
```

The deep model had only three checkpoints which showed any correlation, the sequences that were removed had perplexities between 2000 and 7000.

Article 3

The article has the following title: ”3-d color x-ray imaging radically improved for identifying contraband, corrosion or cancer”
Results for the shallow model

```
1000  ( 676): the _UNK , the _UNK
2000  (1054): the _UNK
3000  (3415): dawnguard dawnguard ...
4000  ( 747): the _UNK of the _UNK
5000  ( 759): the _UNK
6000  ( 347): the _UNK : _UNK
7000  ( 391): _UNK security : _UNK could be _UNK
8000  (1555): outpatient cupcakes cupcakes ...
9000  (2108): deliveries deliveries ... partied partied ...
10000 ( 232): _UNK : _UNK
11000 ( 303): _UNK : the _UNK of the _UNK
```

Results for the deep model

```
1000  (2717): accounts shareholder shareholder ...
2000  ( 707): _UNK to the _UNK
3000  (2624): buoyed 0-magnitude 0-magnitude ...
4000  (1573): long-shot apologized apologized ...
5000  ( 737): the _UNK : the _UNK of the _UNK
6000  (3098): showtime/cbs showtime/cbs coachella coachella ...
7000  (2045): mater mater ...
8000  ( 534): _UNK : the _UNK of the cold
9000  (2098): misappropriated misappropriated ...
10000 (2561): dogs dogs ...
11000 ( 340): _UNK _UNK : _UNK _UNK
```

Here, the network performed poorly, showing no correlation with the actual title. Note that the results with the lowest perplexity contains a lot of unknown words, indicating that a lot of words in the article were not found in the vocabulary.

6.2.2 Qualitative Study of Dropcost

The generated headlines can be inspected in appendix B. While it's arguable clear that the network with dropcost factor 0.30 performed worse than the two other networks, it is hard to judge which of the those two networks had the best performance. The network without dropcost performed better on articles 3 and 9 and the network with dropcost factor 0.15 performed better on article 2,6 and 7. The performance for remaining 5 articles was generally poor and was counted as draws.

7
Discussion

This chapter discusses the implementation, methods and results used and obtained in the project. It also highlights some difficulties that were encountered, as well as views on the possible impact the techniques will have on society.

7.1 The Unconditioned Model

It is hard to evaluate the results from the unconditioned language model since no one has done the exact same thing as we have done.We do know, however, that models almost identical to ours can achieve a perplexity of about 80 on more coherent pieces of text with smaller vocabulary size [35]. When trained on a series of related sentences, the model probably has more relevant context when choosing its words; hence the lower perplexity compared to the results in Table 6.2. A smaller vocabulary size also has this effect since the model has fewer words to choose from.

The unconditioned model had to be relatively small because of the limited video memory available (see *Workstation* in Table 5.1). It can be discussed if better results could have been achieved by increasing the network's size. From the results, no real conclusions can be said about this, but more parameters would probably be useful if the data set was bigger [36]. Tuning all of the other hyperparameters more carefully could also give some potential gains. However, what we noticed during development was that the easiest way of getting better results was to increase the amount of training data.

Lastly, some words about the word embedding. Before the training sessions started we initialized the embedding matrix with pretrained word2vec vectors [1]. As can be seen in Figure 6.3, this gave us almost no difference when training, except slightly better results in the beginning. This was quite disappointing since we thought that the opimization process would be sped up significantly by using these pretrained vectors. One big reason for this is that not all the words had pretrained equivalents. When this was the case, they would be initialized with the random initializer. Perhaps we could improve the way we tokenize our training data, or let the word vectors train in a separate model before transferring them to the final one. Previous experiments indicate that this can improve the results [37]. Most likely, there was nothing wrong with the word2vec vectors used. These were trained in a similar way and on a corpus also containing news articles.

7.2 The Conditioned Model

Abstract summarization of text is still an active research area, and previous attempts using similar methods have struggled with sequences longer then a couple of sentences, despite having access to both larger data-sets and faster computers [27]. Given how colorful English headlines are, it is not surprising that our implementations struggled with the task.

The quality of the generated headlines varies greatly, and the perplexities of the sentences are not always indicative of the quality. One contributing factor to this is that the cost function (from which the perplexity is calculated), is "rewarding" assignment of high probabilities of even repeated relevant words.
