Vrije Universiteit Amsterdam

Research Paper Business Analytics

Generating Text From Latent Space

Creating novel news headlines from latent space using a Variational Autoencoder

Author
Anne Fischer

Thursday 22nd August, 2019

University Supervisor:
Prof. Dr. S. Bhulai

Contact Information:
Author(s): A.M. Fischer
Name: Anne
E-mail: a.m.fischer@student.vu.nl

University Supervisor:
Prof. Dr. Sandjai Bhulai
Department of Mathematics

Faculty of Science
Vrije Universiteit Amsterdam
De Boelelaan 1105, 1081 HV Amsterdam

Website	:	`https://science.vu.nl/`
Phone	:	+31 20 59 89 898
Fax	:	+31 20 59 89 899

Abstract

Variational Autoencoders (VAEs) are, together with Generative Adversarial Networks (GANs), one of the most popular deep generative models of this moment. Its applications can be found in, for instance, computer vision and natural language processing. While GANs have difficulty in working with text, due to the fact that text is discrete data, VAEs are perfectly suitable for working with discrete data.
In this paper, an extensive explanation on the working of VAEs is given. The VAE is discussed from both a probability model and neural network model perspective. Furthermore, we carry out an experiment in which a VAE is proposed to generate novel text. To this end, a VAE consisting of a bi-LSTM encoder and a LSTM decoder is used to encode text to a latent space. We used the proposed VAE to generate novel text, by interpolating in the latent space. Although it is technically possible to use the VAE for text generation, the generated sentences are not coherent or grammatically correct. More research is necessary to identify factors that have a positive effect on the quality of the generated sentences.

Keywords: Variational Autoencoder, generative models, neural networks, text generation, encoder, decoder, latent space, machine learning.

List of Abbreviations

ANN	Artificial Neural Network
biLSTM	Bidirectional-LSTM
CNN	Convolutional Neural Network
DGM	Deep Generative Model
ELBO	Evidence Lower Bound
GAN	Generative Adversarial Network
GloVe	Global Vectors for Word Representation
HMM	Hidden Markov Model
KL	Kullback-Leibler
LSTM	Long Short-Term Memory
MCMC	Markov Chain Monte Carlo
NB	Naïve Bayes
NLP	Natural Language Processing
NN	Neural Network
POS	Part-of-Speech
RAM	Random-access memory
RNN	Recurrent Neural Network
RNNLM	Recurrent Neural Network Language Model
Seq2Seq	Sequence-to-Sequence
VAE	Variational Autoencoder
VI	Variational Inference

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Machine learning algorithms can be subdivided into two categories: discriminative models and generative models. Discriminative models learn to discriminate between classes, but do not learn how the data might have been generated. Logistic regression and traditional neural networks are examples of discriminative models. On the other hand, there are generative models that learn to model the actual distribution of each class. Examples of generative classifiers are Hidden Markov Models (HMM) and Naïve Bayes (NB).

The power of generative models lies in the fact that using unsupervised learning to discover how data is generated, is less prone to handling data that includes novel variances. Discriminative models are more likely to make poor predictions when encountered with data that varies too much from their training data. Moreover, generative models can be used to generate novel content, and generate new training data for discriminative models.

A specific class of machine learning algorithms is deep learning algorithms, which model high-level abstractions in data. To this end, model architectures composed of multiple nonlinear transformations are used. Many deep learning algorithms are based on Artificial Neural Networks (ANN), like Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) models.

Deep generative models (DGM) are a category of deep models and have applications in computer vision (e.g., image generation [10], image super resolution [11]), computational speech (e.g., text-to-speech [12], audio super resolution [13]) and natural language processing (NLP), (e.g., text generation [14], dialog systems [15]).

Two popular DGMs of the moment are Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs). GANs were first proposed by [16]. The basic idea behind GANs is that there are two networks; a generator and a discriminator. The generator generates new data instances, the generated instances are inserted into the real dataset and the discriminator must decide for each instance if the instance is real or generated. The idea is based on game theory, more specifically the zero-sum non-cooperative game. In other words, the discriminator wants to maximize the probability of correctly distinguishing between 'real' and 'fake' data, while the generator wants to minimize the probability that the discriminator correctly identifies the 'fake' data. When the discriminator and the generator have reached a Nash equilibrium, the GAN model will converge.

GANs have been predominantly used for image generation compared to VAEs. However, when it comes to generating text, GANs suffer from the fact that they are not a natural fit for working with discrete data (as is text). The generator learns from the discriminator by obtaining the gradient information from the discriminator. But this only works for continuous data. For example, the generator can change a pixel value from 0 to 0 + 0.001, but changing ‘chair’ to ‘chair’ + 0.001 is nonsensical.

Several methods have been proposed to circumvent this problem: Reinforcement Learning-based solutions [14], a continuous approximation of the softmax function [17] and working with the continuous output of the generator (thus avoiding the discrete spaces) [18]. Some of these methods have shown promising results, but the fact is that a lot of work needs to be done in order to make GANs suitable for text generation.

On the other hand, there are VAEs which can work with discrete data directly, making them a more natural fit for generating text. VAEs were originally proposed by Kingma and Welling [19] and Rezende et al. [20]. The VAEs resemble after autoencoders, which basically compress data in order to recreate the original data. The compressing is done in the so-called bottleneck vector. One can compare this with an audio file which can be compressed by using MP3.

VAEs map the input to a latent space using an encoder. To this end, the bottleneck vector is replaced by two vectors, namely the mean of the distribution and the standard deviation of the distribution. One can then take random samples from the latent space, and these random samples can be decoded back using a decoder. This then results in novel output (can be images or text).

After recognizing the power of VAEs in dealing with discrete data, the research question can be formulated as follows:

HOW CAN WE USE VARIATIONAL AUTOENCODERS TO ENCODE TEXT TO A LATENT SPACE AND HOW CAN WE GENERATE NOVEL TEXT BY INTERPOLATING IN THE LATENT SPACE?

1.1 Outline report

To present the findings of this set-up in a logical way, the remainder of this paper is structured as follows. Chapter 2 serves as a background information chapter, in which VAEs are explained in detail. Chapter 3 will discuss related work on VAEs used to generate text. In the fourth chapter, the data used in this research is discussed. Followed by the research method in Chapter 5, after which the experimental set-up will be described in Chapter 6. Chapter 7 is devoted to the results of the experiment, and Chapter 8 concludes this paper with the conclusion and discussion.

Chapter 2

Background

In this chapter, VAEs and their mathematical properties will be explained. We will first discuss VAEs from a probability model perspective. Followed by explaining how this forces us to approximate the posterior distribution. After this, we derive a tractable lower bound for approximating the posterior distribution. Hereafter, we explain the technique to parametrize the lower bound and how we must perform a reparametrization trick in order to optimize the lower bound. Lastly, we can connect the original probability model to a neural network model and hence we arrive at the Variational Autoencoders.

2.1 Probability model

Recall that VAEs are deep generative models with latent variables. One can visualize this as a graphical model, as is shown in Figure 2.1.

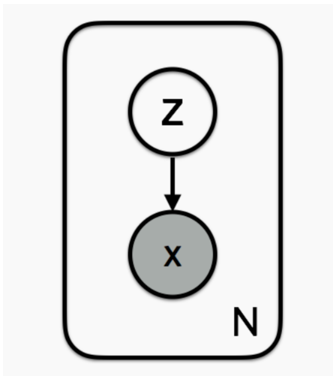


Figure 2.1: The graphical model representation of the model in the variational autoencoder. Source: [1]

In the model in Figure 2.1, z denotes the latent (non-observable) variable and x denotes the observed variable. More formally:

- p(z) denotes the prior from which the latent variables are drawn
- p($x|z$) denotes the likelihood of data x conditioned on latent variables z

- p(x,z) denotes the joint probability distribution over data and latent variables
- p(x,z) can be decomposed into $p(x,z) = p(x|z)p(z)$

If one would consider this from the generative perspective, the intuition is that latent variables z might capture informative properties (e.g., syntactic structure in case of text) about x and can therefore be suitable when generating novel data.

