MASTER'S THESIS | LUND UNIVERSITY 2017

News text generation with adversarial deep learning

Filip Månsson, Fredrik Månsson

Department of Computer Science
Faculty of Engineering LTH

ISSN 1650-2884
LU-CS-EX 2017-18

News text generation with adversarial deep learning

Filip Månsson
tfy12fm1@gmail.com

Fredrik Månsson
tfy12fma@gmail.com

September 6, 2017

Master's thesis work carried out at Sony Mobile Communications AB.

Supervisors: Håkan Jonsson, hakan1.jonsson@sonymobile.com
Pierre Nugues, pierre.nugues@cs.lth.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

In this work we carry out a thorough analysis of applying a specific field within machine learning called **generative adversarial networks**, to the art of natural language generation; more specifically we generate news text articles in an automated fashion. To do this, we experimented with a few different architectures and representations of text, evaluated the results and used the information retrieved from the results, to create a model that should give the best result. For evaluation, we used perplexity and human evaluation. We also looked at the token distribution to see which model captures the texts most successfully.

We show that it is possible to use generative adversarial networks to generate sequences of tokens that resemble natural language, but this does not yet reach the quality of human-written text. Further hyperparameter tuning and using a narrower-subjected corpus could improve the output.

Keywords: Machine learning, generative adversarial learning, GAN, natural language generation

Acknowledgements

We would like to thank both of our supervisors for helping us with this project and taking the time and effort to answer our questions as well as providing us with valuable feedback. We also want to send our regards to our parents and siblings for their support throughout our lives.

Contents

Chapter 1
Introduction

This chapter gives a brief overview of the topic at hand and the problem formulation of the thesis as well as related work. We also provide a short description of the contributions.

1.1 Overview

Recently there have been an increasing number of reports about the impact that fake news has on the world, not exclusively constrained to elections but also to other influential areas of the global market such as the stock market (Rapoza, 2017). The source of the fake news can be driven by the purpose of actually affecting an outcome or simply to increase the living standard by making a quick and simple profit of them (Kirby, 2016). What these sources have in common is that there is a human being behind the texts.

Over the past few years machine generated texts become more and more common within all from news articles (Eidnes, 2015) and scientific reports to plays inspired by Shakespeare (Karpathy, 2015). Today there are several different methods and techniques available for producing texts of various types and qualities. As computation power grew with time, so did the complexity of the models capturing the languages. Many of the more successful models are based on deep learning and neural networks (Karpathy, 2015). A more recent model not primarily used for text generation is the **generative adversarial network**, (**GAN**).

The main idea behind GAN (Goodfellow, 2017) is that instead of training one network you train two and you train them against each other as in the mini-max game. One of the players in the game is called the **generator**, whose purpose is to generate samples that resemble those drawn from the training data. The other player is the **discriminator**, responsible for classifying samples as real or generated. The generator is trained to produce samples that deceive the discriminator, while the discriminator is trained on the classified samples as in traditional supervised learning.

1.2 Problem Definition

The goal of this thesis is to implement and make use of GANs for generation of news articles. The primary objective is to make the articles readable with less emphasis on truthfulness. Therein we will also get an understanding of the potential within automatic text generation and possibly also generate a corpus containing fake articles as an aid for creating methods of classification. We will look into how text has been generated previously using machine learning methods and then try to incorporate the learned information into the attempt of utilizing a GAN for the purpose of generating news articles. We will attempt to use a few different architectures, evaluate these, and finally use the insights made to create a better model.

We are specifically going to look at the original GAN and **Wasserstein-GAN** (**WGAN**) for text generation. We will train models with different settings and architectures and evaluate them using perplexity and human evaluation. We will also look at the distributions of tokens within the texts.

1.3 Related Work

The topic of text generation is gaining more attraction and research regarding this area is on the rise. The following text piece discusses research that is related to our work.

Text generation has been done by different approaches such as a sequence to sequence model by Sutskever et al. (2014) where they translated English texts to French by using two **long short term memory** (**LSTM**) networks: one encoder and one decoder. The nature of this model also allowed for a varying input length as the output from the encoder is always mapped to a vector of a fixed size. The result of the implementation showed comparable to a reference **statistical machine translation system** (**SMT**). This good result was partly achieved due to the fact that they were using LSTM cells, cells that are specifically designed to remember long term dependencies. The authors also stated that they achieved a better result by reverting the input to the translator.

When it comes to generating realistic images GANs have proven to be successful. However, regarding the task of generating text or any kind of task which involves text as input or output, the use of GANs has not yet been as satisfactory. Many of the issues concern the representation of text as a continuous space according to Goodfellow (2017).

GANs have only recently been applied to text generation; for instance Li et al. (2017) has applied them to dialogue generation using reinforce algorithms. The authors use a GAN and compares the performance with more common methods of dialogue generation. The paper also introduces some useful tricks such as teacher enforced training to guide the generator towards the right path. This is done during the generation process of the training, by having a "teacher" intervene and force the generator to output the correct response.

A known problem with generating discrete outputs such as words is the inability to back-propagate useful information from the discriminator to the generator. Kusner and Hernández-Lobato (2016) overcome this problem by using the Gumbel-softmax distribution and managed to generate discrete sequences of elements.

Another approach was investigated by Zhang et al. (2016), where they use an LSTM network as generator and an **convolutional neural network** (**CNN**) network as discrimi-

nator. The LSTM network worked as an encoder, mapping from an encoded feature vector (word embeddings) to the succeeding word; in this case the output is in the form of one-hot encoding i.e. a binary vector of the same size as the vocabulary and the only legal combination of values are those with a single one and all others are zeroes. So a word will be represented by a vector of zeros except for at the specific words index, where there is a one. A sentence was constructed by always choosing the word with the highest probability given the previous word/vector and by applying an `argmax` function to the output of the generator. This was repeated until the generator reached the end of a sentence (a special token). The discriminator is pretrained by being fed real sentences and sentences with swapped word order. By doing so it should learn the structure of sentences. The problem with discrete outputs was then solved using an approximated discretization by using a `softmax` function followed by an `argmax` operation. By feeding generated sentences and real sentences to the discriminator, they carried out training in an adversarial way, where a form of feature matching was used instead of the original objective function proposed by Goodfellow et al. (2014).

1.4 Contributions

We hope the results produced by this thesis will help Sony Mobile Communications in the evaluation of the potential of GANs. The goal is to contribute to the research on how to represent words and how to use GANs to generate text. The thesis will hopefully also aid with the creation of a corpus useful when investigating methods for detection of fake news and thereby prevent some spread of misinformation.

From what we can see we are the first to have our model output embeddings directly instead of using a one-hot encoding. We are also among the first to use the WGAN structure for generating text.

Chapter 2

Background

This chapter covers the background of text generation and more about neural networks as well as deep learning. More information is also provided concerning generative adversarial networks of different flavors.

2.1 Text Generation

Natural language generation (NLG) is the science of "linguistic manipulating of data" and "NLG manipulates (linguistically) deeper information to produce shallower information" (Evans et al., 2002). As a result of this definition, it is not enough to simply parse the data or keep the same level of information (as for instance creating a summary or translating a text), but by using the data to produce new information.

There exists several techniques for NLG including rule-based, statistical and data-driven. Rule-based models rely on hand-written specialized rules for generation. Included in the rule-based models are also template-based where the models make use of predefined phrases or grammar. The problem with these are that they often require many rules to be crafted and the resulting model will often be restricted to a small specialized domain as well as a small vocabulary. This in turn means that the model may perform very well but it lacks flexibility (Manishina et al., 2016).

Statistical models (usually some combination of N-gram models) are based on counts retrieved from the training data. The models allows for explicit modeling of the context and joint probabilities. The issue with these models arise because they are limited to what they see and their assumption of independence. This results in that these models can't capture any long term dependencies (not longer than N).

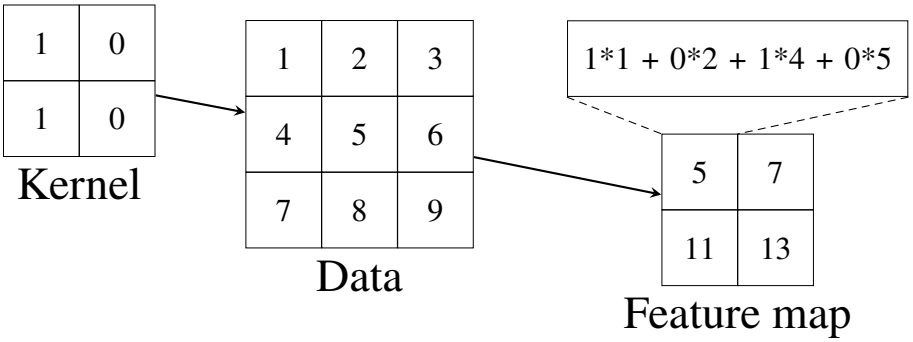

Figure 2.1: Illustrating a simple 2D convolution with a kernel size of 2x2 and a stride of 1. Here we use the "valid" technique where we restrict the output to only consist of points where the kernel completely fits within the input.

2.2 Neural Networks

Deep learning uses neural networks (or artificial neural networks) as building blocks. This name comes from the idea that the models used in deep learning has been engineered to mimic or take inspiration from the brain. In the early beginning of neural networks, as early as the 1940s, they were using simple linear models to classify the inputs as belonging to one of two classes. Today a lot of the trained models rely on algorithms that were developed by the early researchers but could not be applied as the computation costs where too high. Neural nets have also had an recent upswing as it has become easier to find large data sets, which in turn means that it requires less tuning of the nets, (Goodfellow et al., 2016, p. 13-20).

2.2.1 Convolutional neural networks

Convolutional neural networks (CNN), are a type of network that is applied on matrix or grid-like data. The first word in the name, i.e. convolution, reveals that the main operation in this type of network is the convolution operation. You can think of a convolutional operation as taking the weighted average of several data points. When dealing with chunks of data in matrix form this can be seen as the dot product. You place the aforementioned weights in a matrix called the kernel of the wanted size. Usually the kernel is much smaller than the matrix containing the data and therefore you will need to stride, meaning that you slide the kernel over the data matrix. The final result or the output of the convolutional operation is called a feature map. See Figure 2.1 for a visual demonstration of a convolution, (Goodfellow et al., 2016, p. 330-334).

2.2.2 Recurrent neural networks

Recurrent neural networks (RNN) is another flavor of neural networks. Unlike the CNN that specialized in processing grid-like data structures the RNN model was created to manage sequential data. Recurrent neural networks make use of parameter sharing across the

model layers. The sharing of parameters makes it possible to generalize and use the same weights over different positions in time. This means that it doesn't matter whether for instance a word is at the first position of the sentence or in the last, the model will treat it the same way, (Goodfellow et al., 2016, p. 373-374).

2.2.3 Long short term memory

Normally RNNs are capable of using context information i.e. predicting the next word given the previous, but this becomes harder as the distance between the dependencies within the sequences grows. For instance given the sequence "My name" even the standard RNN model should be able to simple predict that the next word is "is". Consider the sequence "Last summer I went to Denmark for vacation. It was ... I have decided that next year I will return to" we want the model to predict "Denmark" but it is not easy for the model to remember the dependency. Long short term memory network (hereon referred to as LSTM) is a variant of RNN developed to solve this issue. The key concept to LSTMs is the cell state, a memory keeping track of vital information that the cell has seen. This state can for instance contain information about the subject of the sentence as is it a thing or a person and based on that use the correct pronoun. In each interaction with a LSTM cell, the internal workings decides what to do with the state, whether to remove or add information to it, (Olah, 2015).

2.2.4 Residual learning

A more recent technique introduced into the deep learning community is the use of residual learning (goes also under the name residual block or resblock). The main purpose behind residual learning is to remove the degradation of training accuracy introduced by the depth of a model. The solution is to use a shortcut from the input of the block to the output. This has shown great empirical results, (He et al., 2015).

2.3 Generative Adversarial Networks

Generative adversarial networks (GANs) were first introduced by Goodfellow et al. (2014) and have become very popular. Despite its rather new entry to the family of machine learning techniques, it has shown great results when applied to image synthesizing.

Generative adversarial networks belong to the family of generative models. Given a set of training data from a distribution, GAN models will learn a distribution representing an estimate of the original data distribution. The model can either represent an actual distribution or it can generate samples from one. Often it is the latter, generation of samples, which is more common.

Training models often mean that you need to have a data set with labels for all data. However generative models can be trained with a data set containing a mix of both labeled and unlabeled data, removing an otherwise limitation caused by some lack of information (Goodfellow, 2017).

Conventional training in machine learning is based on minimizing the differences, usually MSE (Mean squared error), between the output and the target. This is clearly not pos-

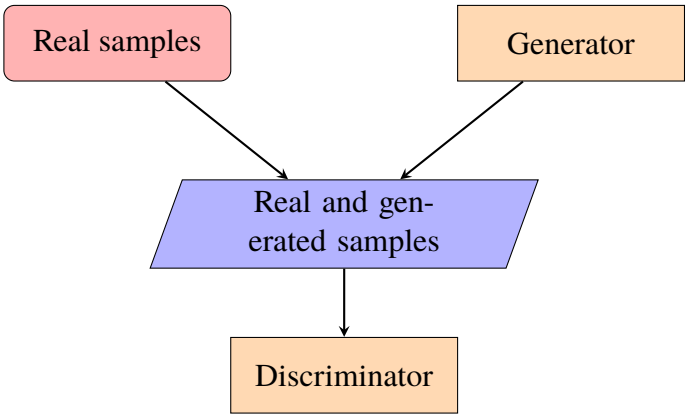

Figure 2.2: Illustrating the overall structure of the GAN

sible when there might be multiple targets which brings us to another advantage of GANs: the property to cope with outputs which are multi-modal, i.e. each input may correspond to multiple outputs, which are all correct. This was illustrated in Lotter et al. (2015) where the task was to predict the next frame given previous frames in a video. In the video, a human looking face was rotating at random speed. There are therefore multiple frames that all could be correct. By using the approach of GANs, the predicted frame was clearer suggesting it had chosen a single frame and not an average of frames as was the case when using a MSE loss.

If we instead look at what kind of data text is, how could this multimodal property show? We know for instance that there are multiple ways of describing an object. Consider for instance the task of describing a car. The car has four wheels, a steering wheel, windows and so on. But it also has a color. We can change the description of the car by giving it another color. The descriptions are no longer the same since the colors are different but they are all correct since they all describe the same object, a car.