A possible solution to this problem would be to reduce the probability for words that already appear in the sequence, possibly by a weighted factor inversely proportional to the word's frequency in the vocabularies; thus allowing common words such as "to", "in" and "the" to appear multiple times without significant penalties. In general models might benefit from having a more exact concept of of the generated sequence.

Some practical aspects could also have been improved. The most critical resource for the second model was RAM, limiting both the size and amount of layers. More importantly, the vocabularies had to be limited to 40,000 tokens. As a consequence of this, the tokens fed to the network consisted of 29 and 27 percent unknowns for titles and articles respectively. More available memory would allow larger vocabularies to be used, reducing the amount of unknown tokens fed through the network.

The usage of model-specific library-functions when implementing the models reduced the amount of work required by us, but also reduces the amount of control over the code. If the code had been written using only low-level library-functions, it is possible that that further optimizations could have been made. However, such an approach might not have been feasible due to the increased amount of work required.

There are still many modification that might improve performance that we have not tested. As well as using pretrained word representations the weights of the decoding RNN not dealing with the context c_t could be initialized from a pretrained language model, this might help with giving the sematic and gramtic structure a more decisive role.
Further, inspiration could be taken from the recent development of Dynamic Memory Networks (DMNs) introduced by [38] whose main difference to RNNsearch and RNNsearch* is the employment of iteration when calculating alignments and the possibility to update annotations.

Also, while we had trouble implementing RNNsearch*, we are still optimistic that using attention over sentence or phrase representations could improve performance since it would give the model more data and perhaps make it easier to pick relevant alignments.

7.2.1 Dropcost and Negative Log Likelihood

It's commonplace in NLP to model a probability distribution and minimize the negative log likelihood of the training set. This works well with translation [25] [24] and language modeling [36].

In the context of text summarization on the other hand, the target isn't as clear. Viewing the possible continuations of the summary as a tree, many paths may be just as likely. Thus, it might be beneficial for a summarization model to aim mostly at keywords visited by most summaries. These keywords seem easy to pick up from the input sequence and serve as safe bets. Since knowledge of grammar and semantics is deliberately learned and not programmed no explicit penalties are given for this behavior even though it gives unwanted results.

To avoid that the model only picks these safe bet words, we proposed the objective modification technique dropcost. Unfortunately, our results after doing so are unclear and further investigation is needed before any real conclusions on its impact can be drawn.

7.3 Quality of the Data Set

The data used in the project might have needed more preprocessing. Some sentences just contained a few words or were concatenated with punctuation marks, making them far too long. Other peculiarities also existed (like different kind of brackets and symbols), that gave the headlines the possibility to take quite complex forms. Perhaps certain symbols like punctuation marks could have been replaced with newline, creating shorter and more concise word sequences. We could also have sorted out the headlines with lengths above or below some thresholds. "Dumbing" down the data would have made it easier for the model to learn. Further examples would be to replace not just single digits with a certain token, but numbers on the whole. Lastly, more data to train on would have allowed for better results as well as the possibility to learn more complex patterns.

7.4 Analysis of the Turing-like Test and the Results

The Turing-like test was constructed in order to easily establish a measure of the plausibility of the generated headlines. As the measurement should be made on the model's ability to generate common news headlines, rather than on some hand picked set of headlines, as many headlines as possible would have to be evaluated. With the use of social media, it was easy to distribute the test. We did not expect more than 100 participants, as those who carry out surveys often attract participants with some sort of reward, and we did not. It was therefore surprising that the test was taken 121 times in total. The test was open for anyone to participate in and an unrestricted amount of times. This probably helped increase the number of

participants, as they did not have to sign in to identify themselves. It was a strategic move to increase the number of participants, as smaller tests were considered more likely to be completed. Since the presented test was randomly chosen out of 20 tests and no correct answers were presented in the end, one could participate many times without learning the solutions, and thereby contribute to the statistics even more. Test number 20 was not taken by anyone, which seems unlikely to happen, as the average number of participants per test was approximately six. One explanation to this could be that there was a problem with the link to the test or the randomization of the links, but no such problem was found.