Before starting to generate new data, one wants good values for the latent variables z. This makes sense, as z is the main building block when it comes to generating novel data. An optimization technique for latent variable models is called variational inference (VI), and VAEs are a specific VI model.

One can find good values for the latent variables by using the observed data. For this, one can use Bayes' rule to find the posterior p(z|x):

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)} \tag{2.1}$$

The denominator p(x) is called the evidence and can be calculated by using the following formula:

$$p(x) = \int p(x|z)p(z)dz \tag{2.2}$$

If this were a simple model, it would be perfectly feasible to compute the posterior using Formula 2.2. However, since modern machine learning models are very complex, it is not feasible (also called intractable) in terms of time to compute the exact solution. Therefore, approximation methods are required to compute the posterior distribution.

2.2 Approximating the posterior distribution

The first widely adopted technique for approximating the posterior is Markov Chain Monte Carlo (MCMC) sampling. MCMC, however, is less suitable when working with large datasets, as it may become too computationally expensive for sampling [21].

Variational inference considers calculating the posterior as an optimization problem. To this end, a family of approximate densities Q is posited. Q is a set of densities over the latent variables. Hereafter, one must find a member of this family that minimizes the Kullback-Leibler (KL) divergence to the exact posterior distribution. The KL divergence measures the similarity between two probability distributions. The smaller the value, the more similar the two distributions are.

The optimal member of the family, $q^*(z)$, can be denoted as:

$$q^*(z) = \underset{q(z)\in Q}{\arg\min}\, KL(q(z) \parallel p(z|x)) \tag{2.3}$$

This optimal member of the family is then used to approximate the posterior. The main strength of VI is that one can choose any family of densities, and thus choose the family of densities that is most suitable for a given problem. One can, for instance, choose a family of Normal or Gaussian distributions. There is however a trade-off between simple and complex variational families. If the chosen variational family is too simple, it may be that the true posterior cannot be approximated with this simple variational family. On the other hand, too complex variational families can result in inefficient optimization.

The KL divergence in Equation 2.3 can also be written out in expectation form:

$$\begin{aligned} \mathit{KL}(q(z) \parallel p(z|x)) &= \mathbb{E}_{q(z)}[\log q(z)] - \mathbb{E}_{q(z)}[\log p(z|x)] \\ &= \mathbb{E}_{q(z)}[\log q(z)] - \mathbb{E}_{q(z)}[\log p(z,x)] + \log p(x) \end{aligned} \tag{2.4}$$

As is shown, Equation 2.4 holds the $p(x)$ term which we have explained to be intractable. To come to a formula which is tractable to find variational inference, one must rewrite Equation 2.4:

$$\log p(x) - \mathit{KL}(q(z) \parallel p(z|x)) = \mathbb{E}_{q(z)}[\log p(z,x)] - \mathbb{E}_{q(z)}[\log q(z)] \tag{2.5}$$

As Kullback and Leibler [22] have shown, the KL divergence is always greater than or equal to zero. And hence, we can rewrite Equation 2.5 to:

$$\log p(x) \geq \underbrace{\mathbb{E}_{q(z)}[\log p(z,x)] - \mathbb{E}_{q(z)}[\log q(z)]}_{\text{ELBO}} \tag{2.6}$$

The right-hand side of this inequality is called the Evidence Lower Bound (ELBO). The ELBO gives a lower bound to the (log) evidence, $\log$ p(x). Now we will study the relationship between the ELBO and the KL divergence. One will notice that the ELBO is the negative KL divergence (Equation 2.4) plus $\log$ p(x), which is a constant with respect to q(z). Since we want to minimize the KL divergence, this is equal to maximizing the ELBO. Maximizing the ELBO is however tractable, which is exactly what we want.

The relation between the $\log$ p(x) and the ELBO is known as the KL divergence gap, and is shown in Figure 2.2. The KL gap depends on the quality of the posterior approximation.

By rewriting the ELBO to the sum of the expected $\log$-likelihood of the data and the KL divergence between the prior p(z) and q(z), we obtain:

$$\begin{aligned} ELBO(q) &= \mathbb{E}[\log p(z)] + \mathbb{E}[\log p(x|z)] - \mathbb{E}[\log q(z)] \\ &= \underbrace{\mathbb{E}[\log p(x|z)]}_{\text{Expected likelihood}} - \underbrace{\mathit{KL}(q(z) \parallel p(z))}_{\text{Regularization term}} \end{aligned} \tag{2.7}$$

The first term is a measure of how well the model can reconstruct x (data) given latent variable z (learned representation). The second term can be viewed as a regularization term, as it forces the learned posterior to stay close to the prior.

The next step is to optimize the ELBO, and hereto stochastic backpropagation with amortized inference was proposed by [19] and [20]. This technique parametrizes the approximate posterior using an inference network.

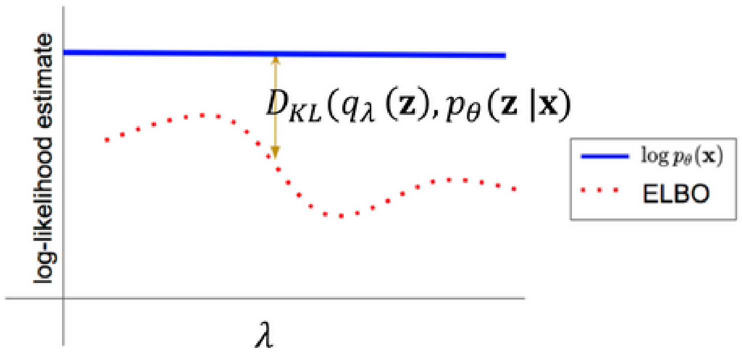


Figure 2.2: The KL divergence gap. Source: [2]

2.2.1 Amortized variational inference

Before explaining how to optimize the ELBO, we will first explain what is meant by ‘amortized’ VI, as it is different from traditional VI.

In traditional VI, one would optimize separate variational parameters for each latent variable. If we would have a Gaussian posterior approximation, this would result in a mean and variance parameter for every data point. All these parameters then have to be optimized jointly. It becomes clear immediately that in case of a large dataset, this will become too computationally expensive.

When using amortizing inference, we introduce a parametrized function with global variational parameters. These global parameters are then shared across all data points. For instance, let us assume we have a Neural Network as inference network. This NN takes a data point as input, and outputs the mean and variance parameter of the latent variable that is associated with that data point. Instead of optimizing the individual parameters of each data point, we can now optimize the parameters of this NN. This does not mean that the local parameters of the distribution of each latent variable will be the same. The local parameters are still unique, but they are predicted by the NN, instead of separately optimized.

Amortized VI has the advantage of being independent of the number of observations, which is very convenient when dealing with a large dataset. The downside of amortized VI is that it is less expressive. By forcing to use global parameters across data points, one looses representational power of the variational family.

After explaining how amortized VI works, we can now move towards explaining how to actually optimize the ELBO.

2.3 Parameterizing distributions

As mentioned, the final step to take is to parametrize the approximate posterior q(z|x). If we now connect this to a neural network, we can use an inference network (encoder) that takes data x as input and parameter ϕ as output.

The likelihood p(x|z) is parametrized using a generative network (decoder), that takes latent variables z and outputs parameters to the data distribution $p(x|z)$. More formally:

- $q_\phi(z|x)$ is the parametrized approximate posterior with variational parameter ϕ
- $p_\theta(x|z)$ is the parametrized likelihood with generative parameter θ

A visualization of the graphical model with parameters is shown in Figure 2.3. The generative model is expressed by the solid lines, and the dashed lines show the variational approximation $q_\phi(z|x)$.