The ability to generate samples originating from a distribution means that we do not actually need to explicitly learn a complete distribution. This is especially important when the distribution is high dimensional or for some other reason hard to represent or learn. Representing all possible articles would require a hyperspace of infinite dimension. Representing all English words or a fraction of them is tractable but then you need to learn what words should be in an article, in what order, how long should the article be and so on in order to generate new ones.

Generative adversarial networks consist of two models, a generator and a discriminator. By having samples of data drawn from a distribution representing our target distribution, the generator's purpose is then to use noise as input, often a uniform distribution, and generate samples that look like they were drawn from the same distribution as the data samples. The discriminator is fed with data samples from the "real" distribution and samples that are fake, generated by the generator, and is then asked to predict which ones are real and which ones are fake. The output from the discriminator is then propagated to the generator so as to update, improve and generate more realistic samples. You can think of it this way: the generator acts as a criminal producing counterfeit money and the discriminator as the police trying to discriminate fake money from real. For an overview of the network see Figure 2.2.

2.3.1 Generator

The generator is defined as a function G that is differentiable with respect to its parameters θ_g and input $\mathbf{z}$. $\mathbf{z}$ is to be considered as noise drawn from a distribution P_z, e.g. some Gaussian (normal) distribution. The generator will then map $\mathbf{z}$ to samples $G(\mathbf{z})$ corresponding to a sample drawn from the generator's distribution P_g which will hopefully after training be the same as the true data distribution P_{data}. The mapped samples $G(\mathbf{z})$ are fed to the discriminator and the generator is then trained to fool the discriminator by having it give high probabilities to the generated samples.

2.3.2 Discriminator

The discriminator D is just like the generator differentiable with respect to its parameters θ_d and input $\mathbf{x}$ and $G(\mathbf{z})$, where $\mathbf{x}$ is real data samples drawn from the P_{data} distribution. The discriminator then outputs a value representing the probability of the sample belonging to the real data distribution P_{data}. The discriminator will then be trained, in a supervised setting, to give a high probability to real data samples and a low to samples generated by the generator.

2.3.3 Cost function

In a more mathematical perspective, the generator and discriminator cost functions are related by Equation 2.1.

$$\min_G \max_D \left(\mathbb{E}_{\mathbf{x} \backsim P_{data}(x)}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \backsim P_z(z)}[\log(1 - D(G(\mathbf{z})))] \right) \tag{2.1}$$

Where $\mathbb{E}_{\mathbf{x} \backsim P_{data}(x)}$ and $\mathbb{E}_{\mathbf{z} \backsim P_z(z)}$ are the expected values of the P_{data} and P_z distributions. Optimizing Equation 2.1 with respect to the discriminator will result in minimizing the Jensen-Shannon divergence between P_{data} and the P_g distribution, see Equation 2.2.

$$JS(P_{data}, P_g) = D_{KL}\left(P_{data} \middle|\middle| \frac{P_{data} + P_g}{2}\right) + D_{KL}\left(P_g \middle|\middle| \frac{P_{data} + P_g}{2}\right) \tag{2.2}$$

where D_{KL} is the Kullback-Leibler divergence:

$$D_{KL}(P_{data}(x) || P_g(x)) = \int_{-\infty}^{\infty} \frac{P_{data}(x)}{P_g(x)} P_{data}(x)\, dx$$

Since the generator's cost function is dependent on the discriminator's parameters θ_d, and the discriminator's cost function is dependent on the generator's parameters θ_g, the solution will also be equivalent to a Nash equilibrium which will be where the generator's cost function is minimized with respect to θ_g and the discriminator's cost function is minimized with respect to θ_d, (Goodfellow, 2017).

2.3.4 Algorithm

Algorithm 1 describes how the training procedure is carried out in generative adversarial networks.

Algorithm 1 Implementation of GAN

procedure GAN
  for number of iterations **do**
    for number of steps to run D **do**
      $\mathbf{z} \leftarrow$ *minibatch of m samples from* $P_z(z)$.
      $\mathbf{x} \leftarrow$ *minibatch of m samples from* $P_{data}(x)$.
      Update D by maximizing:
      $D \leftarrow \nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} [\log D(\mathbf{x}^{(i)})] + [\log(1 - D(G(\mathbf{z}^{(i)})))]$
    $\mathbf{z} \leftarrow$ *minibatch of m samples from* $P_z(z)$.
    Update G by minimizing:
    $G \leftarrow \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} [\log(1 - D(G(\mathbf{z}^{(i)})))]$

2.3.5 Known issues

We have already mentioned a vital problem with GANs and that is their inability to generate discrete outputs such as words. The reason behind this is that the generator needs to be differentiable and thus can only have continuous data as input (Goodfellow et al., 2014).

When convergence is reached the generator should produce samples similar to those drawn from P_{data}. However, GANs consist of two networks trained by maximizing and minimizing the cost function depicted in Equation 2.1. This is unfortunate since updating one of the networks may be the same as moving the other network in the opposite direction, away from convergence and in worst case the Nash equilibrium will never be reached. The two networks will then get stuck and the outputs only oscillate between the same points.

One of the problems originating from non-convergence is **mode collapse** or sometimes referred as the **helvetica scenario**. Mode collapse occurs when the generator maps different samples from P_z to the same point x. As mentioned by Goodfellow et al. (2014) the generator must not be trained too much, effectively overpowering the discriminator, since an optimal generator is the Dirac delta function $\delta(x)$ where x are points the discriminator give high probabilities. This will of course result in a reduction of P_g approximation of P_{data}. There are also problems concerning the power or capacity of the discriminator and the generator. If the discriminator is very confident and can sort out real and generated samples with high accuracy, the gradients backpropagated to the generator will vanish, meaning the generator will not improve any further. This scenario may happen any time, especially in the very beginning of training since the generator will not produce any realistic samples at that time, (Goodfellow, 2017).

2.4 Wasserstein-GAN

Since the original paper about GANs was published there has been efforts to improve GANs and remove some of the known problems (mode collapse, stability). Wasserstein-GAN (WGAN) is one of them and was introduced by Arjovsky et al. (2017). They investigated how to best measure the divergence or distance between two distributions since this will largely affect the convergence. If the measure is weak, it will be easier to converge to

the real distribution. The distance measure that was used by Arjovsky et al. (2017) is the Earth-Mover distance or Wasserstein distance. An illustrative way to look at the Earth-Mover distance is to think about it as a measurement of the cost for moving earth from one pile to another (the amount of earth times the distance the earth is moved), hence the name.

After running a set of experiments they concluded that most of the problems such as mode collapse and vanishing gradients never appeared. They also showed properties of the WGAN being more stable and not as dependent of the choice of hyperparameters. In addition the loss of the models related well to the quality of the output.

In order to emphasize what has been improved from the original GAN we will now explain the differences in more detail.

2.4.1 Generator

There is no difference between the generator compared to the original GAN, it still needs to be a function G that is differentiable with respect to its parameters θ_g and input $\mathbf{z}$, where $\mathbf{z}$ is data drawn from distribution P_z. The only difference is the feedback it will get from the discriminator: it will now be a simple difference between the real and generated samples and not a difference between the probabilities. The generator will then fool the discriminator by reducing the difference between real and generated data.

2.4.2 Critic

The purpose of the discriminator will no longer be to discriminate between real and generated samples which is why it is now called critic. The critic will now only output the difference between two samples and will therefore be trained to give a large difference between the two samples. The critic C also needs to be differentiable with respect to its parameters θ_c and input $\mathbf{x}$ and $G(\mathbf{x})$, where $\mathbf{x}$ is drawn from P_{data}.

2.4.3 Cost function

The cost function for WGANs will use the Wasserstein distance instead of Jensen-Shannon as for GANs. The Wasserstein distance which we from now on will call Earth-Mover distance (EM) is given by Equation 2.3.

$$EM(P_{data}, P_g) = \inf_{\gamma \in \prod(P_{data}, P_g)} \mathbb{E}_{(x,y)\backsim\gamma}\big[\|x-y\|\big] \tag{2.3}$$

Where $\prod(P_{data}, P_g)$ is the joint distribution γ, with P_{data} and P_g as marginals.

The EM-distance in this shape will be difficult to calculate (Arjovsky et al., 2017). Arjovsky et al. overcame this by using the Kantorovich-Rubinstein duality which resulted in the cost function given in Equation 2.4.

$$\min_G \max_{C \in L} \Big(\mathbb{E}_{\mathbf{x}\backsim P_{data}(x)}[C(\mathbf{x})] - \mathbb{E}_{\mathbf{z}\backsim P_G(z)}[C(G(\mathbf{z}))] \Big), L \in \text{1-Lipschitz functions} \tag{2.4}$$

Since we are now using the Earth-Mover distance there will be another constraint on the critic in Equation 2.4: the critic also needs to be 1-Lipschitz, i.e. a function that has gradients with a norm equal to one or less, which was solved by clipping the weights of the critic.

Optimizing Equation 2.4 with respect to the critic will result in minimizing the Earth-Mover distance between P_{data} and P_g.

2.4.4 Algorithm

Algorithm 2 describes how the training procedure is carried out in Wasserstein-GAN as implemented by (Arjovsky et al., 2017).

Algorithm 2 Implementation of WGAN, α = learning rate, c = clipping value

procedure WGAN
  for number of iterations **do**
    for number of steps to run C **do**
      $\mathbf{z} \leftarrow$ *minibatch of m samples from* $P_z(z)$.
      $\mathbf{x} \leftarrow$ *minibatch of m samples from* $P_{data}(x)$.
      Update C by maximizing:
      $Grad(C_w) \leftarrow \nabla_{w_c} \frac{1}{m} \sum_{i=1}^{m} [C(\mathbf{x}^{(i)}) - C(G(\mathbf{z}^{(i)}))]$
      $C_w \leftarrow C_w + \alpha * RMSProp(w, Grad(C_w))$
      $C_w \leftarrow Clip(C_w, -c, c)$
    $\mathbf{z} \leftarrow$ *minibatch of m samples from* $P_z(z)$.
    Update G by minimizing:
    $Grad(\theta_g) \leftarrow \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} [C(G(\mathbf{z}^{(i)}))]$
    $\theta_g \leftarrow \theta_g - \alpha * RMSProp(\theta_g, Grad(\theta_g))$

2.4.5 Known issues

Although WGAN showed evidence of curing many of the problems with GANs, Arjovsky et al. (2017) point out that the way the Lipschitz constraint is held, by weight clipping, is not ideal. If the weights are clipped too much, the gradient might vanish when backpropagating through deep networks. As a result, it will take longer time for the network to learn. If the weights are clipped at a larger value, the gradients might instead become very large and as a result slow down training.

Another issue with using weight clipping to enforce the Lipschitz constraint is that it will bias the critic to converge to simpler functions (Gulrajani et al., 2017), which will have a negative impact since it may no longer be the function that can truly optimize Equation 2.4. It was also reported from Arjovsky et al. (2017) that WGAN generally converges slower than the original GAN, although it is to be considered as more stable and thus have a better chance of reaching convergence. Lastly, as can be seen in Algorithm 2, the optimizer used is not a momentum based such as Adam (Kingma and Lei Ba, 2015), this is

due to the fact that it was experimentally found by Arjovsky et al. (2017) more stable to not use any optimizer that uses momentum.

2.5 Improved Wasserstein-GAN

In the original paper on Wasserstein-GAN, the Lipschitz constraint was enforced by clipping the weights of the critic. As Arjovsky et al. mention, this is not an ideal way of enforcing the Lipschitz constraint and they strongly encourage further research to investigate different approaches on how to enforce it. This was later done by Gulrajani et al. (2017), who showed how weight clipping is affecting the results. The authors introduced a new way of enforcing the Lipschitz constraint: gradient penalty, and argue why this approach might converge faster than the original WGAN, be more stable for different problems such as language modeling and images and also more flexible as it can be applied to different network architectures.

The major difference between improved WGAN and WGAN, is the use of gradient penalty instead of weight-clipping, which will affect the cost function of the critic. By investigating the critic as defined in Arjovsky et al. (2017), it was found that it will look like straight lines between points in P_{data} and P_z. Another property of the optimal critic is that the gradients will have a norm of 1 in most parts of P_{data} and P_z.

2.5.1 Cost function

Given the optimal critic and the difficulties of enforcing the Lipschitz constraint, Gulrajani et al. (2017) choose to compute the gradients of the critic with respect to a new distribution P_{c^*}:

$$\mathbf{x} \backsim P_{data}, \mathbf{z} \backsim P_z, \mathbf{c}^* \backsim P_{c^*}$$

$$\epsilon \backsim U[0, 1]$$

$$\mathbf{c}^* = \epsilon\mathbf{x} + (1 - \epsilon)\mathbf{z} \tag{2.5}$$

The cost function will now contain a gradient penalty term based on the new distribution P_{c^*} as in Equation 2.5

$$\min_G \max_{C \in L} \left(\mathbb{E}_{\mathbf{x} \backsim P_{data}(x)}[C(\mathbf{x})] - \mathbb{E}_{\mathbf{z} \backsim P_z(z)}[C(G(\mathbf{z}))] + \lambda \mathbb{E}_{\mathbf{c}^* \backsim P_{c^*}}[(\|\nabla_{c^*} C(\mathbf{c}^*)\|_2 - 1)^2] \right),$$

$$L \in \text{1-Lipschitz functions} \tag{2.6}$$

The last term in Equation 2.6 is the penalizing term where the norm of the critic's gradient is penalized for how far it is from 1. This comes from the fact that the optimal critic has gradients with norm 1. When λ (the gradient penalty hyperparameter) is large, optimizing Equation 2.6 will result in an optimal critic. If the critic can reach its full capacity by training it to optimal, the generator will be optimized according to the exact Wasserstein distance as defined in Equation 2.3.

2.5.2 Algorithm

Algorithm 3 describes how the training procedure is carried out in improved Wasserstein-GAN in accordance to the implementation by Gulrajani et al. (2017). There is one difference and that is that here we update the critic by maximizing Equation 2.6, whereas in Gulrajani et al. (2017) they multiply the cost function with -1 and minimize. The reason for doing this was to facilitate the comparison of previous algorithms.