Moreover, no control surveys were carried out. This means that the scores could be biased by the fact that the participants knew that some headlines were generated and others not. This does not seem to have affected the results much, since all of the smaller tests participated in had greater average score on the generated headlines than on the non-generated ones. This in turn means that the participants in general could differentiate between the two categories. As the average score for generated headlines lies closer to 5 than to 1, it could be said that the generated headlines do not fool humans in general. However, as both the generated and non-generated headlines had average scores closer to 3 than to any of the extreme ends (1 and 5), the generated headlines seem to fool humans to some extent. The non-generated headlines appear to sometimes be distinguished as generated headlines as well. This means that either the non-generated headlines are hard to interpret or that it is not trivial to differentiate between the two categories.

The average score varied greatly between the smaller tests. For example, test number 10 had an average score of 4.02 for the generated headlines and 1.96 for the non-generated, while test 13 had an average score of 3.15 for the generated and 2.83 for the non-generated. Looking at the headlines in test 10, it can be seen that most of the generated ones do not make much sense. One example is:

```
senior bowl takes off so far tv lucky to be (video)
```

The non-generated headlines on the contrary do make sense. An example of this is:

```
women in combat - historic announcement from pentagon
```

However, some generated headlines managed to fool some participants more than others. One example of this is:

```
beyonc silent with fast live... reveals sex with jay-z?
```

The latter headline shows how the model has managed to relate Beyoncé to Jay-Z, which could be the reason the headlines were perceived as totally plausible by some participants, even though the meaning of the headline is quite unclear.

In test 13, the non-generated headlines mainly received higher scores than in test 10, indicating that they were hard to understand. An example of this could be:

```
winds calmer at waialae for sony open
```

It is possible that the majority of the participants did not know that the Sony Open is an existing golf tournament or that Waialae is a place in Hawaii. The scores of the following generated headline contributed to the low average score for the generated headlines in test 13:

```
lifting coalition to middle east peace meets new issue
```

This headline simply seems to be plausible, though its meaning might not be clear.

The generated headline which received the lowest average score (1.2), i.e. was most plausible, was the following.

```
dorner manhunt: body found in burned cabin after shootout
```

This headline did not exist in the training data. However, there exists headlines which probably had a great impact on the model when training. There are 13 headlines in the data set that contain the word ”dorner”:

- police say christopher **dorner** cause of death single gunshot wound to head
- det. jeremiah mackay: proud new father killed in christopher **dorner shootout**
- christopher **dorner** writings: the scariest parts of the alleged cop killer’s ’manifesto’ (photos)
- **dorner**’s lapd is on the way out
- ptsd: a factor in ex-cop christopher **dorner**’s rampage?
- state of the union, christopher **dorner** standoff compete for media prominence
- **dorner manhunt** ends with car chase, gun fight and blaze
- alleged killer **dorner** died from single shot to head
- fugitive ex-lapd cop, christopher **dorner**, believed to be dead after final standoff
- la **manhunt:** suspect christopher **dorner** once lived in enid, oklahoma, returned $8,000 found in road
- officials hope to id charred remains as those of ex-lapd suspect **dorner**
- christopher **dorner** search: police work to confirm if **dorner**’s remains were found **in burned cabin**
- cause of **cabin** fire fuels debate on christopher **dorner** standoff

These headlines clearly show that the model could relate ”dorner” to the words and phrases ”shootout”, ”manhunt” and ”found in burned cabin”. A notable thing is that the generated headline says ”...body found in burned cabin...”, while one of the non-generated headlines says ”...dorner’s remains were found in burned cabin”. This indicates that the model could have learned that ”body” was a possible replacement word for ”remains”. Searching for the words ”body” and ”remains” respectively in the data set, we can find the following headlines (among others).

```
remains found in california cabin after standoff with fugitive-police
report: cops probe body found in burning van, missing man from claymont
```

On the one hand, these headlines seem to be great sources of inspiration for a human to come up with the generated headline. On the other hand, our model is just a

model, and we cannot say with great certainty how it relates different words and phrases.

The two non-generated headlines which received the highest average score (4.33), i.e. was least plausible, was the following.