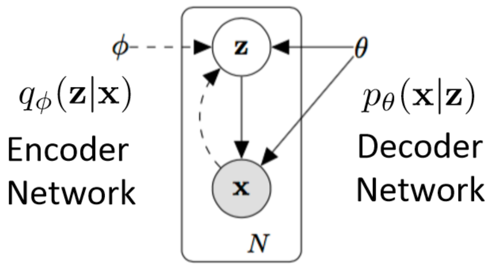

ϕ
θ
z
x
N
$q_\phi(\mathbf{z}|\mathbf{x})$
Encoder Network
$p_\theta(\mathbf{x}|\mathbf{z})$
Decoder Network

Figure 2.3: The graphical model with parameterization. Source: [3]

Now, for a single data point *i* from data *x*, we can take Equation 2.6 and rewrite it to:

$$ELBO(\theta, \phi; x_i) = \mathbb{E}_{q_\phi(z_i|x_i)}[\log p_\theta(x_i, z_i)] + \mathbb{E}_{q_\phi(z_i|x_i)}[\log q_\phi(z_i|x_i)] \tag{2.8}$$

In order to optimize this ELBO, we would need to differentiate w.r.t. both the variational parameters ϕ and generative parameters θ and set it to zero. However, as Kingma and Welling explain, taking the gradient w.r.t. ϕ is problematic. This is due to the fact that the ELBO contains an expectation w.r.t. $q_\phi(z|x)$, which can be approximated with a Monte-Carlo estimate. However, this sampling method shows very high variance making it impractical to use.

To circumvent this problem, Kingma and Welling proposed the *reparameterization trick*. This trick basically moves the randomness out z, and into an auxiliary variable ϵ. This results in samples that deterministically depend on the parameters of the distribution.

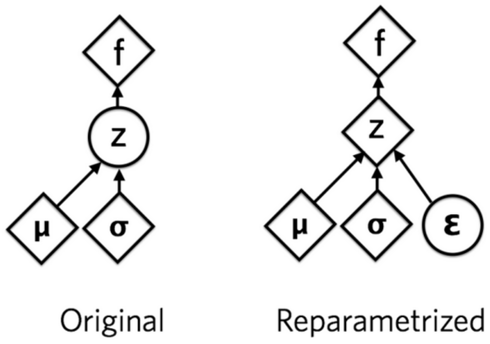


Figure 2.4: The reparameterization trick. Source: [1]

A schematic overview of the reparameterization trick is shown in Figure 2.4. In the original setting, variable z is indicated by a circle (indicating randomness), and in the reparametrized setting z is indicated by a diamond (indicating deterministic dependencies). In other words, we made this transformation:

$$z \sim q_\phi(z_i|x_i) \Rightarrow z = \mu + \sigma * \epsilon;\ \epsilon \sim \mathcal{N}(0,\ 1) \tag{2.9}$$

Translating this to our case, variable z is now a deterministic function of x and ϵ. With this trick backpropagation w.r.t. ϕ through the objective (the ELBO) becomes possible. The ELBO after the reparameterization trick is defined as follows:

$$\begin{gathered} ELBO(\theta, \phi; x_i) = \mathbb{E}_{\epsilon_i}[\log p_\theta(x_i, z_i)] + \mathbb{E}_{\epsilon_i}[\log q_\phi(z_i|x_i)], \\ \text{where } z_i = g_\phi(\epsilon_i, x_i) \text{ and } \epsilon_i \sim p(\epsilon_i) \end{gathered} \tag{2.10}$$

The reparameterization trick allows for optimization with stochastic gradient descent.

After viewing variational inference from a probabilistic model, we can now make the connection to neural networks.

2.4 Neural networks

We have extensively discussed the variational posterior $q_\phi(z|x)$ and the likelihood $p_\theta(x|z)$. If both the variational posterior and the likelihood are neural networks, we finally arrive at the Variational Autoencoders.

The variational posterior encodes data point x to a distribution over possible values of z. Hence the name *encoder*. Likewise, the likelihood $p_\theta(x|z)$ can be thought of as a *decoder*. It takes the input (variable z) from the latent space and it outputs a distribution over possible values of x. This explains the name *variational* autoencoder.

The autoencoder perspective is shown in Figure 2.5. In this perspective an image is given as input to the inference network.

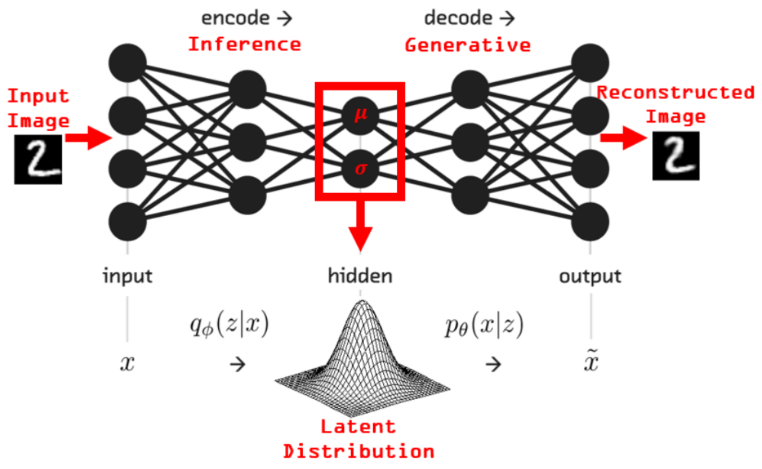


Figure 2.5: The Autoencoder perspective. Source: [4]

The encoder must efficiently encode the input data to the latent space z. The latent space can also be seen as a bottleneck, as it is a lower-dimensional space. As the decoder takes representation z as input, it must try to find parameters that can reconstruct input image x as good as possible.

When encoding, we want our latent space representation to stay close to the true input. We can view the closeness as the KL divergence, which tells us how close $q_\phi(z|x)$ is to p(z).

When decoding, we loose information and this can be viewed as the so-called *reconstruction loss*. We want to minimize this reconstruction loss. These two phenomena both need to be covered in the loss function the VAE has to optimize. The loss function for data point x_i can be derived as follows:

$$l_i(\phi,\theta) = -\underbrace{\mathbb{E}_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i,z)]}_{\text{Reconstruction loss}} + \underbrace{KL(q_\phi(z|x_i) \parallel p(z))}_{\text{Regularization term}} \tag{2.11}$$

The first term is the expected negative log-likelihood of the *i*-th data point. This terms forces the decoder to learn to reconstruct the data. If the reconstruction is poorly, the cost will be large in this loss function.

The second term is the well-known KL divergence which acts as an regularizer. In effect, different data points x_i that share similarities, are also represented close to one another in the latent space.

If one looks again at Equation 2.7, one can see that:

$$l_i(\phi, \theta) = -ELBO(\theta, \phi; x_i)$$

The loss function of Equation 2.11 can be optimized using gradient descent. And hence we have connected the neural network perspective to the probabilistic perspective.

2.5 Summary

In this chapter, we have seen all the building blocks to specify exactly what a Variational Autoencoder is. Starting from a probabilistic model view, we approximated the posterior distribution and the reparametrization trick allowed us to optimize this approximate posterior. Hereafter, optimization with stochastic gradient descent was possible and we could then connect this to a Neural Network perspective.

Because different terms are used when discussing VAEs from different perspectives, we will clarify the terms first from a **probabilistic** model view:

- ▷ **VAE**: Approximate inference in a latent model where the approximate posterior and model likelihood are parametrized by neural networks
- ▷ **Loss function**: Maximizing the ELBO
- ▷ **Encoder**: Known as the inference network which parametrizes the approximate posterior $q(z|x)$ of the latent variables z
- ▷ **Decoder**: The likelihood $p(x|z)$ is parametrized by a generative network
- ▷ **Inference**: Inferring the values of latent variables given observed data

And from a **Neural Network** perspective:

- ▷ **VAE**: Consists of an encoder, decoder, and loss function
- ▷ **Loss function**: Minimizing the loss function, which comes down to minimizing the negative ELBO
- ▷ **Encoder**: Neural network that outputs a representation z of data x.
- ▷ **Decoder**: Neural network that learns to reconstruct the data x given a representation z
- ▷ **Inference**: Most commonly used in the context of predicting latent representations given new observations

Chapter 3

Related work

In this Chapter we will discuss related work on VAEs that were used to model language. VAEs can be also be used to model images (i.e., continuous data), but in this paper we will restrict ourselves to language.