Algorithm 3 Implementation of improved WGAN, α = learning rate, λ = penalizing factor

procedure IWGAN
  for number of iterations **do**
    for number of steps to run C **do**
      $\mathbf{z} \leftarrow$ *minibatch of m samples from* $P_z(z)$.
      $\mathbf{x} \leftarrow$ *minibatch of m samples from* $P_{data}(x)$.
      $\epsilon \leftarrow$ *minibatch of m samples from U[0,1].*
      $\mathbf{c}^* \leftarrow$ *minibatch of m samples from* $P_{c^*}(c^*)$.
      Update C by maximizing:
      $Grad(C_w) \leftarrow \nabla_{w_c} \frac{1}{m} \sum_{i=1}^{m} \left[C(\mathbf{x}^{(i)}) - C(G(\mathbf{z}^{(i)})) + \lambda(\|\nabla_{c^*} C_w(\mathbf{c}^{*(i)})\|_2 - 1)^2\right]$
      $C_w \leftarrow C_w + Adam(w, Grad(C_w), \alpha)$
    $\mathbf{z} \leftarrow$ *minibatch of m samples from* $P_z(z)$.
    Update G by minimizing:
    $Grad(\theta_g) \leftarrow \nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \left[C(G(\mathbf{z}^{(i)}))\right]$
    $\theta_g \leftarrow \theta_g - Adam(\theta_g, Grad(\theta_g), \alpha)$

As we can see in Algorithm 3, it is not longer a restriction to use momentum based optimizers as was the case in Algorithm 2. This was something that Gulrajani et al. (2017) found when conducting different experiments and they believed it was due to the fact that the weights are no longer restricted, thus decreasing the impact on the optimization.

2.5.3 Known issues

Improved Wasserstein-GAN (Gulrajani et al., 2017) was introduced 31 March 2017 and at the time this thesis was written there had been no reports of any further issues. There is though one issue that still remains and that is the speed of convergence. Although the improved version of WGAN seemed to converge faster than the original WGAN, as demonstrated by Gulrajani et al. (2017), it is slower than models structured as GANs.

2.6 Text Representation

The internal representation of text within the models can be of several different forms. There exists models where text is represented on character-level (Karpathy, 2015), on word-level (Zhang et al., 2016) as well as sentence-level and even on document-level (Le and Mikolov, 2014). All of these come with their respective pros and cons. Using a representation on character-level means that you can use a small vocabulary (the characters)

and still produce any word. Usually the characters are encoded as one-hot vectors (a vector filled with zeros except for a single element equal to one) as the sparsity isn't an issue. This is true when using a small vocabulary as the alphabet and a few other characters that are common in text. The downside is that the model using a character-level encoding will not only have to learn the correct sentence structure but also how to spell.

When using a word-level approach you are limited by the vocabulary as the model can't output any word not present. Using a large vocabulary entails a possibility of diversity in the output but it comes with the cost of memory consumption. The encoding can be of the one-hot type where the size of the vocabulary is the dimension of the vectors, another option is to use a fixed size vector (Pennington et al., 2014). One-hot encoding will result in very sparse vectors even if the quantity of words used is of a modest size, hence operating on these will be inefficient. However it is easy to present a result in the form of one-hot encoding by using a `softmax` and `argmax` combination and the output will also be in the form of a probability of each word. The final text output is produced by simply taking the index given by `argmax` and using a lookup table to find the corresponding string.

By using a fixed-sized encoding the issue of sparse vectors is removed as the size of the vectors doesn't depend on the vocabulary size. To get the most of this kind of encoding it would be useful if the encoding itself were meaningful and contained information about the relationship between words. Tools that embed this information into the vectors include Word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014). The difference is that Word2vec is based on a predictive model using neural nets and N-grams where GloVe is count based, using dimension reduction of a co-occurrence count matrix. The difficulties arise not when going from string to vector but the other way around. You could have the model output one-hot vectors as stated above but then you will have to deal with sparse vectors yet again, the other option is to output embedding vectors. To go from a vector to a string, the only useful metric is to calculate the cosine similarity, which is done by taking the cosine of the angle between the two word vectors, and choose the string corresponding to the closest (smallest angle) vector.

The consequences is that there is a need for many vector multiplications, as you have to compute the similarity between the outputted vector and every vector in the vocabulary (if one doesn't use clustering to reduce the number of operations). If we are using matrix multiplications there will, in the final stages, also be a sparse matrix and an `argmax` operation.

To understand how word embeddings will look like we can visualize a set in 3-dimensions with Tensorboard, a visualizing tool for machine learning provided by (Abadi et al., 2015), using principal component analysis (see Figure 2.3).

It becomes clear that the approach of choosing the vector with smallest cosine similarity is only going to make sense if words that mean the same thing or are exchangeable, are mapped closely together.

For sentence- and document-level the same pros and cons as for the word-level apply, with the addition that there is no possibility to create new sentences/documents but only to combine the already existing ones. So using these levels of encoding reduces the granularity and hence also the diversity of the output (when the output is not that long).

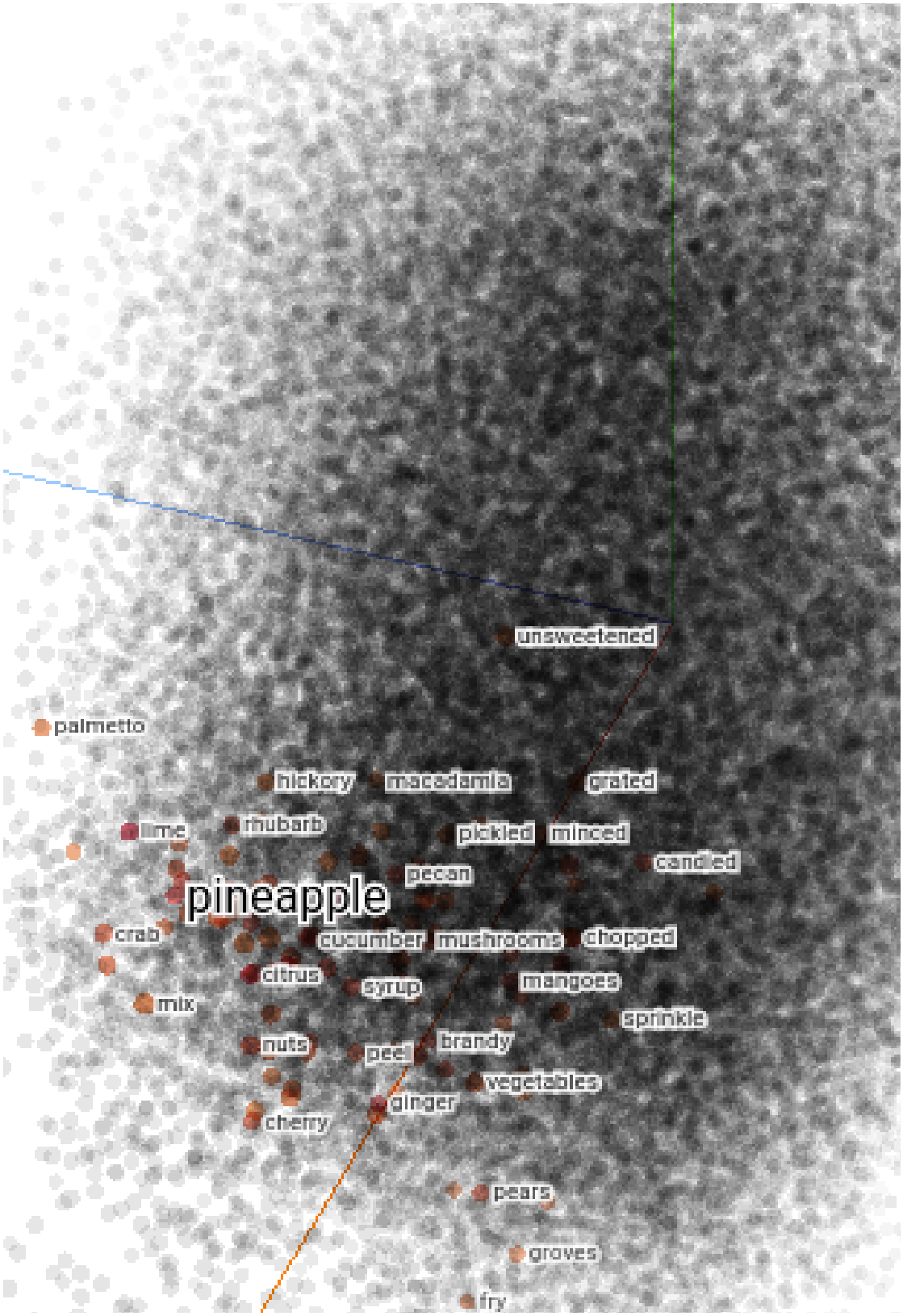

Figure 2.3: The image displays the visualization of the word embeddings with PCA in Tensorboard. Here we have searched for the word pineapple and the words with the closest cosine similarity are highlighted.

Chapter 3
Approach

This chapter describes how we have chosen to approach the problem stated in the beginning of the report. There is also descriptions of the hardware used, how we represent the text, what kind of architectures we have in our different models and finally a motivation of our approach.

3.1 Overall Approach

To answer the questions stated in the goal and problem formulation section, the following methods were used. Firstly we conducted a literature study, were the goal was to see what has been done previously and how it was done. How have they represented the words in the articles? What types of networks did they use? What tools did they use? In what format/type were their output? What were the results? Secondly we investigated and tried to implement generative adversarial networks (GANs), train them for a shorter period of time and finally do further optimization and continue to train the model achieving the best results.

3.2 Setup

The training of the models was done on a machine running Ubuntu 16.04 with a Nvidia Titan X GPU 12 GB. The machine learning library of our choice was Keras with Tensorflow as backend for some models and standalone Tensorflow for some.

3.2.1 Corpus

For the corpus we used roughly one million selected samples from RCV1 (Reuters Corpus Volume 1) (Lewis et al., 2004), 10% of the data selected was used for validation. Some

preprocessing and pruning was done to retrieve articles which mostly consists of sentences (not only tables filled with numbers for instance) and also to convert them from xml to pure text format. We also formatted the texts by encapsulating each article by <*START*> and <*END*> tags as well as <*TITLE*> and <*/TITLE*> tags surrounding the headlines in each article. The result was a mixture of news articles in English published by Reuter between August 20, 1996, and August 19, 1997. The lengths of the articles vary from a few hundred words to a number of thousand words and the topics cover a wide range of the usual English news text.

When the articles are read from disk and being prepared as input to the models, we retrieve the headlines and the corresponding bodies. Firstly we pad or crop the headline to a specified sequence length, then we split the bodies into sequences of the same length (where the last sequence might require padding), with no regards to whether the split occurs in the middle of a sentence or not. For padding we use the <*PAD*> symbol and for unknown tokens we use <*UNK*>. All characters are also converted to lowercase. To tokenize the text into sequences we used the following regular expression:

```
(?![0-9])[\w]+|[0-9;:&.,!?;\%\-\+\=\*\@\£\$)(\/\"\']
```

Essentially what we do is to split on numbers and other non-alphabetical characters. The reason behind this is to reduce the number of needed word embeddings and we also want the network to figure out for itself a useful combination of words. This in turn also has the effect that the models will have to learn for instance where to put apostrophes.

For the case when we have text as input to the generator instead of noise, we used the same text as for the real samples to the discriminator with the difference that the text to the discriminator is shifted one sequence. This is so that the net will learn an ongoing sequence and so that the output from the generator is compared to the "real" output.

3.2.2 Training

To visualize the training process we used Tensorboard by plotting the loss of both the generator and the discriminator. However, it has been mentioned by Arjovsky et al. (2017) that the loss of the models using the standard GAN does not typical reflect the quality of the generated samples, making the training curves hard to interpret. This was to some extent solved by using WGANs instead but in the end it is the generated text that needs to be inspected. For this reason we have printed samples after every 500 training iterations. The goal for the generator is to generate samples representing an estimate of the distribution of news articles.

For the embeddings we used the pretrained 100-dimensional vectors from Stanford, trained using GloVe (Pennington et al., 2014). In total, 400,000 word embeddings were trained on both Wikipedia and Gigaword 5. We normalized all embedding vectors and some data about the resulting vectors is presented in Table 3.1.

Standard deviation	Mean	Min	Max
0.09999	0.00119	-0.59135	0.55772

Table 3.1: The table presents some statistics about the values in the word embedding vectors that were used.

3.3 Models

The standard GAN has randomized noise as input to the generator but in our case where we want to generate coherent articles we want to control the input and have the same input being mapped to a single output. We make use of inputs in the form given by Figure 3.1. This is also the same shape of the output from our different models regardless of the input.

3.3.1 Baseline

To make sure that the models we implemented really do make a difference, we created a baseline for both character and word models. This baseline is to simply output random tokens. For the character model, we used a discrete uniform distribution from NumPy (van der Walt et al., 2011) `random.randint` to give us a index of a character in our vocabulary and for the word embeddings, we used NumPys `random.uniform` to give us values in the range of min-max given by Table 3.1 and mapped these generated embedding vectors to the closest word present in the vocabulary.

3.3.2 GAN model

The model using the standard GAN objective was implemented using Keras (with Tensorflow as backend). We decided to use the encoder-decoder model where the input to the encoder is mapped to a vector of a given size and then the decoder maps this vector to the final output. We used two LSTM layers with 300 hidden cells in each for both the encoder and decoder. For the discriminator the type of network used was a CNN in accordance to (Zhang et al., 2016) where they stated good performance. For the GAN model, we used text as input in the form of word embeddings.

3.3.3 WGAN models

The other models were implemented using the improved WGAN objective and was incorporated using Tensorflow. In total four models were trained using this objective. All these models have batch size=512, sequence length=32, optimizer=Adam in common. Two different structures of networks were used, one using CNN for both the generator and the critic as well as one using LSTMs for the generator and CNN for the critic. For the LSTM generator we used four layers with a hidden size of 1024 followed by a dense layer and a simple adjustment. By looking at what values the LSTM model produced we noticed that there were not any negative values even though most of the word embeddings contain

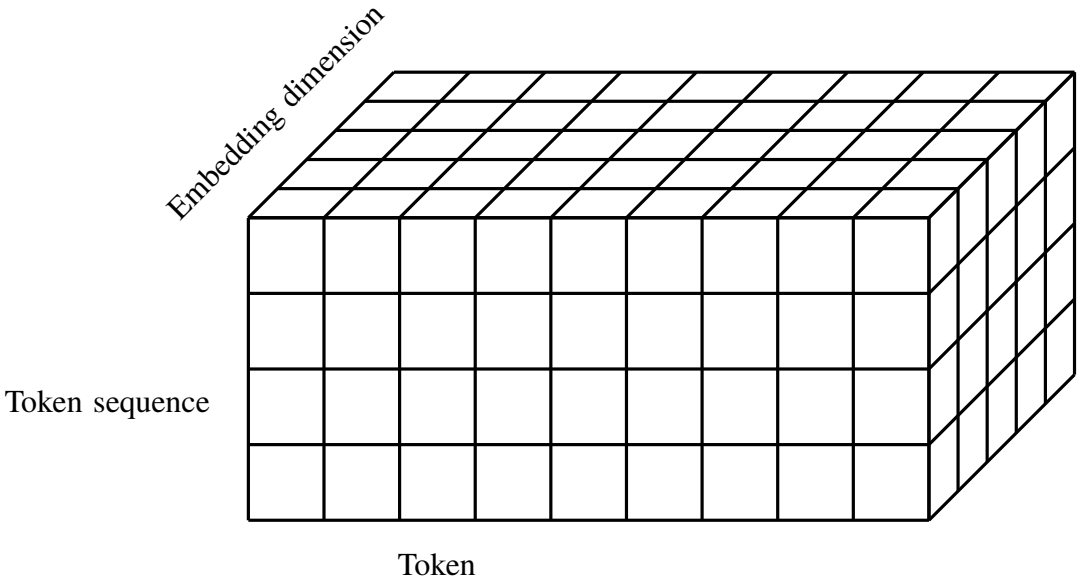

Figure 3.1: The figure illustrates how the input data is fed to the neural nets. The first dimension is the sequence of tokens, the second corresponds to a single token and the third contains the embedding vector for each token. The complete cuboid represents a single batch.