```
google nexus review: apple (aapl) ipad or nexus 0 tablet: which is the best?
peyton's arm and conservative broncos; catching up with reid; mail
```

The latter headline demonstrates that some non-generated headlines did not follow the style of a non-generated headline expected by the participants. The fact that there exists headlines like this in the training data may have affected how the model generates headlines. On the other hand, it can be seen by looking at the training data that not many headlines have this unusual style. The model is therefore unlikely to have learned this feature in great detail.

The first headline of the two seems to be fairly plausible when compared to the second. Possible explanations to why the average score was so high could be that the participants missed that numbers were replaced by zeros, did not recognize "(aapl)" as the Apple stock symbol or did not understand the meaning of the headline.

7.5 Obstacles

The following subsections describe some obstacles that were encountered during the course of the project.

7.5.1 Slow Workflow

The models consist mainly of large tensors (multi-dimensional arrays) of floating point numbers, representing parameters in the ANNs. The vocabularies are represented by arrays or integer tokens as well, making the text corpus equivalently consist of sequences of arrays or tokens. Since both the models, vocabularies and the corpus require so much RAM, it was infeasible to train the models on the group members' laptop computers. This severely slowed down the process of finding bugs and trying new ideas out. As far as we know, the initialization of a model and a vocabulary were done on a single thread on the CPU, which also took a long time. The usual workflow of making changes in the code, recompiling and then instantly test the new code could not be used very well in practice.

It is however our experience that the switch from standard SGD, which is used by most research papers, to the Adam optimizer actually outweighs that of the switch from CPU to GPU. Due to this it was sometimes more practical to train the larger networks on cheap cloud instances with practically unlimited RAM.

7.5.2 Mathematically Complex Field

As the field of machine learning was new to all of the group members, there were a lot of new concepts to investigate. Most of the articles found were well written, but incorporated ideas that needed great deliberation. Some articles provided an intuitive explanation of concepts, e.g. as in [30], which really helped understanding the idea behind it all. Most group members were not used to reading articles with the level of mathematical explanations and formulas used in most papers relevant to the project. Therefore, it required some work to read through them, but was on the other hand good exercise for further studies.

7.5.3 Expectations on Previous Knowledge

While the API pages for TensorFlow were well structured and documented, it was clear that previous knowledge about machine learning was expected from the reader. This made the process of understanding the purpose of functions harder, as new concepts were introduced without further explanation.

7.6 Experience

The task of generating common news headlines can seem extremely difficult, especially when the headline should match a given article. One might wonder how a computer program could understand the content of the article, determine what parts are important for the headline and then construct one that is both grammatically correct and somewhat plausible. There may be several ways to tackle these problems, but one way is evidently to use the techniques of recurrent neural networks and attention models. By dividing a problem into multiple sub-problems and parts, e.g. defining it as an optimization problem, using vectors to represent words and the softmax function to obtain target probabilities, a seemingly impossible task can become feasible.

An interesting parallel to this notion is that deep artificial neural networks used for e.g. image recognition can be seen as having the possibility to learn more and more abstract and task specific features of the data in subsequent layers [39]. The first layer may work as mere detectors of edges in an image, while layers towards the last layer may detect more complex objects such as animals or buildings. The deep networks could therefore be said to, in a sense, have the possibility to divide a problem into smaller problems by itself, though it can be hard to translate this intuition into the task of generating headlines.

7.7 Impact on Society

At the moment, machine learning seems to be a trending topic, which can be seen using Google Trends. The techniques used in this project are relatively new, as some of the key articles proposing them, such as [25] and [18], were published in

2014. Effective machine translation could open up new possibilities for information exchanges, as it helps removing the barrier between people speaking different languages. The results obtained during this project indicate that machine translation possibly could be used as a tool for abstractive summarization as well, though our implementation would need modifications to work effectively and have an impact on society. As mentioned in chapter 1, abstractive summarization could be beneficial in fields such as medical health care, helping to gather the most relevant information from large documents.

The strength of machine learning is that it can be applied to a multitude of problems. The company DeepMind (whose program AlphaGo became the first to beat a professional level Go player in March 2016) has, together with kidney experts and app designers, constructed a smartphone app to help detecting cases of acute kidney injury. This could be a potential life and money saver [40].