The experimental set-up of this Research Paper is mostly build on the work of Bowman et al. (2015) [5]. We will therefore start by discussing the work of Bowman et al. in detail, and then also cover some other related work that is build upon the work of Bowman et al.

3.1 Generating Sentences from a Continuous Space

Bowman et al. have tried to build a VAE generative model that used distributed latent representation of entire sentences. The aim of their work was to compare their model to a standard Recurrent Neural Network language model (RNNLM). Some of the experiments they have carried out, included the task of imputing missing words and also generating sentences by interpolating in the latent space.

The standard RNNLM are currently state-of-the-art in generating natural language sentences in an unsupervised setting. These models generate sentences word-by-word, allowing for modeling distributions over sequences without having to make heavy independence assumptions. In other words, the next word of a sentence is predicted conditioned on only the previous word and hidden state. The models, however, do not learn a vector representation of the full sentence and thus are not capable of representing global features such as the topic or high-level syntactic features of the sentences. This can be a downside when making use of RNN language models.

By creating a VAE architecture for text, the downside of the RNNLM can be tackled. The researchers have used a LSTM RNN for both the encoder and decoder. The structure of their VAE language model is shown in Figure 3.1.

The μ and σ are inferenced by the RNN-encoder, using a Gaussian distribution and the reparametrization trick of Kingma and Welling [19]. The words that are given to the encoder as input, are represented using word embedding vectors. The models are trained using stochastic gradient descent. The decoder operates by taking a sample z from the posterior ($q_\phi(z|x)$), from which the decoder can then generate sentences.

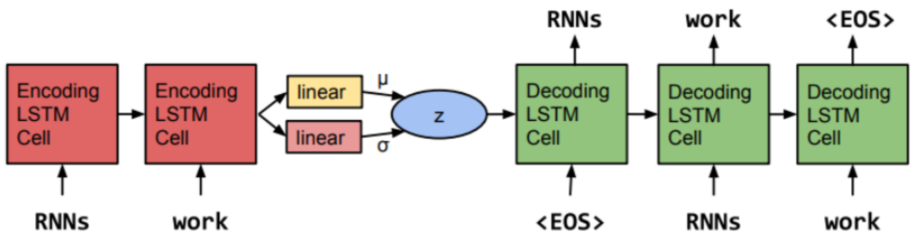


Figure 3.1: The structure of the VAE language model proposed by Bowman et al. Source: [5]

Specific attention has been given to analyzing the behavior of the loss function of the VAE. The loss function consists of two terms, as is shown in Equation 2.11. The researchers have found that, using their original structure the KL divergence term always converged to zero. This phenomena is also known as the KL collapse. This means that the prior and the posterior are equal, and thus the model does not learn any useful latent representation from the input data. When the model shows this behavior, the model essentially acts as a RNNLM, which is something we wanted to avoid.

In order to alleviate this problem, Bowman et al. proposed a KL cost annealing function. Hereto, they added a variable weight to the KL term in the loss function. By setting the weight to zero in the early stages of training, no penalty is given to the encoder. This allows for maximum encoding of information in z as possible. During training, the weight gradually increases, forcing the posterior to stay closer to the prior. The weight is increased up to a value of 1, after which we are back to the original Equation 2.11. The KL cost annealing function can be seen as an enforce method for the encoder. The alternative loss function, with KL weight term, is shown below:

$$l_i(\phi,\theta) = \underbrace{-\mathbb{E}_{z\sim q_\phi(z|x_i)}[\log p_\theta(x_i,z)]}_{\text{Reconstruction loss}} + \underbrace{\kappa}_{\text{KL weight}} * \underbrace{KL(q_\phi(z|x_i) \parallel p(z))}_{\text{Regularization term}}, \qquad (3.1)$$
$$\text{with } \kappa \text{ between } [0,1]$$

Another suggestion Bowman et al. proposed to cope with the KL collapse, is to weaken the decoder. This can be achieved by removing conditional information during learning, i.e., *word dropout*. By removing the previous word, which the RNNLM uses to predict the next word, the decoder is forced to rely on latent variable z to make good predictions. In case of the VAE language model, they have dropped a percentage of the input at random and replaced it with the word token <UNK>.

After carrying out the experiments, the researchers found that they were unable to train models for which the KL term dominated the reconstruction term in the loss function. Suggesting that it is easier to learn local statistics, as does the RNNLM. Also, in terms of total cost of the loss function, they found that the standard RNNLM performed slightly better than the VAE language model.

A qualitative analysis, however, showed that the VAE language model produces more varied vocabulary and words related to the topic of the sentence compared to the RNNLM, when imputing missing words. Also, they were able to let the decoder construct meaningful and grammatically correct sentences by interpolating in the latent space. The interpolated sentences mostly showed syntactic information similarity, such as the number of words and parts-of-speech (POS).

They also showed that the higher the *word dropout* rate, the higher the KL divergence term in the loss function. Meaning the amount of information stored in the latent variable z increases. The qualitative analysis also showed better results for a higher KL divergence term.

3.2 Other related work

The problem with the KL collapse has also been addressed by others. Yang et al. (2017) [6] replaced the LSTM decoder by a dilated Convolutional Neural Network. The structure of their proposed VAE is shown in Figure 3.2.

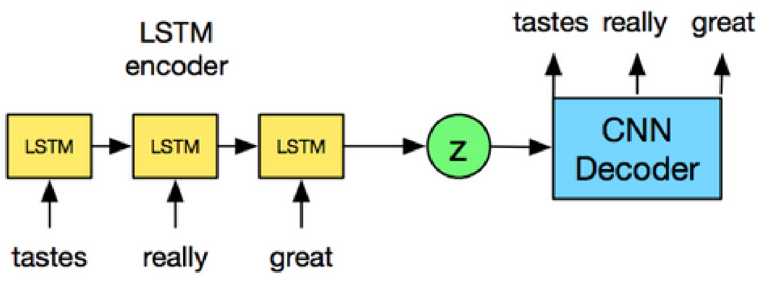


Figure 3.2: The structure of the VAE language model proposed by Yang et al. Source: [6]

They showed a better result (in terms of total cost) on their experiments, compared to the model proposed by Bowman et al. and the RNNLM. By using the dilated CNN as decoder, the KL collapse can be prevented during training.

Also Semeniuta et al. (2017) [23] tried to alleviate the KL collapse during training. Hereto, they proposed a hybrid architecture with a convolutional encoder and a deconvolutional decoder combined with a recurrent layer. Furthermore, they added an auxiliary cost term in the optimization function (the ELBO), in order to force the decoder to rely more on the latent variable z.

Their hybrid VAE language model converges faster and better than the model of Bowman et al. Their proposed auxiliary cost also prevented the KL term to collapse to zero.

Also Hu et al. (2017) [24] have build upon the work of Bowman et al. They have combined a VAE with holistic attribute discriminators. To this end, they have allocated separate dimensions of the latent space to different attributes. In other words, the unstructured variables z are combined with a set of structured variables c, each of which contains an independent semantic feature of sentences. The sentence generator can then generate sentences conditioned on the combined vector (z, c). One can then for instance control the sentiment or tense of a generated sentence by specifying the attribute c. One can see this method as *controlled* text generation.

In conclusion, Bowman er al. were the first to use VAEs for text generation. A qualitative analysis of their proposed showed the capability of the VAE to generate consistent and grammatically correct sentences. However, they also noticed the KL collapse prevented their model from learning. Hereto, they proposed several techniques to alleviate this problem. Other researchers have built upon the work of Bowman et al., and proposed different solutions to the KL collapse or more advanced techniques to create a VAE capable of generating text.