Model	Generator	Input	Encoding	Level
GAN	LSTM	text	word embeddings	word
char-noise_in-CNN	CNN	noise	one-hot	char
char-text_in-CNN	CNN	text	one-hot	char
word-text_in-CNN	CNN	text	word embeddings	word
word-text_in-LSTM	LSTM	text	word embeddings	word

Table 3.2: These are the five models we have trained and evaluated. The discriminator has always consisted of a CNN network, since early studies showed it was good for sentence classification, word order and so on. So we have only tried changing the generator from CNN to LSTM. As can also be seen, we have used noise as input only once and this is because we can hopefully reach better results and have more control of the output if we start with text instead. Otherwise the hyperparameters are the same for every model.

negative values. We therefore added a negative term to the output of the LSTM generator, this term was the minimum value found in the embedding vectors (≈ -0.6). This shifted the possible output of our model so that it now would be able to capture values between -0.6 and 0.6 (in accordance to Table 3.1), corresponding to the span of our embeddings (see Figure 3.3 for an overview). For the CNN generator and discriminator, we use the same structure as in the repository related to Gulrajani et al. (2017) with a few alterations. The activation function in the generator's residual blocks (resblock) when using word embeddings will instead of being a ReLU be a tanh, we make this change since that we want to have outputs in the range of [-1, 1] instead of [0, 1] when using character encoding, we will also have a `softmax` operation before the output (see Figure 3.2). The discriminator makes use of a convolutional layer followed by five resblocks, (ReLU-conv-ReLU-conv) and finishes with a linear layer (see Figure 3.4). For a summary of the models see Table 3.2

The first model used noise as input to the CNN generator and one-hot-character-encoding. The second had text as input to the same type of CNN network and character encoding. The third was the same as the second, with the one-hot-character-encoding replaced by word embeddings of 100 dimensions. The last model made use of the LSTM generator with text as input and word embeddings as encoding.

As we are using the WGAN objective there was no pretraining of neither the generator nor the critic as this objective is more stable than the GAN objective.

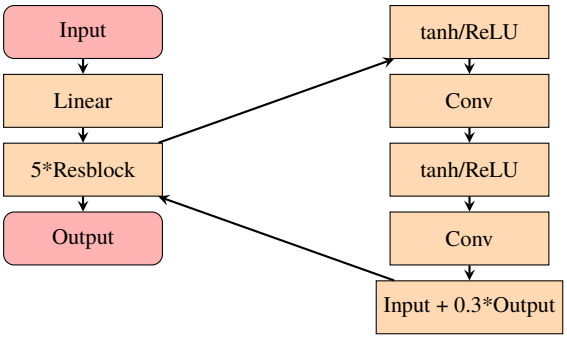

Figure 3.2: This is the structure of the convolutional generator, we have first a linear/dense mapping to reduce the dimensionality of the input. Then we have five residual blocks consisting of a mixture of tanh/ReLU and convolutional layers (kernel size of 5). The ReLU is used for characters (one-hot) and the tanh is used for words (word embeddings). The output of the residual block consists of the untouched input to the block with the addition of a fraction of the result of the above operations. If we are training a model on characters then there will also be a softmax operation before the output node.

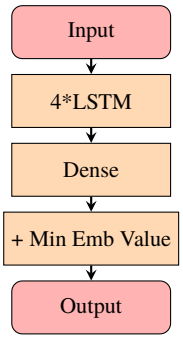

Figure 3.3: This is the structure of the LSTM generator, it consists simply of four LSTM layers (with 1024 hidden units) followed by a dense layer and finally a bias adjustment of the output.

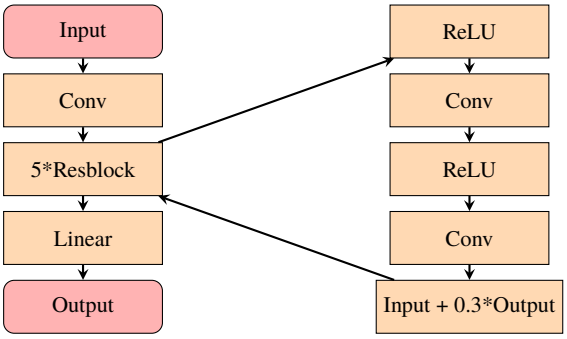

Figure 3.4: This is how the discriminator looks like. We have a convolutional layer at the top followed by five resblocks (using ReLU) and finally a linear layer.

3.3.4 Motivation of approach

Our hypothesis is that with these models, using different architectures, we should be able to see which approach is better and more suitable for generating text. The random generator as a baseline will of course not produce any satisfying results, it will rather show what the generator actually needs to do, which is finding a function in the space of word embeddings that will be an estimate of sequences within news articles. By first investigating how others have used characters and one-hot encodings we want to find current problems and hopefully motivate the use of word embeddings as the next step to take. As far as we know, this is the first time where word embeddings are used as the final output.

As mentioned in the theory, the discreteness of the output has to be solved and the way we do this is by using one-hot encoding for characters and word embeddings for words. We only used 181 characters for the models on character level because we chose to only use the characters present in the Google billion words data set (Chelba et al., 2013).

The GloVe word embeddings encode words that are similar to the same region in the word embedding space and this will hopefully make it easier for the generator as it just has to map the input to the correct part of space, to generate words that are correct given the input. The corpus used as input is encoded by matching words with corresponding word embeddings which are in total 400,000, 100 dimensional vectors. This means that we have a vocabulary of 400,000 words which, unfortunately, will not cover all words such as certain names mentioned in the corpus. This will limit the expressive power of the generator but we think it is a reasonable limitation. Stanford also has pretrained word embeddings of 300 dimensions which should perform better since they contain in a sense more information, however we chose to use only 100 dimensions as to simplify for the generator by reducing the amount of values to learn.

We are then feeding the generator with a sequence of words from the corpus and will let the discriminator affect the generator by the loss it gives for the generated text. The chosen sequence length is going to be a an important parameter as the generator has to draw conclusions on the given input. The reason for choosing both CNN and LSTM networks, was because they both consider the current context in their learning process. The CNN using kernels of a fixed size that strides over the input, while the LSTM is a recurrent neural network and has its own cell state with memory. This is considered when choosing sequence length. If it is given one word then we can not expect it to produce any coherent text. Five words are what the CNN will look at since the kernel has the same size while a LSTM network can learn to remember sequences of different length.

Another parameter that is important is the batch size since we are only updating after one batch, it means that the weights of the neural networks are updated according to the mean loss for the whole batch. If we use a large batch size then the input will vary more since it will contain multiple articles, that do not need to be related. The updates may therefore no longer be satisfactory. A smaller batch size might be more suitable but takes longer time.

Lastly, the reason for only training the models for 15 epochs, was that it was a very time consuming experiment and we were only going to use these models for comparison. Therefore, we decided that there was no need for any further training.

Chapter 4
Evaluation

Here we present how we have chosen to evaluate our models with motivation as well as present the achieved results. We finish this chapter with a discussion of the aforementioned outcome.

4.1 Metrics Used

The best way to test a language model is to use it in a designated environment and measure how much this model improves the overall experience. However this is not always as easily done as said. It can be quite expensive and dependent on the application, hence there exists a need for a metric independent of the environment and simple to obtain, (Jurafsky and Martin, 2016).

4.1.1 Perplexity

As a measure of how good the generated output was compared to the expected "real" output, we used perplexity. We trained N-gram models on both of these documents of text and used these to calculate the probabilities (Equation 4.1) that is then used to calculate the entropy (Equation 4.2) and finally the perplexity (Equation 4.3).

To deal with N-grams the model has not seen, we make use of a smoothing technique called "backoff stupid" (Brants et al., 2007). It essentially works as follows: if the current N-gram is not available then we will revert to using the counts from a simpler (N-1)-gram (multiplied by a backoff factor α. As an examples consider the trigram ("Dogs", "like", "treats") if we don't have any counts for that trigram we will instead look for counts of the bigram ("like", "treats"). If also this count is missing we will revert to the base case of the unigram ("treats"), see Equation 4.4.

$$P(w_i|w_{i-1}, ..., w_{i-N+1}) \approx \frac{Count(w_i, w_{i-1}, ..., w_{i-N+1})}{Count(w_{i-1}, ..., w_{i-N+1})} \tag{4.1}$$

$$H(T) = \frac{1}{|T|} \sum \log P(k), k = (w_i, ..., w_{i-N+1}) \in T \tag{4.2}$$

$$PP(T) = 2^{H(T)} \tag{4.3}$$

Where T is a set of N-grams (sequences of N succeeding words) retrieved from text.

$$S(w_i|w_{i-k+1}^{i-1}) = \begin{cases} \frac{Count(w_{i-k+1}^{i})}{Count(w_{i-k+1}^{i-1})}, & \text{if } Count(w_{i-k+1}^{i}) > 0 \\ \alpha S(w_i|w_{i-k+2}^{i-1}), & \text{otherwise} \end{cases} \tag{4.4}$$

To evaluate the models we used a held-out set of sentences retrieved from the corpus used in the training. From this set we extracted 2000 sentences that were used to compare the constructed N-gram models (one for real and one for generated text). We used an α value of 0.4 as proposed by the authors of Brants et al. (2007).

4.1.2 Human evaluation

Because of the complexity of natural languages there exists no good general metric for evaluation of quality and taking into mind that our goal is to produce texts indistinguishable from man made, we therefore chose to also have a human evaluation where we present text taken from the training corpus and text generated by our models to human volunteers. The task for these volunteers was to judge the text and decide whether they were created by man or machine. Each piece of text was judged upon sentence quality (a subjective measurement) and readability/understandability, using grading scales (0-10). Finally a binary decision about the source of the text was asked for (human or machine). All subjects had also been informed that capitalization and other giveaways, that would introduce obvious clues and bias the decision making, had been removed.

4.2 Results

Apart from presenting the results of the metrics introduced in the previous section we will also provide some graphs concerning the distribution of the outputs. First for each model, we present the distribution of both real and generated text, then the generated distribution is presented. To see, in some sense how well structured or correctly the generator was at positioning characters/words, we show the distribution of bigrams for both real and generated text. The distribution of the real text, both characters and words, are from the corresponding expected output so that it really was the text the generator should have produced, or close to. We want to point out that for the distributions of the model using characters with noise as input, there will not be any expected output. Instead we have used the same expected output as when having text as input. This will not affect the results since an optimal generator should produce distributions similar to the expected output. As we are producing text we will also show samples of generated text in Section 4.2.3.

Model	Perplexity real	Perplexity generated	Relative perplexity	Unique tokens
char-noise_in-CNN*	13.01482	12.97411	0.99687	34
char-text_in-CNN*	13.01482	13.49805	1.03713	32
word-text_in-CNN	578.03741	1108.60480	1.91788	10795
word-text_in-LSTM	578.03741	1356.24841	2.34630	28005

* Per character.

Table 4.1: The table provides the perplexity of the models we have trained. The real perplexity is the calculated perplexity for the real text taken as samples from the corpus and the generated is samples from the generated output. The relative perplexity is defined as $\frac{Perplexity\ generated}{Perplexity\ real}$. We used bigrams and 2000 validation sentences.

We are aware of that there is no "fair" way to compare the models using characters and the ones using words. There are fewer characters than words and the perplexity should therefore be lower, as can bee seen in Table 4.1.

4.2.1 Results using characters

The first model is the random generator, producing characters randomly. This is considered to be the baseline as mentioned in Section 3.3.1. The distribution of randomly generated characters can be seen in Figure 4.1-4.3.

The results of the CNN model using characters and noise as input are presented in Figure 4.4-4.6.

Using text rather than noise as input, keeping the same CNN model with characters, the distributions in Figure 4.7-4.9 were produced.

4.2.2 Results using words

This section contains results of the models when using words represented as word embeddings as input. The baseline using a random word generator is presented first in Figure 4.10-4.12.

The results of the CNN model using words as input and output are presented in Figure 4.13-4.15.

The distribution of the generated text using the LSTM model with words are presented in Figure 4.16-4.18.

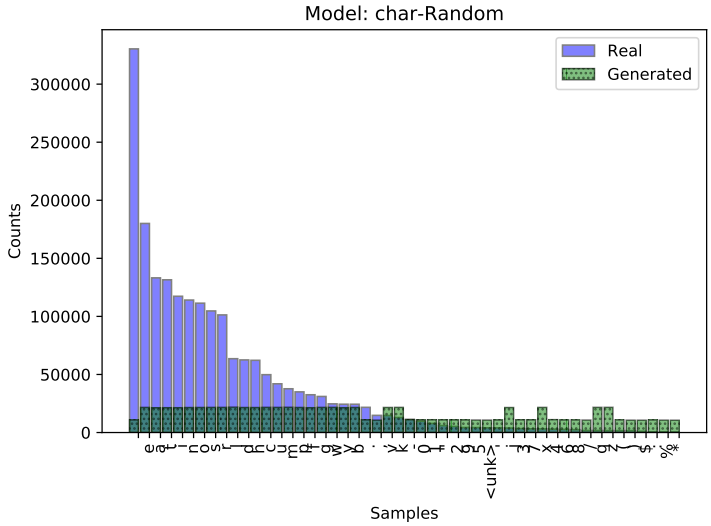

Figure 4.1: The image displays the distribution of the 50 most common characters in real text and the number of times these characters occurred in generated text from the model using characters and a random generator.