8

Conclusion

A machine learning technique called deep learning has improved the way computers solve tasks like speech and object recognition [6]. Recurrent neural networks are particularly suited to modeling sequences with long-term dependencies [17]. These networks enable sequence generation of natural language, possibly of better quality than using other techniques such as n-gram. With the use of an attention mechanism, it should also be possible to find relevant parts of an article for constructing summary-like headlines.

In this project, we implemented one model for generating random news headlines. The model produced headlines of moderate plausibility and quality. Two other models with attention mechanisms were implemented for generating headlines based on some input text. These models generated nonsense, probably due to overly complex dependencies between article text and headline.

We conclude that deep recurrent neural networks can, given enough time and resources, be used to model natural language such as news headlines.It is likely that the models with attention mechanisms could be refined to generate headlines of better quality. However, it remains unclear to what extent this technology could produce summary-like headlines to an input text.

During the project, insight has been given into the field of machine learning, and deep learning with recurrent neural networks in particular. Even though the outcome of the models were of moderate to poor quality, it has been an insightful journey, which has inspired to further studies in the area.

Bibliography

[1] "word2vec," Feb. 04 2016. [Online]. Available: https://code.google.com/archive/p/word2vec/

[2] J. H. Martin and D. Jurafsky, "Speech and language processing," *International Edition*, 2000.

[3] S. Bird, E. Klein, and E. Loper, *Natural Language Processing with Python*, 2015. [Online]. Available: http://www.nltk.org/book/ch08.html

[4] L. R. Bahl, P. F. Brown, P. V. De Souza, and R. L. Mercer, "A tree-based statistical language model for natural language speech recognition," *Acoustics, Speech and Signal Processing, IEEE Transactions on*, vol. 37, no. 7, pp. 1001–1008, 1989.

[5] E. Charniak, "Statistical parsing with a context-free grammar and word statistics," *AAAI/IAAI*, vol. 2005, no. 598-603, p. 18, 1997.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, "Parsing natural scenes and natural language with recursive neural networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 129–136.

[8] B. A. Hockey and M. Rayner, "Comparison of grammar-based and statistical language models trained on the same data," in *Proceedings of the AAAI Workshop on Spoken Language Understanding*. Citeseer, 2005.

[9] M. Kågebäck, O. Mogren, N. Tahmasebi, and D. Dubhashi, "Extractive summarization using continuous vector space models," in *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)@ EACL*, 2014, pp. 31–39.

[10] P. F. Brown, P. V. Desouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai, "Class-based n-gram models of natural language," *Computational linguistics*, vol. 18, no. 4, pp. 467–479, 1992.

[11] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, "On using very large target vocabulary for neural machine translation," *CoRR*, vol. abs/1412.2007, 2014. [Online]. Available: http://arxiv.org/abs/1412.2007

[12] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015. [Online]. Available: http://neuralnetworksanddeeplearning.com/index.html

[13] L. Bottou, *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*. Heidelberg: Physica-Verlag HD, 2010, ch. Large-Scale Machine Learning with Stochastic Gradient Descent, pp. 177–186. [Online]. Available: http://dx.doi.org/10.1007/978-3-7908-2604-3_16

[14] O. Delalleau and Y. Bengio, "Parallel stochastic gradient descent," Aug. 11 2007. [Online]. Available: http://www.cs.toronto.edu/~amnih/cifar/talks/delalleau_talk.pdf

[15] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[16] M. Kågebäck, "Three layer feed-forward neural network," Feb. 15 2016, modified with permission from the author. [Online]. Available: http://www.cse.chalmers.se/research/lab/mlcourse/hw3/hwnn.pdf

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

[18] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[19] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[20] Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain, "Neural probabilistic language models," in *Innovations in Machine Learning*. Springer, 2006, pp. 137–186.

[21] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[22] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[23] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," *The Journal of Machine Learning Research*, vol. 12, pp. 2493–2537, 2011.

[24] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[25] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[26] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *CoRR*, vol. abs/1409.1259, 2014. [Online]. Available: http://arxiv.org/abs/1409.1259

[27] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," *arXiv preprint arXiv:1509.00685*, 2015.