In the next chapter, we will elaborate on the data used for our experiment.

Chapter 4

Data

To illustrate how we can use a VAE to encode text to a latent space, and create novel text by interpolating in the latent space we will use the freely available News Category Dataset [25].

The dataset contains about 200k news headlines + 200k short descriptions of the article from the year 2012 to 2018 obtained from the news website the Huffington Post (HuffPost). All headlines have been annotated with their corresponding category. A snapshot of the headlines along with the short description is shown in Table 4.1 and Table 4.2 respectively.

Table 4.1: Snapshot of headlines from News Category Dataset.

News Category Dataset - headlines

News headline	**Category**
There Were 2 Mass Shootings In Texas Last Week, But Only 1 On TV	Crime
Will Smith Joins Diplo And Nicky Jam For The 2018 World Cup's Official Song	Entertainment
'Trump's Son Should Be Concerned': FBI Obtained Wiretaps Of Putin Ally Who Met With Trump Jr.	Politics
Mystery 'Wolf-Like' Animal Reportedly Shot In Montana, Baffles Wildlife Officials	Weird News

Table 4.2: Snapshot of short descriptions from News Category Dataset.

News Category Dataset - short description

Short description	**Category**
She left her husband. He killed their children. Just another day in America.	Crime
Of course it has a song.	Entertainment
The wiretaps feature conversations between Alexander Torshin and Alexander Romanov, a convicted Russian money launderer.	Politics
'We have no idea what this was until we get a DNA report back.'	Weird News

In total there are 40 different news categories. The categories along with their headline counts are shown in Table 4.3.

Table 4.3: Categories along with their headline count.

Category	**Headline count**	**Category**	**Headline count**
Arts	1509	**Media**	2815
Arts & Culture	1339	**Money**	1707
Black Voices	4528	**Parenting**	8677
Business	5937	**Parents**	3955
College	1144	**Politics**	32739
Comedy	5175	**Queer Voices**	6314
Crime	3405	**Religion**	2556
Culture & Arts	1030	**Science**	2178
Divorce	3426	**Sports**	4884
Education	1004	**Style**	2254
Entertainment	16058	**Style & Beauty**	9649
Environment	1323	**Taste**	2096
Fifty	1401	**Tech**	2082
Food & Drink	6226	**Travel**	9887
Good News	1398	**Weddings**	3651
Green	2622	**Weird News**	2670
Healthy Living	6694	**Wellness**	17827
Home & Living	4195	**Women**	3490
Impact	3459	**World News**	2177
Latino Voices	1129	**WorldPost**	6243

A few statistics of the dataset are shown in Table 4.4. As is shown, the short description is on average 10 words longer than the headline. During the experiment, we will concatenate the headlines and short description into one dataset, yielding 401.706 instances in total. There are 116.617 unique words present in the dataset.

Table 4.4: Statistics of dataset

	Average number of words
Headline	9,54
Short description	19,73

In the next chapter we will explain the methodology for our experiment.

Chapter 5
Research method

In this section, the proposed VAE text generation model is presented. We will start with explaining how we pre-processed the data in order to feed it to the model. Next, we will elaborate on the architecture of the VAE model. We will explain the working of each of the components, and clarify the design choices made.

5.1 Pre-processing

We cannot feed the raw text to a neural network, as NNs can only work with numerical vectors as input. In order to create these vectors, we will first need to do some pre-processing. The first step to take is to *tokenize* the text. Tokenization means splitting texts into space-separated sequences of words, and then split these sequences into lists of tokens (words). By default, all punctuation is removed.

Next, all instances will be transformed into sequences of integers, with each word mapped to a unique integer value. Since not all sentences are equally long, the vectors also differ in length. Hereafter, we need to *pad* all sequences to a fixed vector length because the model cannot work with different vector lengths. We set the fixed the vector-length at 10 words. Meaning, all sentences longer than 10 words will be truncated and all sentences shorter than 10 words will be filled up with zeros. A schematic overview of the pre-processing pipeline is shown in Figure 5.1.

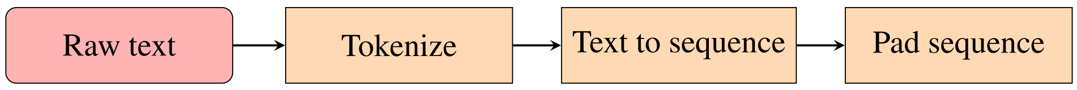


Figure 5.1: Pre-processing pipeline

5.2 VAE architecture

As explained in the previous section, after pre-processing we will have a sequence and this sequence will be fed to the model. Since we want to generate text (i.e., a sequence), our VAE model will be based on a so-called sequence-to-sequence (seq2seq) architecture.

This architecture is comprised of two models: a bidirectional LSTM encoder and an LSTM decoder. We will first explain the working of a (bi)LSTM network in detail, after which we will explain how this is implemented in the VAE model.

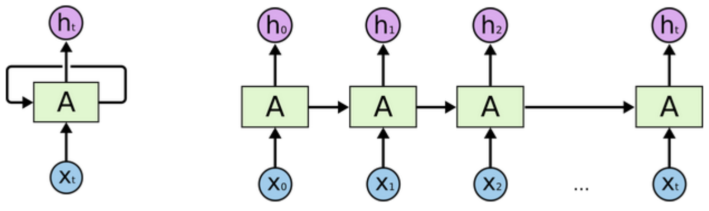
5.2.1 Encoder-Decoder networks

We will start with giving a detailed description on the working of a LSTM network (the decoder in the VAE). Followed by the working of a bidirectional-LSTM (biLSTM) network (the encoder).

5.2.1.1 LSTM network

A LSTM network is build upon a Recurrent Neural Network (RNN), which are both an extension to a standard NN. Standard NNs are a feed-forward network, i.e., they feed information straight through a number of layers and assume that two successive inputs are independent of each other.

This assumption, however, does not hold when one is dealing with text. For instance, when one wants to predict the next word in a sequence, it is important to know which words came before it. RNNs are able to use this sequential information, and are thus much more suitable when dealing with text. RNNs have loops in their network, allowing for a 'memory' which captures information about previous input. The RNN can be seen as multiple copies of the same network, each passing information to a successor network. A part of a RNN with a loop and an unrolled RNN is showed in Figure 5.2.



(a) RNN with loop (b) Unrolled RNN

Figure 5.2: Visualization of a RNN. Source: [7]

By unrolling we mean that we write out the network for the entire sequence. For instance, in case of a sentence of 10 words, this network would be unrolled into a 10-layer Neural Network. To put it more formally:

- x_t denotes the input at time step *t*.
- h_t denotes the hidden state at time step *h*. h_t is calculated based on the previous hidden state and the input of the current time step. $h_t = f(Ux_t + W_{h_{t-1}} + b)$, with f a non-linearity function, W and U the weights, and b the bias

- σ_t denotes the output at step t. $\sigma_t = \frac{1}{1+e^{-t}}$. For instance, if we want to predict the next word in a sentence, it would be a vector of probabilities across the vocabulary.

A RNN shares the parameters U, W across all steps, reflecting the fact that we are performing the same task at every time step, just with different inputs. Sharing the parameters greatly reduces the total number of parameters we need to learn.

In theory, RNNs can pass on information of previous time-steps infinitely long. In practice however, RNNs suffer from *vanishing* or *exploding* gradients, which obstructs the model from learning long-term dependencies. The gradients are used to update the parameters of the RNN. But when the gradient becomes smaller and smaller, the parameter updates become insignificant. In effect, no learning is done.

On the other hand, if the gradients have large values, they can get larger very fast and eventually crash the model. This is the *exploding* gradient problem.