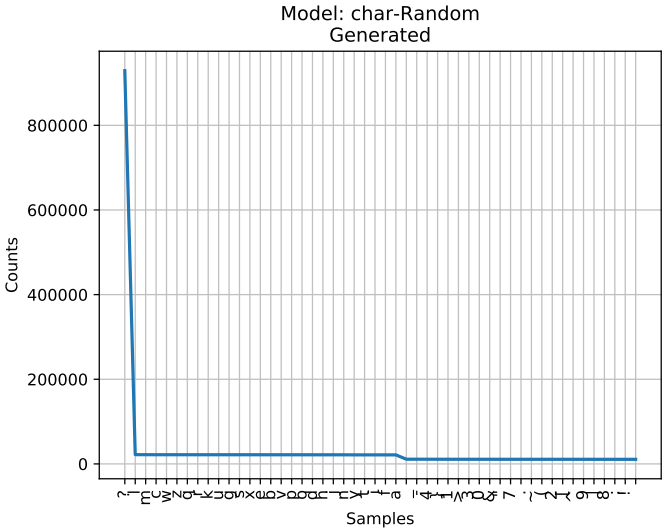

Figure 4.2: The image displays the distribution of the 50 most common characters in generated text from the model using characters and a random generator. The reason for the peak on the question mark is that unknown tokens (various random encoded internet characters) were replaced by a question mark.

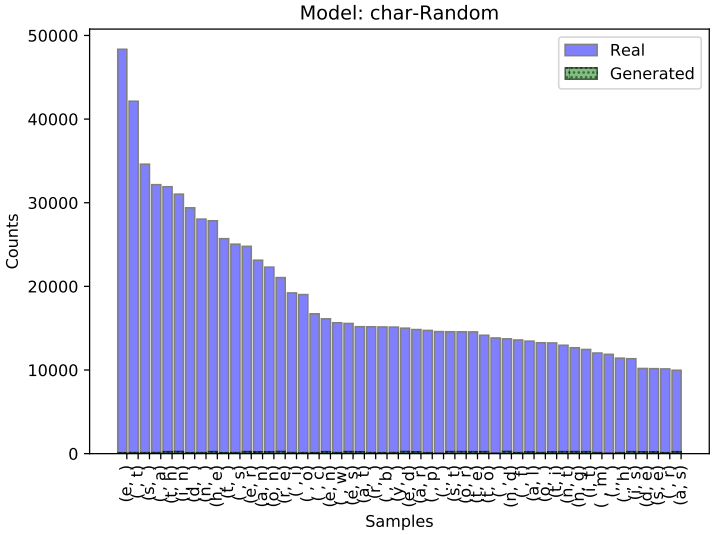

Figure 4.3: The image displays the bigram distribution of the 50 most common characters in real text and the number of times these characters occurred in generated text from the model using characters and a random generator. Since it is a random generator there is a very low chance of actually follow the bigram distribution of real text, which is why we do not see the generated distribution

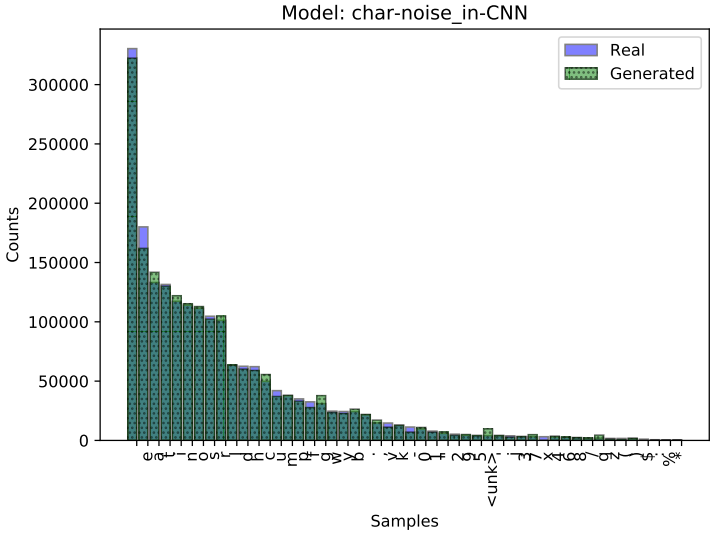

Figure 4.4: The image displays the distribution of the 50 most common characters in real text and the number of times these characters occurred in generated text from the model using characters, noise as input and a CNN architecture.

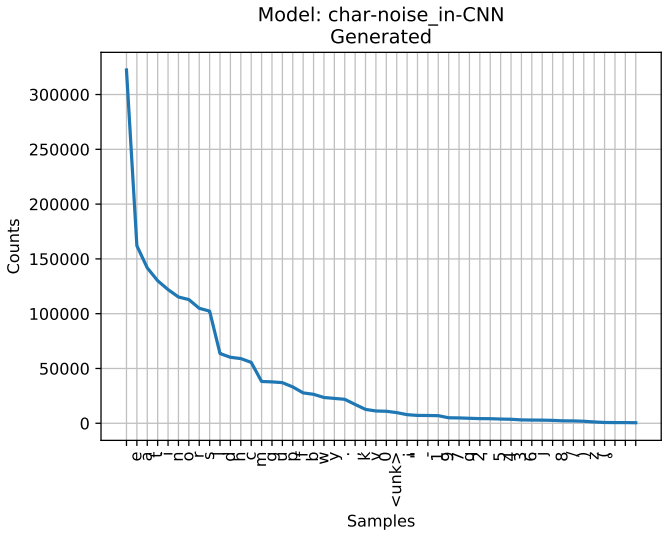

Figure 4.5: The image displays the distribution of the 50 most common characters in generated text from the model using characters, noise as input and a CNN architecture.

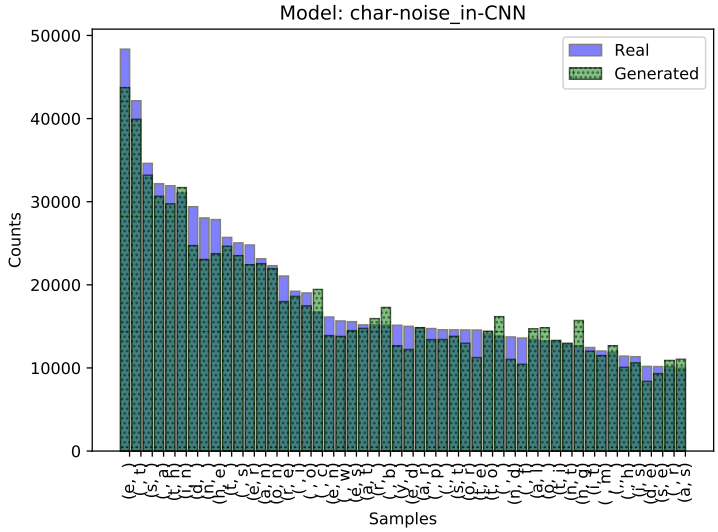

Figure 4.6: The image displays the distribution of the 50 most common bigrams in real text and the number of times these bigrams occurred in generated text from the model using characters, noise as input and a CNN architecture.

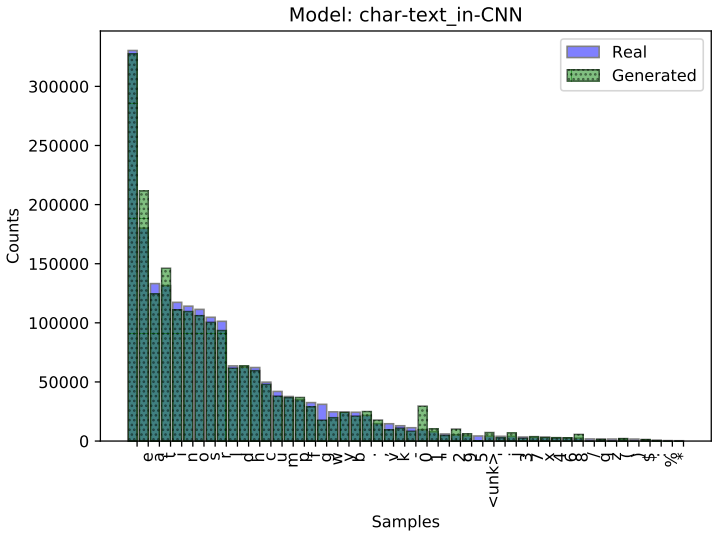

Figure 4.7: The image displays the distribution of the 50 most common characters in real text and the number of times these characters occurred in generated text from the model using characters, text as input and a CNN architecture.

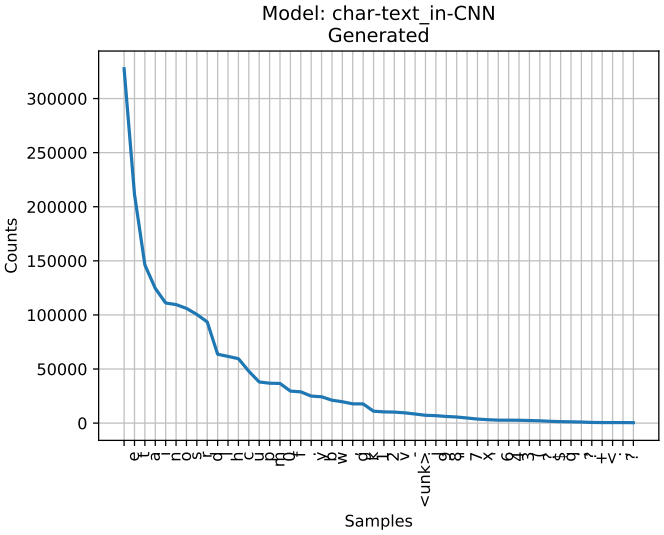

Figure 4.8: The image displays the distribution of the 50 most common characters in generated text from the model using characters, text as input and a CNN architecture.

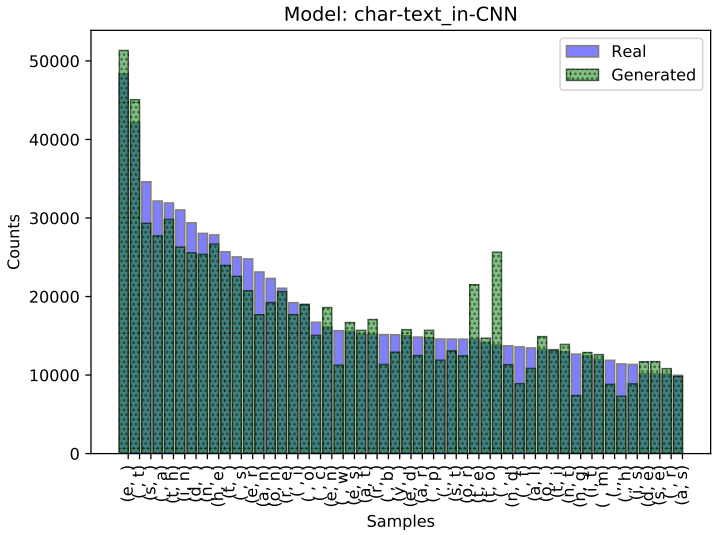

Figure 4.9: The image displays the distribution of the 50 most common bigrams in real text and the number of times these bigrams occurred in generated text from the model using characters, text as input and a CNN architecture.

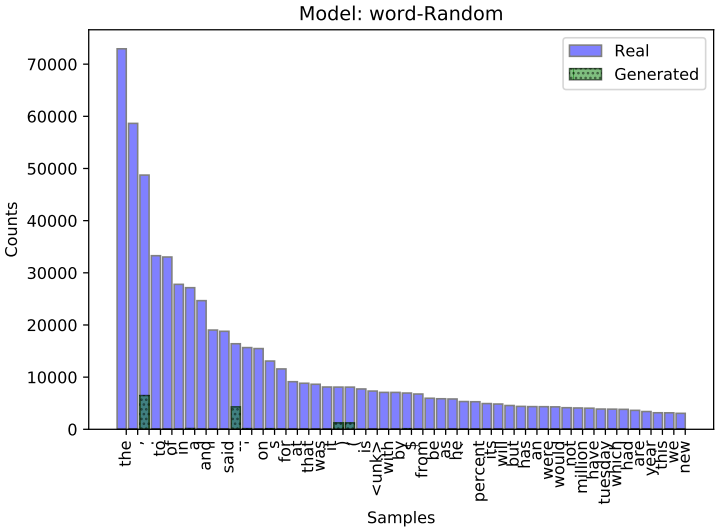

Figure 4.10: The image displays the distribution of the 50 most common words in real text and the number of times these words occurred in generated text from the model using words and a random generator.

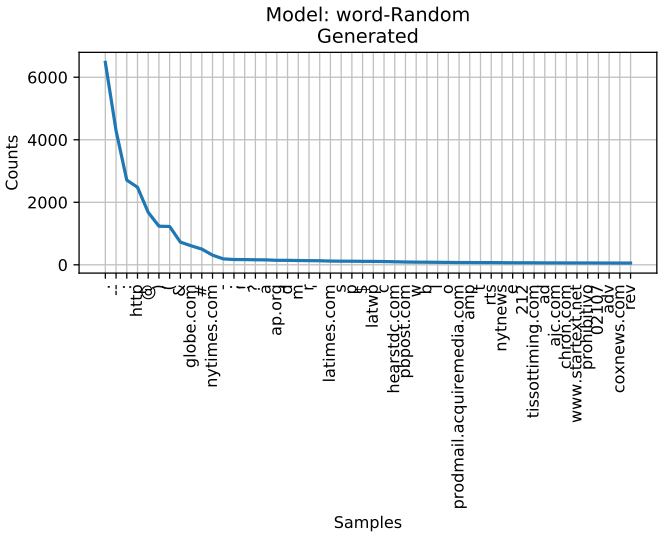

Figure 4.11: The image displays the distribution of the 50 most common words in generated text from the model using words and a random generator.