[28] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents," *CoRR*, vol. abs/1405.4053, 2014. [Online]. Available: http://arxiv.org/abs/1405.4053

[29] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," Sep. 16 2012. [Online]. Available: http://arxiv.org/pdf/1206.5533v2.pdf

[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[31] M. Mikael Kågebäck and H. Salomonsson, "Word sense disambiguation using a bidirectional lstm," *Lexical and Computational Semantics (* SEM 2016)*, 2016.

[32] C. Zhang and D. S. Weld, "Harvesting parallel news streams to generate paraphrases of event relations." in *EMNLP*, 2013, pp. 1776–1786.

[33] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," Feb. 04 2016, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[34] A. M. TURING, "I.—computing machinery and intelligence," *Mind*, vol. LIX, no. 236, pp. 433–460, 1950. [Online]. Available: http://mind.oxfordjournals.org/content/LIX/236/433.short

[35] G. Inc. (2015) Example/benchmark for building a ptb lstm model. Accessed: 2016-04-25. [Online]. Available: https://github.com/tensorflow/tensorflow/blob/master/tensorflow/models/rnn/ptb/ptb_word_lm.py

[36] T. Mikolov, "Statistical language models based on neural networks," *Presentation at Google, Mountain View, 2nd April*, 2012.

[37] L. Eidnes. (2015) Auto-generating clickbait with recurrent neural networks. Accessed: 2016-04-26. [Online]. Available: https://larseidnes.com/2015/10/13/auto-generating-clickbait-with-recurrent-neural-networks/

[38] A. Kumar, O. Irsoy, J. Su, J. Bradbury, R. English, B. Pierce, P. Ondruska, I. Gulrajani, and R. Socher, "Ask me anything: Dynamic memory networks for natural language processing," *arXiv preprint arXiv:1506.07285*, 2015.

[39] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *CoRR*, vol. abs/1411.1792, 2014. [Online]. Available: http://arxiv.org/abs/1411.1792

[40] "Deepmind," Apr. 05 2016. [Online]. Available: https://deepmind.com/health.html

A

Articles

Listed below are the articles used in Section 6.2. Since the model only reads the first 200 words, the articles have been truncated.

A.1 Article 1

share discuss bookmark because the ny times recently exposed chinas prime minister, wen jiabao, for having made billions of dollars through business dealings, chinese hackers have been trying to hack and infiltrate the ny times for the past 4 months. security experts say the hackers used methods consistent with the chinese military. according to security firm mandiant (which was hired by the times), the chinese hackers tried to hide their identity by routing their attacks on the ny times through universities in the united states. previous hacking attacks have used this method and many of those attacks originated from china. the ny times reports: the attackers first installed malware malicious software that enabled them to gain entry to any computer on the timess network. the malware was identified by computer security experts as a specific strain associated with computer attacks originating in china. more evidence of the source, experts said, is that the attacks started from the same university computers used by the chinese military to attack united states military contractors in the past. no customer data has been stolen and from what they can tell, there has been no evidence that sensitive files about the wen family gathered through reporting have been obtained either. china has a history of hacking american companies, sometimes causing them to get hacked right back. many of its hacking initiatives are done in order to control chinas public image, along with stealing secrets. which means this aint the first time and it wont be the last. [ny times] share

A.2 Article 2

streisand to receive lincoln center award reuters january20,2013,12:04amtwn share tweet new york–barbra streisand will add the film society of lincoln center's chaplin award to her roster of honors, in recognition of her achievement as a director, writer, producer and film star, the group said on friday. streisand, who shot to fame in the 1960s on broadway and as a major recording star, will receive the honor at the 40th annual chaplin award gala in new york on april 22 that will feature celebrity guests and a host of film and interview clips. the board is very excited to have barbra streisand as the next recipient of the chaplin award, ann tenenbaum, the film society of lincoln center's board chairman, said in a news release. she is an artist whose long career of incomparable achievements is most powerfully expressed by the fact that her acclaimed 'yentl' was such a milestone film. the group cited streisand as the first american woman artist to receive credit as writer, director, producer and star of a major feature film. it also noted she is the only artist to receive an academy award, tony, emmy, grammy, directors guild of america award, golden globe, national medal of arts and peabody awards, france's legion d'honneur and the american film institute's lifetime achievement award. she was also the first female film director to receive a kennedy center honor. we welcome her to the list of masterful directors who have been prior recipients of the chaplin award tribute, added tenenbaum, referring to luminaries such as alfred hitchcock, billy wilder and martin scorsese.. . .