To tackle this problem, the LSTM network was proposed by Hochreiter and Schmidhuber (1997) [26]. As was shown in Figure 5.2, the RNN consists of a chain of repeating modules of neural networks. Inside the module, there is a very simple structure, such as a single $\tanh$ layer. LSTMs also have a chain structure, but the repeating module has a much more sophisticated structure within. A visualization of a LSTM cell is shown in Figure 5.3.

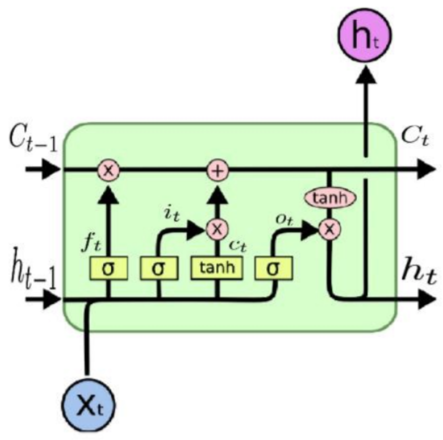


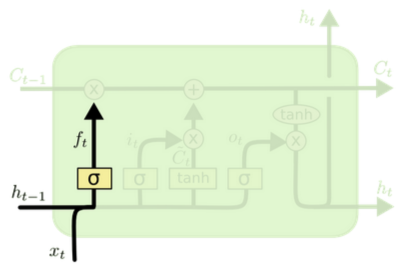
Figure 5.3: The structure within a LSTM cell. Source: [8]

The upper horizontal line in Figure 5.3 is the cell state. This cell state can be seen as a conveyor belt. This belt runs through the entire chain of modules, with some interactions on its way. The LSTM can remove or add information to the cell state. These actions (removing/adding) are controlled by so-called *gates*.

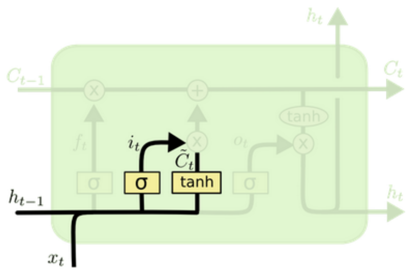
LSTMs have four NN layers: input gate layer i_t, output gate layer o_t, forget gate layer f_t and cell state C_t (also referred to as memory state). There are four steps the LSTM takes:

1. Decide what information to remove from the cell state
2. Decide what new information to store in the cell state
3. Update the old cell state, C_{t-1}, into the new cell state C_t
4. Decide what information to output

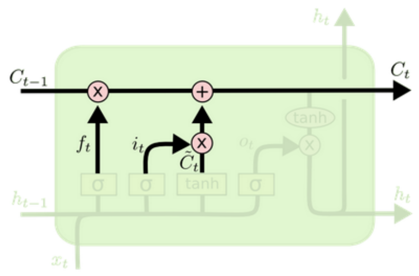
We will elaborate on each of the steps taken, and the steps are also visualized in Figure 5.4.



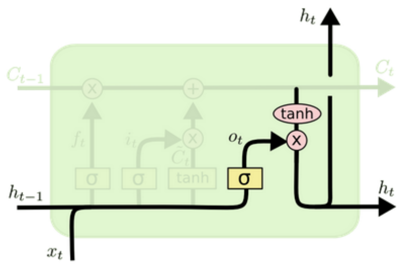
(a) Decide what to remove from cell state



(b) Decide what new information to store in cell state



(c) Update the old cell state into the new cell state



(d) Decide what information to output

Figure 5.4: Working of a LSTM cell. Source: [7]

5.2.1.2 Forget gate

The LSTM starts with deciding what information to remove from the cell state. To this end, it uses the forget gate layer f_t. The formula of f_t is:

$$f_t = \sigma(W_f * x_t + U_f * h_{t-1} + b_f) \tag{5.1}$$

The forget gate takes the input of the previous hidden state, h_{t-1}, and the input, x_t into account, and outputs a number between zero and one. A value of zero means all information will be removed, while a value of one means all information is kept.

5.2.1.3 Input gate

The next step is to decide what new information to store in the cell. The input gate i_t will first decide which values to update. Secondly, a vector with new candidates values, $\tilde{C_t}$, is created by a $\tanh$ layer. This vector can be added to update the state, which will be done in the next step. The formulas for i_t and $\tilde{C_t}$ are:

$$\begin{aligned} i_t &= \sigma(W_i * x_t + U_i * h_{t-1} + b_i) \\ \tilde{C_t} &= \tanh(W_C * x_t + U_C * h_{t-1} + b_C) \end{aligned} \tag{5.2}$$

5.2.1.4 Update new cell state

Now the old cell state, C_{t-1}, will be updated into the new cell state C_t. The formula for C_t will be:

$$C_t = i_t * \tilde{C_t} + f_t * C_{t-1} \tag{5.3}$$

As is shown, the old state C_{t-1} is multiplied by the forget state f_t, and the candidate values $\tilde{C_t}$ are multiplied by how much we want to update each state value (i_t).

5.2.1.5 Output gate

Lastly, the LSTM must decide what to output. This output will be based on the cell state, after applying a filter. The sigmoid layer will decide what parts of the cell state to output, and then a $\tanh$ function will be applied on the cell state to push the values between -1 and 1. After applying the $\tanh$ function, we will multiply it by the output of the sigmoid layer. We can summarize this as follows:

$$\begin{aligned} o_t &= \sigma(W_o * x_t + U_o * h_{t-1} + b_o) \\ h_t &= o_t * \tanh(C_t) \end{aligned} \tag{5.4}$$

Now that we know how a LSTM network works, we can explain how a biLSTM network works.

5.2.1.6 biLSTM network

Bidirectional LSTMs are an extension of standard LSTMs [27]. Using a biLSTM means that two instead of one LSTMs are trained on the input sequence. The first LSTM takes the input sequence as-is, and the other takes a reversed copy of the input. In effect, the LSTM that runs backwards contains information from the future and this can add additional context to the network. The outputs of the two networks are concatenated at each time step. A schematic overview of a biLSTM network is shown in Figure 5.5. A standard LSTM only contains information of the past, as the only inputs it receives are from the past.

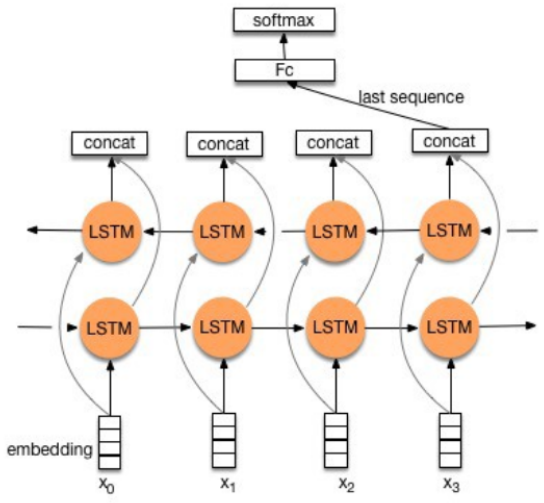


Figure 5.5: Schematic overview of biLSTM network. Source: [9]

Until this point, we have explained that our model is based on a seq2seq architecture and explained in detail the working of both neural nets. Next, we will elaborate on the specific input given to the encoder and decoder, and also the output they return.

5.3 Encoder architecture

Recall from Chapter 2 that we have defined $q_\phi(z|x)$ as our (approximate) variational posterior. We will specify this as a diagonal Gaussian distribution. Our posterior distribution then becomes:

$$q_\phi(z_n|x_n) = \mathcal{N}(z_n|\mu_\phi(x_n), \text{diag}(\sigma^2_\phi(x_n))), \text{ given } x_n \tag{5.5}$$

As explained in Chapter 2, we will not optimize every local variational parameters ϕ_n for every data point, but learn the global variational parameters ϕ_n instead. These parameters are the weights of the inference network.