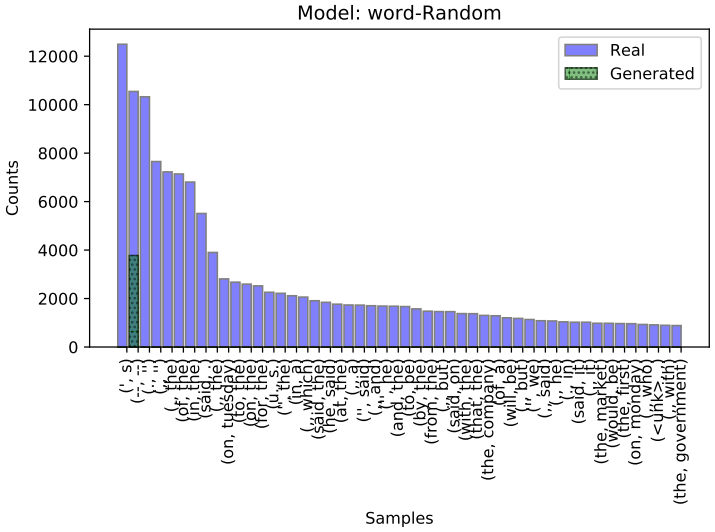

Figure 4.12: The image displays the distribution of the 50 most common bigrams in real text and the number of times these bigrams occurred in generated text from the model using words and a random generator.

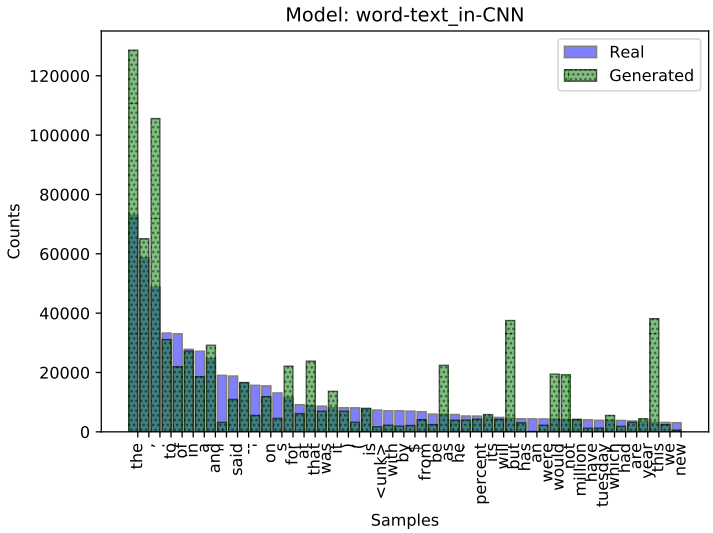

Figure 4.13: The image displays the distribution of the 50 most common words in real text and the number of times these words occurred in generated text from the model using words, text as input and a CNN architecture.

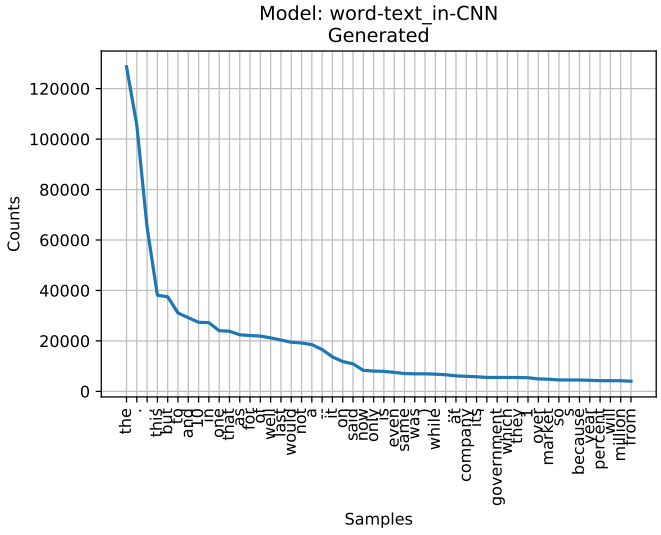

Figure 4.14: The image displays the distribution of the 50 most common words in generated text from the model using words, text as input and a CNN architecture.

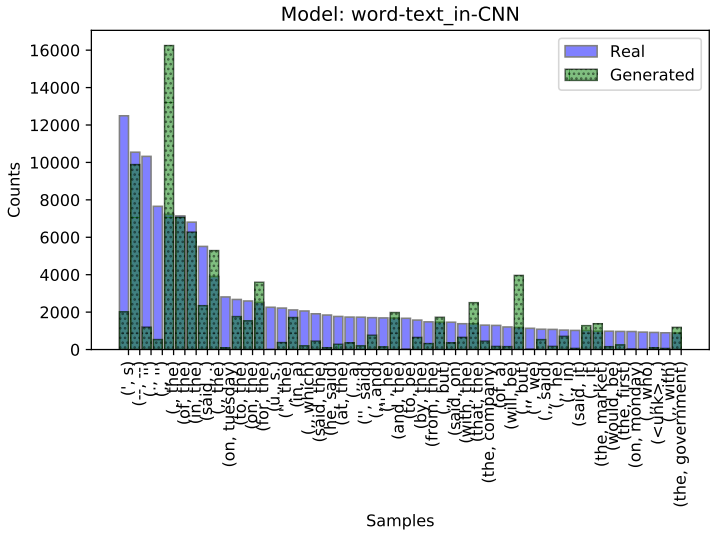

Figure 4.15: The image displays the distribution of the 50 most common bigrams in real text and the number of times these bigrams occurred in generated text from the model using words, text as input and a CNN architecture.

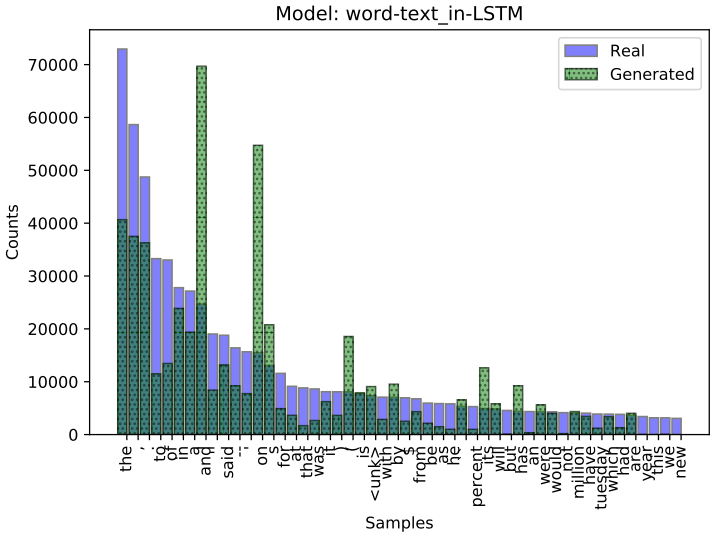

Figure 4.16: The image displays the distribution of the 50 most common words in real text and the number of times these words occurred in generated text from the model using words, text as input and a LSTM architecture.

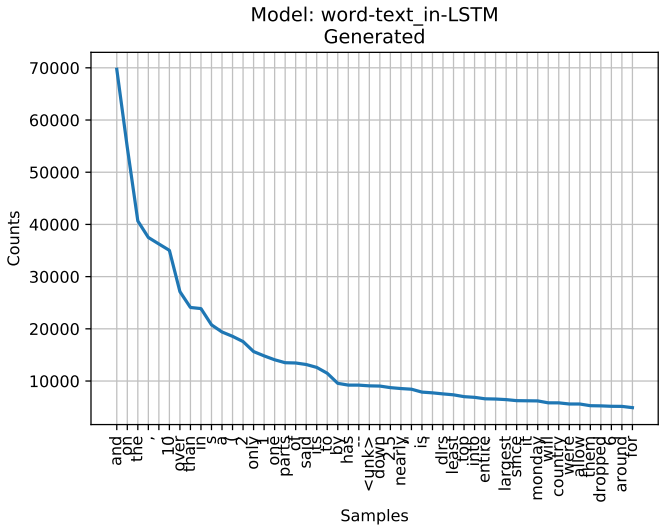

Figure 4.17: The image displays the distribution of the 50 most common words in generated text from the model using words, text as input and a LSTM architecture.

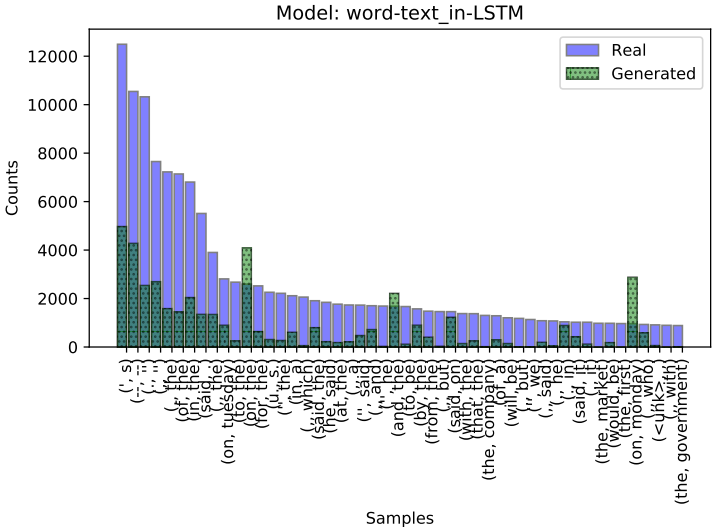

Figure 4.18: The image displays the distribution of the 50 most common bigrams in real text and the number of times these bigrams occurred in generated text from the model using words, text as input and a LSTM architecture.

4.2.3 Generated text

Here we present some chosen input together with the corresponding output (Output 4.1-4.5). We also present the cosine similarity between the words in the real text and words in generated text (Table 4.2-4.3), to show if the words are close in the word embedding space. For the models using characters, the output has not been modified so the models themselves output space and all other characters. For the models using words there is a small set of rules applied to format numbers using dots and also some spacing rules that should produce a more realistic text. All output as well as input was converted to lowercase, to ease the training.

All the output in this section comes from the models having seen 15 million samples, approximately 15 epochs.

```
Input:
jailed mexico priests say they were tortured.

Expected output:
two mexican priests and two indian peasants jailed in the
   southern state of chiapas on murder

Output:
the but but but the the the the the the the the the the the
    the
```

Output 4.1: This is text generated using the GAN model. The tokens after "Input:" is the sequence that was fed to the generator and the sequence after "Expected output:" is the sequence that the network is supposed to output. The output at the bottom is what was generated and it consists of only a few words, this we believe to be due to mode collapse. Both the input and the expected output was taken from RCV1 (Reuters Corpus Volume 1) Lewis et al. (2004).

```
at minicate," the prestion. - Na
r the offers. Theration to git w
oum Aierain snroor vative Corist
anding to month. Reachings "The
nMirch Sanaacer as the batiori t
rn to Vister-Juyo's stage ynt Cr
wt good retailing selpedyjhan, s
rod sales by compless think conf
```

Output 4.2: This is text generated using the char-noise_in-CNN model after 15 epochs. In this case we don't have any text as input and therefore we don't have any expected output either. The text looks as if it could be English and some words are, but it is mostly gibberish. Each line corresponds to a new sample.

```
Input:
efficiencies by making additiona

Expected output:
l acquisitions and taking relate

Output:
to trade  6 a could increased a
```

Output 4.3: This is text generated using the char-text_in-CNN model after 15 epochs. We see more of an English looking text here, but the context doesn't really make sense. Both the input and the expected output was taken from RCV1 (Reuters Corpus Volume 1) Lewis et al. (2004).

```
Input:
which has at least 11,000 troops guarding oil facilities.
   but industry insiders estimate an oil company ' s
   security bill adds up to 15 percent

Expected output:
to the cost of operating in colombia. the country ' s
   second- largest oil pipeline, which pumps an average of
   about 180,000 barrels

Output:
market well the has been well on, used for smuggle even as
   ., gunfire reforms this the same wager well only one one
    the likely. as 2 last.
```

Output 4.4: This is text generated using the word-text_in-CNN model after 15 epochs. Both the input and the expected output was taken from RCV1 (Reuters Corpus Volume 1) Lewis et al. (2004).

cosine similarity	word pair
0.523052275181	(to,market)
0.783084392548	(the,well)
0.572136759758	(cost,the)
0.6645143432	(of,has)
0.54659473896	(operating,been)
0.775556531482	(in,well)
0.332902302319	(colombia,on)
0.875598728657	(.,,)
0.638985135495	(the,used)
0.687446832657	(country,for)
0.00338191422634	(',smuggle)
0.373330190977	(s,even)
0.0504865399913	(second-,as)
0.504115641117	(largest,.)
0.452609419823	(oil,,)
0.165595079398	(pipeline,gunfire)
0.351772516966	(,,reforms)
0.816518425941	(which,this)
0.219204455614	(pumps,the)
0.746450826314	(an,same)
0.123519420536	(average,wager)
0.703419327736	(of,well)
0.730744822666	(about,only)
0.176737155124	(180,000,one)
0.269187569618	(barrels,one)
-0.046010620892	(<PAD>,the)
0.0537598095834	(<PAD>,likely)
0.0277358759195	(<PAD>,.)
0.0859375121072	(<PAD>,as)
0.0152215640992	(<PAD>,2)
-0.0605937875807	(<PAD>,last)
0.0277358759195	(<PAD>,.)
Average: 0.380960361733	

Table 4.2: The table presents cosine similarities between expected output and generated output from Output 4.4.

cosine similarity	word pair
0.727724075317	(the,,)
0.358857048614	(accuracy,change)
0.698415338993	(,,are)
0.0433240260675	(adequacy,have)
0.632742226124	(or,want)
-0.102951368865	(completeness,investors)
0.564333200455	(of,start)
0.517440795898	(information,by)
0.621940746539	(and,a)
0.761393652045	(is,very)
0.525390179479	(not,compromise)
0.323087534536	(responsible,revenue)
0.784133195877	(for,.)
0.251485407352	(any,analyses)
0.236745105774	(errors,said)
0.736122131348	(or,they)
-0.0127025833353	(omissions,was)
0.736353428799	(or,as)
0.101893976331	(for,drugmaker)
0.766874372959	(the,with)
0.523790419102	(results,its)
0.0281946472824	(obtained,undermining)
0.564072668552	(from,bring)
0.581362962723	(the,hopes)
0.644826591015	(use,with)
0.434332966805	(of,australia)
0.6454402631	(such,in)
0.0998355895281	(information,portugal)
1.0	(.,.)
-0.136632487178	(crisil,britain)
0.359177640627	(is,shares)
0.785137832165	(also,.)
Average: 0.462566924501	

Table 4.3: The table presents cosine similarities between expected output and generated output from Output 4.5.

```
Input:
taken due care and caution in compilation of data for this
   product. information has been obtained by crisil from
   sources which it considers reliable. however, crisil
   does not guarantee

Expected output:
the accuracy, adequacy or completeness of information and
   is not responsible for any errors or omissions or for
   the results obtained from the use of such information.
   crisil is also

Output:
, change are have want investors start by a very compromise
    revenue. analyses said they was as drugmaker with its
   undermining bring hopes with australia in portugal.
   britain shares.
```

Output 4.5: This is text generated using word-text_in-LSTM model after 15 epochs. Both the input and the expected output was taken from RCV1 (Reuters Corpus Volume 1) Lewis et al. (2004).