A.3 Article 3

electron microscope its ability to identify the composition of the scanned object could radically improve security screening at airports, medical imaging, aircraft maintenance, industrial inspection and geophysical exploration. the x-ray system developed by professor robert cernik and colleagues from the school of materials can identify chemicals and compounds such as cocaine, semtex, precious metals or radioactive materials even when they're contained inside a relatively large object like a suitcase. the method could also be extended to detect strain in fabricated components, for example in aircraft wings, and it can be used to image corrosion processes and chemical changes. in healthcare, the system can be used to detect abnormal tissue types from biopsy samples. in geophysical exploration it could be used to quickly analyse the content of core samples taken from bore holes. in a recent experiment the team used the technology to x-ray a usb dongle that controls webcams. they were able to identify the different elements and components inside the dongle by analysing the energy sensitive radiographs and fluorescence patterns. . . .

B

Dropcost data

Listed below are the selected headlines from the qualitative study on the effects of dropcost on the conditioned language model.

1.

Dropcost	Perplexity	chinese hackers have been hacking the new york times for the past 4 months
0	454	chinese attacks hack hack hack hack hack hack hack
0.15	626	chinese chinese attacks : chinese chinese attacks
0.30	191	chinese hackers hack hack hack hack hack hack hack hack hack hack

2.

Dropcost	Perplexity	report: algeria hostage crisis ends; death toll unclear
0	51	algerian hostages : algerian hostages , algeria hostages , algeria hostages , algeria hostages
0.15	245	algeria bodies : death toll of algeria
0.30	291	algeria hostage crisis : algeria crisis says army ' s death

3.

Dropcost	Perplexity	4 gadgets that defined las vegas ces 2013
0	3	0 gadgets that defined ces ces 0000 ces ces
0.15	8	0 ces 0000 : ces vegas vegas vegas vegas vegas vegas vegas vegas vegas vegas vegas vegas
0.30	14	0 vegas xperia z ultra vegas 0000

4.

Dropcost	Perplexity	anderson cooper and aretha franklin defend beyonce's lip-sync at obama inauguration
0	331	inauguration obama ' s inauguration performance
0.15	254	beyonc lip lip lip lip synced at the inauguration
0.30	60	beyonce ' s lip-syncing : beyonce lip-synched

5.

Dropcost	Perplexity	iran cyberattacked us banks according to government officials
0	624	us attacks : banks , banks
0.15	121	banks attack
0.30	337	the banks : the banks

6.

Dropcost	Perplexity	kim kardashian shares baby bump picture: "popped outta nowhere!"
0	160	kim kardashian bump shows baby bump
0.15	92	kim kardashian unveils baby !
0.30	1651	kim kardashian ! kim kardashian ! kim kardashian ! kim kardashian ' s baby baby baby baby baby baby

7.

Dropcost	Perplexity	north korea nuclear test expected
0	22	north korea calls for nuclear test
0.15	7	north korea threatens nuclear test
0.30	2	north korea nuclear test test test

8.

Dropcost	Perplexity	fla. man charged after allegedly driving 18-wheeler into van's path, killing 1
0	686	man arrested for fatal suspect
0.15	491	man charged with ex-wife
0.30	366	judge county convicted death

9.

Dropcost	Perplexity	super bowl 2013 live updates, score: 49ers, ravens face off in new orleans
0	52	super bowl 0000 : live stream , ravens off off in new orleans
0.15	21	super bowl 0000 : 00ers , score score score score score score in new orleans , 00ers , 00ers , 00ers , ...
0.30	78	nfl ravens live : ravens , ravens in super bowl xlvii

10.

Dropcost	Perplexity	intel scores 4q beat despite 27% slide in profits
0	246	intel shares fall in q0
0.15	53	intel beats 0q profit
0.30	179	strong 0q 0q 0q 0q year of 0q 0q 0q 0q 0q 0q 0q 0q 0q 0q 0q 0q 0q 0q results