Our approximating distribution of Equation 5.5 outputs the mean $\mu_\phi(x)$ and log variance $\log \sigma^2_\phi(x)$. In order for our inference network to output the mean $\mu_\phi(x)$ and log variance $\log \sigma^2_\phi(x)$, we need to specify the architecture of our encoder network. That is, $q_\phi(z_n|x_n)$ is a neural network with several hidden layers.

5.3.1 Word embedding

The NN architecture starts with giving the padded sequences as input to our network. We have capped the padded sequences at 10 words, hence there will be 10 time steps in the model. Each word of the sentence will be fed to the model at each time step.

In many NLP applications *word embeddings* are used as extra feature for a given task. Word embeddings represent words as real-valued vectors in a high-dimensional space. To this end, words that are semantically close are also mapped close to each other in the high-dimensional space.

We will include an embedding layer in the model, using the pre-trained 300-dimensional Global Vectors for Word Representation (GloVe) [28]. We will use the GloVe to create an embedding matrix, and this matrix is then given to the embedding layer as word embedding weights. Our dataset contains almost 120.000 unique words, but to reduce memory requirements and complexity we will use the 12.000 most frequently used words to create our embeddings matrix. As we chose the 300-dimensional version of GloVe, we will end up with a [12.000 x 300] embedding matrix.

Thus, in this layer each word will be transformed into a real-valued vector of length 300. In effect, 300 features are created.

After the embedding layer, several hidden layers with certain transformations will follow and the network gives $\mu_\phi(x)$ and $\log \sigma^2_\phi(x)$ as output. A total overview of our encoder network is shown in Figure 5.6. Each layer processing the output from the previous layer and passes on its output to the next layer. As is shown, the encoder outputs the latent variables z_{mean} and z_{log_var} with 32 neurons.

Also, a summary of the hyperparameters of the biLSTM network along with the values used is given in Table 5.1.

Table 5.1: biLSTM Hyperparameters used in this research and their values.

Type of layer	**Hyperparameter**	**Value/Method**
	Layer	LSTM
biLSTM	Merge mode	Concatenate
	Weights	None
	Number of neurons	[96, 32]
Fully connected	Activation function	{Linear, ELU}
	Dropout probability	[0,2]

5.3.2 Sampling latent variable *z*

After obtaining z_{mean} and z_{log_var} from the encoder, there is one final step we need to do before we can obtain z. We cannot sample z directly from a Gaussian which parameters are the outputs of the encoder. As explained in Chapter 2, we will first need to do the reparameterization trick if we want to train the network with gradient descent. We will thus create a sampling function that is outside the network. We can then obtain z by using the sampling function, which does not depend on anything in the encoder network.

After explaining how to obtain latent variable z, which is the input for the decoder, we can now move towards the decoder architecture.

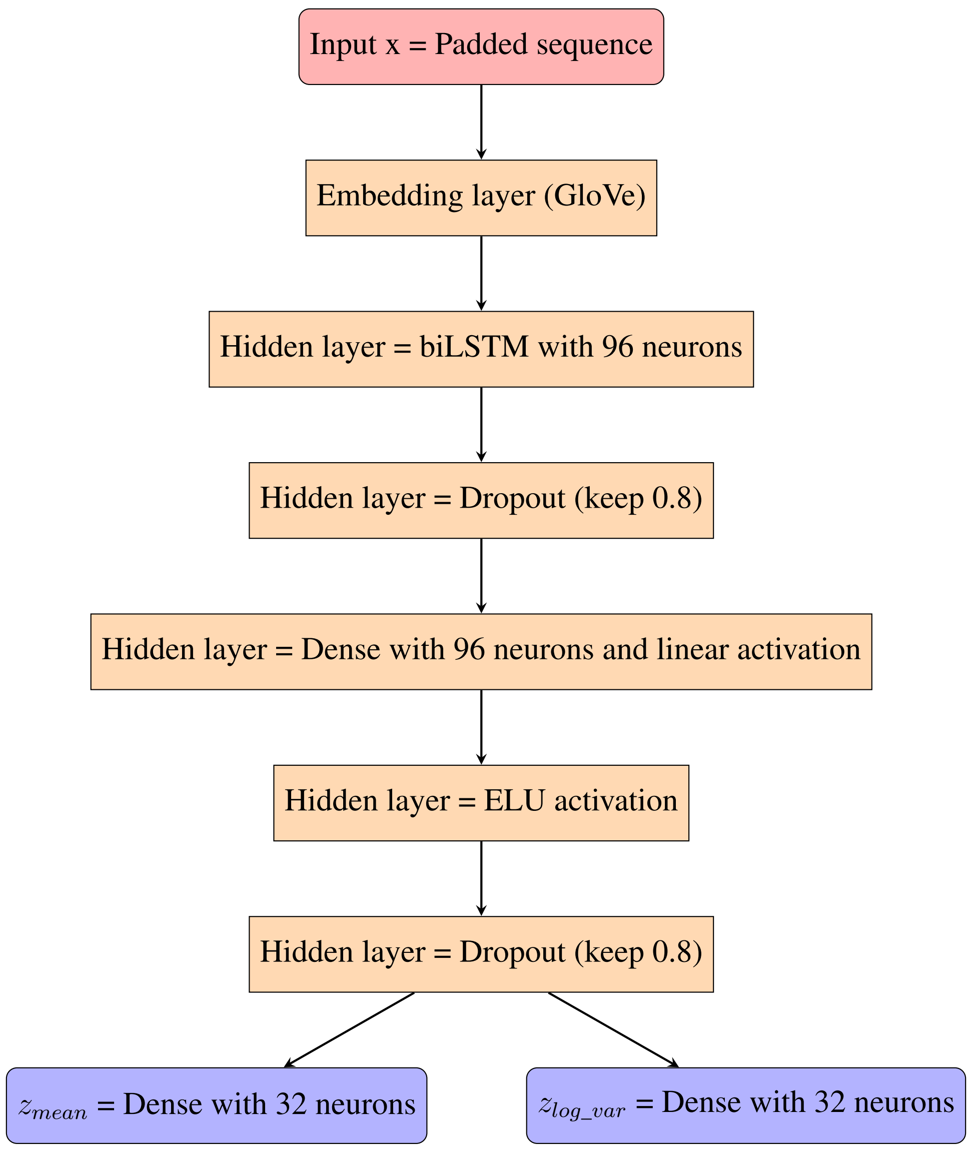

Input x = Padded sequence
Embedding layer (GloVe)
Hidden layer = biLSTM with 96 neurons
Hidden layer = Dropout (keep 0.8)
Hidden layer = Dense with 96 neurons and linear activation
Hidden layer = ELU activation
Hidden layer = Dropout (keep 0.8)
z_{mean} = Dense with 32 neurons
z_{log_var} = Dense with 32 neurons

Figure 5.6: Encoder architecture

5.4 Decoder architecture

Recall from Chapter 2 that we have defined $p_\theta(x|z)$ as the parameterized likelihood and $p_\theta(x|z)$ is our decoder network.

After obtaining z by sampling, we feed the latent representation at every time step as input to the decoder by repeating the vector. Meaning, each time step will get the same input but a different hidden state.

This repeated vector will then go into a LSTM hidden layer, followed by a TimeDistributed layer. The TimeDistributed layer applies fully connected dense on each time step and outputs on each time step separately. The decoder will output the reconstructed (decoded) mean of original input x. A schematic overview of the decoder architecture is shown in Figure 5.7. Also the parameters of the LSTM and TimeDistributed layers are shown in Table 5.2.