4.3 Final Model

Based on the results of the different models above we tried to tailor a better model, therefore we now show results of a LSTM model with text as input but with a smaller batch size as well as a shorter sequence length.

In Table 4.5 we present the calculated perplexity for the final model as well as the number of unique tokens generated. Output generated by our final model can be found in Output 4.6. The distributions of the final model is displayed in Figure 4.19-4.21.

```
Input
few years is part of a western effort

Expected output
to sap their national economic strength. but

Output
party to undermine that development to leave.
```

Output 4.6: This is text generated using the final word-text_in-LSTM model after 64 epochs. Both the input and the expected output was taken from RCV1 (Reuters Corpus Volume 1) Lewis et al. (2004).

The final model was also used to produce longer articles, where the model was fed with headlines and had to continue, using its own output as input, sequence after sequence. The articles can be found in Appendix A.

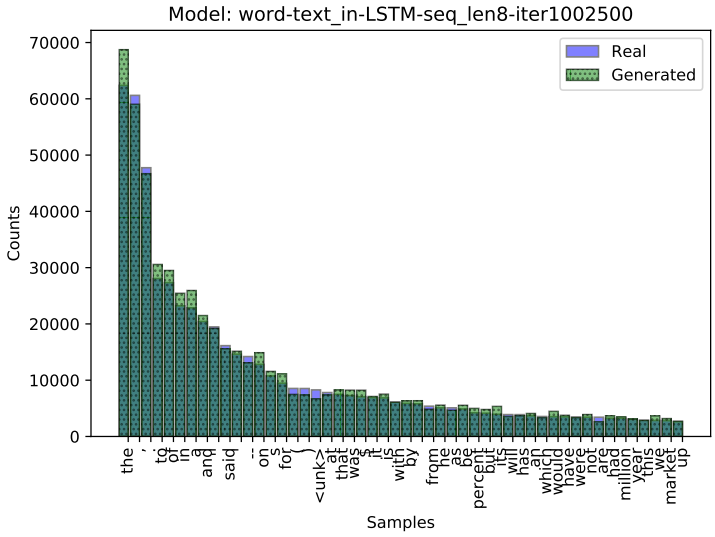

Figure 4.19: The image displays the distribution of the 50 most common words in real text and the number of times these words occur in generated text from the model using words, text as input and a LSTM architecture that has been running for a longer time.

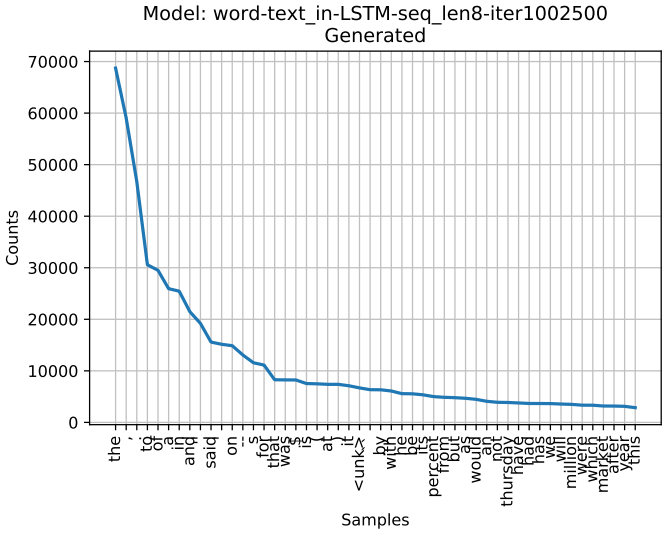

Figure 4.20: The image displays the distribution of the 50 most common words in generated text from the model using words, text as input and a LSTM architecture that has been running for a longer time.

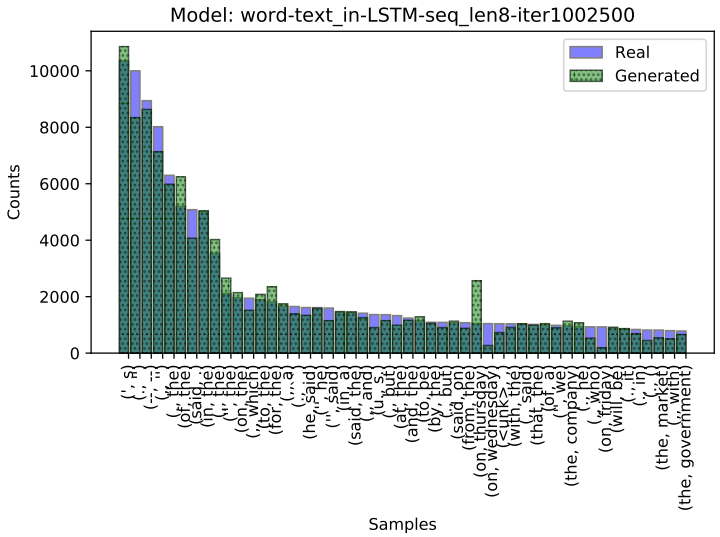

Figure 4.21: The image displays the distribution of the 50 most common bigrams in real text and the number of times these bigrams occur in generated text from the model using words, text as input and a LSTM architecture that has been traiened for a longer time.

cosine similarity	word pair
0.482047945261	(to,party)
0.247945472598	(sap,to)
0.421248550489	(their,undermine)
0.562421612451	(national,that)
0.72526542695	(economic,development)
0.482985556126	(strength,to)
0.598414540291	(.,leave)
0.904912769794	(but,.)
Average: 0.553155234245	

Table 4.4: The table presents cosine similarities between expected output and generated output from Output 4.6.

Model	Perplexity real	Perplexity generated	Relative perplexity	Unique tokens
word-text_in-LSTM-seq8	566.655205	1139.665736	2.011216	35999

Table 4.5: The table provides the perplexity of the final model we trained. The real perplexity is the calculated perplexity for the real text taken as samples from the corpus and the generated is samples from the generated output. The relative perplexity is defined as $\frac{Perplexity\ generated}{Perplexity\ real}$. We used bigrams and 2000 validation sentences.

4.4 Human Evaluation

The subjects were of variant ages and with different backgrounds and none of them had any knowing of our previous results. They were asked questions concerning quality, readability, understandability and if the given sequences of text were produced by a human or machine. The sequences were of the same length as the models were trained on. In total there were 10 different models, five of them were no models, but instead real text taken from the corpus we trained our models on. The result is displayed in Table 4.6 and the questionnaire can be found in Appendix B.

Model	Quality	Readability	Understandability	Machine generated
Char-noise_in-CNN	4	4	4	5/6
Char-text_in-CNN	4	5	4	4/6
Word-text_in-CNN	4	5	5	**3/6**
Word-text_in-LSTM	**6**	**6**	5	**3/6**
Word-text_in-LSTM-seq8	**6**	**6**	**6**	5/6
Real-text-seq32	6	7	6	1/6
Real-text-seq32	7	7	6	2/6
Real-text-seq32	8	8	7	2/6
Real-text-seq32	8	8	7	3/6
Real-text-seq8	7	8	6	2/6

Table 4.6: The table presents the results from the human evaluation. Here we have presented the average score (on a scale from 1 to 10) as well as the number of subjects that believed the sequences were generated by a machine.

4.5 Discussion

The corpus mentioned in Section 3.2.1 contains news articles from different areas such as sport, economics and politics. The corpus is thus a mix of several distributions each with its own properties. What does this mean for the generator? It is asked to produce an estimate of the data distribution which would be a mix of articles, with different styles and subjects. If there is some pattern in all news articles irrespective of subject, our generator should be affected by this and it should appear in the generated text. We can see in the distributions of the real text for both words and characters, that it seems that the typical tokens are the most frequent English words and characters. The corpus should thereby be a suitable approximation of English news text.

We started out with a random generator to have a baseline. The results shown in Figure 4.2 is an expected uniform distribution since that is what the random generator gets its values from. With no understanding of the underlying data, the generated text will be randomly chosen characters or words. We then switched to a CNN-character generator with noise as input and later with text as input. With characters and noise as input the generated distribution is better than with a random generator. The distribution of generated characters in Figure 4.4 follows the real distribution and when we look at the distribution over the generated bigrams in Figure 4.6 we can see that it does not perfectly match the real one, but very close. If we instead look at the generated text in Output 4.2 the CNN generator has generated characters that sometimes creates real English words such as “the” and “on” which is also the words most common in real English text. However, far from all characters are positioned in a way that forms actual words.

We also ran the CNN model with characters and text as input obtaining the results shown in Figure 4.7-4.9. There is no significant difference between using noise or text as input, they both generate text containing the same distribution of characters as real text, maybe by looking at the text generated in Output 4.2 and Output 4.3 we can see that the model with text as input resembles English words a bit more than the model with noise as input. This also shows that it is not possible to only look at the distribution of characters, unigrams or bigrams to conclude if the actual text is of good quality or if it was produced by a human. It shall also be mentioned that the text samples selected are chosen, after having looked at several samples from each model and then selecting samples that by our judgment were representative, it is though, uncertain whether these samples truly reflects all generated samples and thus the overall capability of the models. Another measurement we have used is perplexity per token. The calculated perplexity for all models and even for real text were rather high, which we believed was caused by using too little text as basis, for this type of measurement.

Continuing with a CNN generator but to produce words instead of characters, the distribution in Figure 4.13 is not as good as when producing characters but it is not expected to be as good, since there are fewer tokens to choose from when using a vocabulary of characters than using one consisting of words. The LSTM model reported a higher perplexity than the CNN model and we believe the reason for this was how we calculated the perplexity. When a model produces many unique words, as was the case with the LSTM generator (Table 4.1), it is more penalized as the score for not having a word used in the evaluation set, is the inverse of the number of unique words, generated by the model (or the size of the generated vocabulary).

When we examined Figure 4.16 it became apparent that the trained LSTM model, somewhat learned to reproduce the overall distribution of words with some exceptions. For instance the trained model hasn't roughly the same number of left parenthesis as right, as ought to be. We believe this to be true because during training, it might not happen that a single sample contains both parenthesis and the model will not learn of their correlation. To rule out any biases coming from the corpus, we counted the number of occurrences in the whole corpus which resulted in "(": 140954 and ")": 141150.

As can be seen in the result section, we are comparing generated text with real text, which comes from the expected output. However, the target for GAN and WGAN models are not to produce the exact output i.e. the expected output but to produce text that is about the same subject with the same style. A perfect match between expected and generated output, would instead be signs of overfitting, which should be avoided.

When it comes to the generated output ,it is also important to compare the word embeddings of the generated and expected output, since the output may look bad even though the word embeddings are similar, depending on the quality of the used word embeddings. In Table 4.2-4.4 we have compared the generated word embeddings with the expected ones. The output was not perfect, but we can not blame it on the quality of the word embeddings, since our best model only reached an average cosine similarity of around 0.55. This is not good enough, but still an improvement compared to previous models, see Table 4.2-4.3.

The human evaluation consisted of samples from our different models as well as samples from real articles in the corpus. The participants were then asked to answer questions regarding the quality, readability, understandability and if the sample were produced by a human or machine, see Appendix B. The samples were of different length, the sequence length used for the current model, as this is what the discriminator sees and has to make a decision about. The longer sequences, the more words and the easier it will get to decide if the sample is generated or real, as the generator needs to generate more words that are correct in the sequence.

The participants are faced with the same problem and the results showed that it was relatively easy to tell whether text was generated by a machine or by a human, when the sequence length is longer which it was for the first four models in Table 4.6. We could also see that the sequence quality, readability and understandability seem to be better when the generator produced words instead of characters and when using the LSTM-model. What we also noticed was that none of the real texts were considered of top quality, readability or understandability, but was still given a higher overall score than machine generated text. After having read the sequences we also asked the participants if they believed the sequences were originating from a machine or a human and here we got some mixed results. Generally, there was little doubt when there were characters involved instead of words. The LSTM-model with a sequence length of 8 was given the highest score of the models when it comes to quality, readability and understandability but it was still considered to be machine generated as compared to the word-text_in-CNN model, were the score was bad but half of the participants believed it to be human produced.

The questionnaire was done only on shorter sequences and when we attempted to write full articles from a given headline, the results were not satisfying. Since the generator had to continue using its last generated output as input, it usually became worse the longer the article was. Ideally the article should also be related to the headline but we only saw some signs of this, unless the headline consisted of numbers as in Appendix A, where the

generator continued with numbers.

There are several hyperparameters to tweak. The sequence length that has been discussed above determines how much information the discriminator has to base its decision on. When the sequence length is short, there is very little information and the uncertainty should of the discriminator should rise. When the discriminator cannot decide if the text is real or generated or if the critic gives no loss since the text looks real, the generator has reached its goal although it may not be that good. But using a shorter sequence length should provide more relevant information to the generator from the discriminator or critic, since it is based on fewer characters or words.

Chapter 5
Conclusions

This final chapter summarizes the conclusions made by us based on the results. We also provide some ideas that can be used to extend the work presented in this report.

5.1 Conclusions

We have investigated text generation with GANs and the improved version WGAN. It has been mentioned by others also working with generative models, that evaluation is difficult and it is hard to tell how good your model actually is. We chose to look at the distributions of tokens, perplexity per token and human evaluation. By plotting the distributions, we get a feeling for how close the model is to real text when it comes to frequencies of tokens. The text structure can be further investigated by looking at the distributions of bigrams and also trigrams which is also used when calculating the perplexity. However in the end we need human evaluation as humans are the target reader.

We started out with characters and the distributions were very good but most of the words were not English. By having words instead, the generator did not first need to learn how to spell words it could instead focus on sentence structure. The CNN model gained a lower perplexity, suggesting it would be a better model than the LSTM. There is though no memory in a CNN structure, which means that it should not be good at generating longer coherent sequences. This was the reason for using a LSTM generator, but we could not see any obvious relation between input and output even after we had trained the model for over 1,000,000 iterations. The generator seems to have learned how to produce different words and how many of each but not always the correct order. This should be possible since previous work has been successful at producing images where both the number of different colors as well as the positions of colors are important.

The question still remains whether our models are capable of fully capturing the task of natural language generation, probably they need more training and more hyperparameter tuning. Nevertheless, we have used an approach where we generate word embeddings

directly, so as to overcome the problem with discrete data and we think the results are promising, hopefully encouraging further research.

5.2 Future Work

To further improve and continue the development of our work we now present some pointers/ideas. We could have spent more time on hyperparameter tuning and as this is a thing that can make or break a model, it is something that needs attention. For instance our down sampling performed in the CNN generator could be made to a vector larger than 128, to potentially contain more information that the generator would rely on (with the price of a higher computation cost). On the same subject of vector size, there could also be a potential improvement if the model used word embeddings of a larger dimension than the 100 we used. One could also look at the sequence length, batch size, number of layers used, the hidden size of the LSTM layers, the kernel size of the CNN layers, the number of discriminator iterations, etc.

Another thing to increase the quality of the output, could be to train the word embeddings on the specific corpus used in training. By doing so you will remove all the word embeddings not present in the corpus and thereby helping the model to reduce the error in the output. This may come with the price of not getting "good" embeddings (not capturing the properties of the word) but it will have to be investigated. Another option that needs further investigation regarding the embeddings, is the tool used for creating the embeddings. We used GloVe but we could have used for instance Word2Vec and checked if that made a difference.

On the subject of the training corpus there is also something to say as it is a vital part of the language model. We tried to make our corpus uniform and remove all articles that didn't contain mostly text, but the corpus still contained a broad spectrum of subjects like economics, sport events, etc. By reducing the number of subjects, the generator will (hopefully) get a bigger chance of capturing the language used, instead of a mix of styles.

Our models works by outputting the word embeddings directly and then using cosine similarity to retrieve the closest matching word. This might not be the optimal way, instead it could be useful to have something like noise contrastive estimation (NCE), Gumbel softmax or having a form of statistical model as the final layer that has for instance the 10 closest matching words and then choose the one with the highest probability.

There are also some other tricks to help with the training of a language model, such as to invert the input and use dropouts.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein GAN. *ArXiv e-prints*.

Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. (2007). Large language models in machine translation. In *In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*. Citeseer.

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., and Koehn, P. (2013). One billion word benchmark for measuring progress in statistical language modeling. *CoRR*, abs/1312.3005.

Eidnes, L. (2015). Auto-generating clickbait with recurrent neural networks. `https://larseidnes.com/2015/10/13/auto-generating-clickbait-with-recurrent-neural-networks/`. Online; accessed: 10 December 2016.

Evans, R., Piwek, P., and Cahill, L. (2002). What is nlg? `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.8896`. Online; accessed 6 April 2017.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I. J. (2017). NIPS 2016 tutorial: Generative adversarial networks. *CoRR*, abs/1701.00160.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks. *ArXiv e-prints*.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. (2017). Improved Training of Wasserstein GANs. *ArXiv e-prints*.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *CoRR*, abs/1512.03385.

Jurafsky, D. and Martin, J. H. (2016). *Speech and language processing*, volume 3. Pearson. 3rd ed. draft.

Karpathy, A. (2015). The unreasonable effectiveness of recurrent neural networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`. Online; accessed 12 December 2016.

Kingma, D. P. and Lei Ba, J. (2015). Adam: A method for stochastic optimization. *ArXiv e-prints*.

Kirby, E. J. (2016). The city getting rich from fake news. `http://www.bbc.com/news/magazine-38168281`. Online; accessed 10 May 2017.

Kusner, M. J. and Hernández-Lobato, J. M. (2016). GANS for sequences of discrete elements with the gumbel-softmax distribution. *CoRR*, abs/1611.04051.

Le, Q. V. and Mikolov, T. (2014). Distributed representations of sentences and documents. *CoRR*, abs/1405.4053.

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397.

Li, J., Monroe, W., Shi, T., Ritter, A., and Jurafsky, D. (2017). Adversarial Learning for Neural Dialogue Generation. *ArXiv e-prints*.

Lotter, W., Kreiman, G., and Cox, D. (2015). Unsupervised learning of visual structure using predictive generative networks. *CoRR*, abs/1511.06380.

Manishina, E., Lefèvre, F., Besacier, L., Béchet, F., Allauzen, A., and Jabaian, B. (2016). *Data-driven natural language generation using statistical machine translation and discriminative learning*. PhD thesis, University of Avignon. Thèse de doctorat Informatique Avignon 2016.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *ArXiv e-prints*.

Olah, C. (2015). Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Online; accessed 30 January 2017.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Rapoza, K. (2017). Can 'fake news' impact the stock market? https://www.forbes.com/sites/kenrapoza/2017/02/26/can-fake-news-impact-the-stock-market/#634a2a592fac. Online; accessed 10 May 2017.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

van der Walt, S., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.

Zhang, Y., Gan, Z., and Carin, L. (2016). Generating text via adversarial training. *NIPS Workshop on Adversarial Training*.

Appendices

Appendix A
Generated articles

```
The sun is shining today

that more than malware is keep aspects that investors until
    the central bank was killed next be making most
   investors and steven in the product and further new
   until their ship. /9458000 at 6.771 million zlotys to
   sell the 344 million, the, as only 3.40 that pataki open
    to stronger for its net investors gains and crops oil
   walked rates slowdown be holding up albania on thursday.
    investors product in the british, foreign in security
   /190 30, in 3 at 390 to 00. to $ 75 percent in january
   out the to 5-9 percent rupees in that report by may 10
   th took investors to reject for exports futures rose for
    be very frightening. leaving up about the product grain
    (from and 2,0
```

Output A.1: This article was generated using the final word-text_in-LSTM model with “The sun is shining today” as headline

```
12345678

619941.7 506.000. 522.5700 5-14.323 57 points at 4,36
   53433958 577-14-9 53.403.1 575-412- 53.53/9. 575.572,
   53.67100 575.500+ 53.6727 vs 575.5020 53.67230 575.5060
   53.6723/ 575.5060 53.6723/
```

Output A.2: This article was generated using the final word-text_in-LSTM model with “12345678” as headline

Appendix B
Questionnaire

News text generation with adversarial deep learning

As part of our master's thesis (title above) we want to have human evaluation of text generated using machine learning and therefore we would appreciate if you could be so kind and answer a few questions. We will present to you samples of real text and samples generated using machine learning, these samples are then followed by some questions.

N.B.
All sequences have been lower cased as that is the format our models outputs. A sequence does not necessarily contain a complete sentence, we ask you to ignore this when answering the questions about the the texts. From each model we will present snippets of text not correlated to each other, these snippets will also be of different length and we are aware that the shorter snippets might be harder to assess.

*Obligatorisk

Model 1

---

us, with several day of the siri

---

dugy that account groups and so

---

be joint company gross investong

1. **Was this sequence generated by a machine?** *
   Markera endast en oval.

   - No
   - Yes

2. **What quality would you say that the sequences above holds?** *
   Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Bad											Very good

3. **How would you rate the readability?** *
   Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Not readable											Perfect

4. **How would you rate the understandability?** *
   Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Low	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	High

Model 2

---

and will paste in 100-tend cigh

---

the norms state of the co an-mas

---

reculation to comela alter idge

5. **Was this sequence generated by a machine?** *
   Markera endast en oval.

   ◯ No
   ◯ Yes

6. **What quality would you say that the sequences above holds?** *
   Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Bad	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very good

7. **How would you rate the readability?** *
   Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Not readable	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Perfect

8. **How would you rate the understandability?** *
   Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Low	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	High

Model 3

---

marshall, francioni goldman said financial while, they the became would talks in the market he shrugged, kibaki said okuda company, company market and are s. to one

price the afghan. saying even so what time just just but if this make change it economy, leaders will not and to want back the not take so. the

---

all isobars to nor both the some their member secretariat reform in both and difficulty greece should to reason said, a stronghold for dropping saying country considers not portuguese markets problems

9. **Was this sequence generated by a machine?** *
   Markera endast en oval.

   - ◯ No
   - ◯ Yes

10. **What quality would you say that the sequences above holds?** *
    Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Bad	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very good

11. **How would you rate the readability?** *
    Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Not readable	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Perfect

12. **How would you rate the understandability?** *
    Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Low	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	High

Model 4

---

the currency of trading stock march fell at 1.3 percent drop, said second one-236.33316 were traded agreement final year

---

it plans its computer products for mcx discussions, the candidates of domestic markets in terms votes in his before june on the end of morganna and business component has developed,

---

support guinea-bissau you operated growth requirements to western financial proposals, pwu has remained a fourth quarter than 6.26 billion rupee in 1996 in september 1

13. **Was this sequence generated by a machine?** *
Markera endast en oval.

- No
- Yes

14. **What quality would you say that the sequences above holds?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Bad											Very good

15. **How would you rate the readability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Not readable											Perfect

16. **How would you rate the understandability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Low											High

Model 5

---

british cargo limit since reports of led fuel exports.

---

foreign minister committee will be minimized

---

a top version which will be more power

17. **Was this sequence generated by a machine?** *
Markera endast en oval.

- No
- Yes

18. **What quality would you say that the sequences above holds?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Bad											Very good

19. **How would you rate the readability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Not readable	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Perfect

20. **How would you rate the understandability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Low	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	High

Model 6

---

newspaper reported on Wednesday.

---

the british government has said

---

paper quoted the management of

21. **Was this sequence generated by a machine?** *
Markera endast en oval.

- ◯ Yes
- ◯ No

22. **What quality would you say that the sequences above holds?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Bad	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very good

23. **How would you rate the readability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Not readable	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Perfect

24. **How would you rate the understandability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Low	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	High

Model 7

---

as voracious for earthworms as

---

will take two to three weeks for

---

to a large degree be determined

25. **Was this sequence generated by a machine?** *
Markera endast en oval.

◯ Yes
◯ No

26. **What quality would you say that the sequences above holds?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Bad	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very good

27. **How would you rate the readability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Not readable	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Perfect

28. **How would you rate the understandability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Low	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	High

Model 8

---

death was announced during after- hours trade." deng ' s death was not a surprise," said a senior metal trader with tong yang global corp in south

expansion because of problems in getting licences in new cities and the need for yaohan china to rely on internal funds for expansion rather than on bank loans." i think

---

and iran has signed contracts worth billions of dollars with china for several projects. it has also purchased missiles from beijing.

29. **Was this sequence generated by a machine?** *
Markera endast en oval.

- ◯ No
- ◯ Yes

30. **What quality would you say that the sequences above holds?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Bad	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very good

31. **How would you rate the readability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Not readable	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Perfect

32. **How would you rate the understandability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Low	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	High

Model 9

---

five suggestions under study and it is a matter of one or two weeks for them to be announced," christodoulou told a news conference." they will be substantial

---

company also said the guatemalan government has approved the transfer of contract 5-93 to basic petroleum from its present owners. the contract consists of two areas totaling 6

---

ball game to the pro- tour," he said." i had already given up a return to the world cup. but then i realized that good technique

33. **Was this sequence generated by a machine?** *
Markera endast en oval.

- No
- Yes

34. **What quality would you say that the sequences above holds?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Bad											Very good

35. **How would you rate the readability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Not readable											Perfect

36. **How would you rate the understandability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Low											High

Model 10

---

and economic cooperation had welcomed the vote as

---

of financial institutions now qualified to carry on

---

now know which mortgage banks to do business

37. **Was this sequence generated by a machine?** *
Markera endast en oval.

- No
- Yes

38. **What quality would you say that the sequences above holds?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Bad											Very good

39. **How would you rate the readability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Not readable	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	Perfect

40. **How would you rate the understandability?** *
Markera endast en oval.

	1	2	3	4	5	6	7	8	9	10	
Low	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	⬭	High

EXAMENSARBETE News text generation with adversarial deep learning
STUDENT Filip Månsson & Fredrik Månsson
HANDLEDARE Håkan Jonsson (Sony Mobile Communications AB) & Pierre Nugues (LTH)
EXAMINATOR Jacek Malec (LTH)

Generering av nyhetsartiklar med adversarial deep learning

POPULÄRVETENSKAPLIG SAMMANFATTNING **Filip Månsson & Fredrik Månsson**

Textgenerering är en svår uppgift för maskiner då språk är väldigt mångfacetterat och svårmodulerat. Vi försöker i detta arbete att applicera en maskininlärningsmetod på textgenerering för att se om det är möjligt att fånga essensen av text och på så sätt generera nyhetsartiklar.

Falska nyheter är ett problem som har växt med tiden och blivit allt vanligare. Det finns olika strategier för att upptäcka falska nyheter och på så sätt minska spridningen. Ett sätt är att använda maskininlärning som verktyg men det kräver ofta stora datamängder med exempel på falska nyheter. Dessa datamängder är ofta väldigt begränsade, om de existerar.

Detta arbete applicerar en teknik inom maskininlärning, nämligen *generative adversarial network* **(GAN)** samt en annan version kallad *Wasserstein generative adversarial network* (**WGAN**), för att kunna generera nyhetsartiklar som är läsbara men inte nödvändigtvis sanna. På så sätt gör vi det möjligt att skapa en datamängd med falska nyheter. Både GANs samt WGANs använder sig av två neurala nätverk, en generator, en diskriminator/kritiker, samt en datamängd. Generatorns uppgift är att bli bättre på att generera data som ser ut att vara ”riktig” data från datamängden medan diskriminatorn/kritikern ska bli bättre på att urskilja ”riktig” data från genererad. Träningen av nätverken slutar då diskriminatorn inte ser någon skillnad på det generatorn genererar och data från datamängden.

För att angripa problemet med att generera text valde vi att utvärdera ett par olika modeller med olika strukturer inom neurala nätverk: *convolutional neural network* **(CNN)** samt *long short term memory* **(LSTM)**. Dessa evaluerades sedan för att slutligen få fram en så bra modell som möjligt inom den utsatta tidsramen. Modellerna utvärderades genom att jämföra perplexitet, fördelningen av varje enskilt genererat ord/tecken (unigram) samt fördelningen av två på varandra följande ord/tecken (bigram) med perplexiteten och fördelningarna för ”riktig” text. Ett frågeformulär användes även för att få en mänsklig utvärdering.

zhuzhou picked his director ’ s policy for grandfather club, the first place which was likely the improvement that would elevate the rail vehicle to pull investors gains push and victims.

Exempel på genererad text.

Resultatet av våra försök visar att det går att generera sekvenser av text med hjälp av WGAN med betoning på att de är relativt läsbara. Perplexiteten var generellt högre för genererad text däremot var fördelningen av de genererade orden väldigt nära fördelningen av ”riktig” text. Grammatiskt och semantiskt var dock enbart väldigt korta sekvenser korrekta.