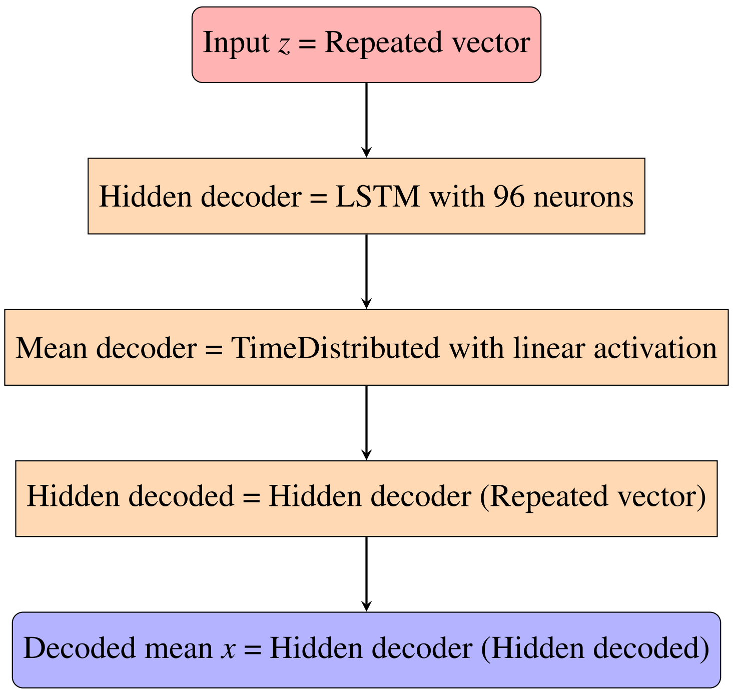

Input z = Repeated vector
Hidden decoder = LSTM with 96 neurons
Mean decoder = TimeDistributed with linear activation
Hidden decoded = Hidden decoder (Repeated vector)
Decoded mean x = Hidden decoder (Hidden decoded)

Figure 5.7: Decoder architecture

Table 5.2: LSTM and TimeDistributed Hyperparameters used in this research and their values.

Type of layer	**Hyperparameter**	**Value/Method**
LSTM	Units (Neurons)	96
	Return sequences	True
	Recurrent dropout	0,2
Time distributed	Layer	Dense with linear activation

Now we have explained how to encode our raw input to latent space, and explained how we can feed this output as input into the decoder. After obtaining the decoded mean x from the decoder network, it is now time to show how we use our output from the encoder and decoder to train our network.

5.5 Custom loss layer

As shown in Equation 2.11, our loss function consists of two parts: the reconstruction loss and the regularization term (KL divergence).

The KL divergence can be computed in closed form (see Kingma and Welling [19] for the derivation):

$$KL(q_\phi(z) \parallel p_\theta(z|x)) = -\frac{1}{2}\sum_{n=1}^{N}(1 + z_{log\_var} - (z_{mean})^2 - e^{z_{log\_var}}), \quad \text{with N the dimensionality of latent variable } z \text{ (32 in our case)} \tag{5.6}$$

To estimate the reconstruction loss we use weighted cross-entropy loss for a sequence of logits [29], which calculates the final softmax internally. Hereto, we let decoded mean *x* (y_pred) be the sequence of logits and input *x* is our true label (y_true).

In conclusion, we use the output from the encoder (z_{mean}, z_{log_var}) to compute the KL divergence, and the output from the decoder (decoded mean *x*) to estimate our reconstruction loss. The total cost will be the two terms summed, and this cost function will be optimized during training.

5.6 Summary

To summarize, we discussed how to pre process the data in order to feed it to a NN. Next, we explained extensively the working of (bi)LSTM networks, which are the building blocks of both the encoder- and decoder network. We also considered the architecture of both the encoder and decoder, as well as the different parameters used in the architecture. Finally, we examined how the output of the encoder and decoder are used to define the loss layer. The loss layer is used to train our network.

In the next chapter we will explain how we have set up our experiment.

Chapter 6
Experimental set-up

In order to find the best possible architecture of the VAE to generate text, experiments have been carried out. In this chapter we will start with explaining how we have split our dataset into a train/validation/test set, and elaborate on some more machine learning set-up choices. Followed by explaining what different parameter setting we have used in our models. Thereafter, we will elaborate on how we will evaluate our models. Lastly, we will give a brief overview of the programming tools used.

6.1 Machine Learning set-up

Before we will run several NN architectures (models) on the data, we will first split the data in a training, validation and test dataset. Since our dataset is relatively large, we will use most of the instances for training, and a relatively small number of instances for validating and testing. From the 401.706 instances, we will use 10.000 instances for both the validation and test dataset, and the remaining instances for the training dataset.

The training dataset will be solely used for training the network. The validation dataset will be used for choosing the best possible model after training. That is, we will track the loss on the validation data during training the NN network, and the model which has the lowest loss on the validation data will be chosen as the best model. The test dataset will be used to calculate the final performance of the model (i.e., we compute the loss on the test data). Also, we will use the test data to create new text.

In order to obtain a robust estimate of the performance of each model, we will run each model 10 times on the dataset. To this end, we will shuffle the dataset before each run. In effect, the training/validation/test dataset will contain different instances in each run. The performance measure is then averaged across all runs for each different model.

We will train each model for 200 epochs and we will use the gradient descent algorithm Adam [30] to update the weights of the network during training. Since 200 epochs is quite a lot, resulting in a high computational cost, we will also consider so-called ‘early stopping’ during training. Meaning, if the loss does not improve during 10 consecutive epochs, we will stop the training process.

Since our dataset is relatively large, our entire training dataset cannot fit into Random-access memory (RAM). To be able to use the the entire training dataset, we will use a 'fit_generator' function. This function gradually feeds our headlines to the model until all we have reached the desired number of epochs. In effect, the memory requirements are drastically reduced.

6.2 Model architecture set-up

As explained in the previous chapter, our network consists of different layers with different number of neurons and activation functions. We will keep all those parameters along with their values fixed. We will, however, experiment with different values for the weight of the KL divergence (see Equation 3.1). If the weight is set to zero, no regularization at all is enforced on the encoder. This gives maximum freedom to the encoder, as it can encode as much information as possible into the latent variable z. If the weight is set to one, we are back at Equation 2.11, and equal weight is given to the reconstruction loss and the KL divergence. We will experiment with the following values for the KL weight term: [0, 0.25, 0.5, 0.75, 1].

6.3 Evaluation methods

Our models will be evaluated both quantitatively and qualitatively.

The quantitative analysis will consist of computing the final loss (dissected into the reconstruction loss + KL divergence) on the test dataset. The model is trained on minimizing the loss function, i.e., a lower loss should indicate a better model. In order to evaluate whether this is true, a qualitative analysis is needed.

One could state that the qualitative analysis is of more importance to this research paper, as our research goal was to *'generate novel text by interpolating in the latent space'*.

As explained in [5], a linear interpolation between sentences can also be seen as a *homotopy*. We will follow the definition of [5] when creating a homotopy between two codes $\vec{z_1}$ and $\vec{z_2}$:

$$\begin{aligned} z_t &= \vec{z_1} * (1-t) + \vec{z_2} * t, \\ &\text{for } t \in [0,1]. \end{aligned} \tag{6.1}$$

In other words, if we encode two sentences to a latent space and we enter those latent space points into the formula, the formula finds points on the line between the input points. We then have new latent space points, which we can decode back using the decoder and as a result we obtain a new sentence.

The quality of the obtained homotopy can be evaluated by several criteria:

- **Grammar**: Are the new sentences grammatically correct?

- **Syntactic information**: Do the new sentences have syntactic similarities with the two input sentences (e.g., number of words, POS)?
- **Semantic similarity**: Do the new sentences contain words that are semantically close with words from the two input sentences?
- **Diversity**: Are the newly generated sentences diverse and original?

6.4 Programming tools

Programming the VAE model is done in Python using Keras [31] with Tensorflow as back-end. Furthermore, since training complicated NN architectures is very computationally expensive, we will use Google Colaboratory. Google Colab is a free Jupyter notebook environment that runs in the cloud, and allows for the free use of a GPU.

Chapter 7

Results

In this chapter, the results of this research are presented. We will start with showing the progress of the loss function during training the networks. Followed by the final quantitative performance on the test dataset. Next, we will move to the qualitative analysis and create novel sentences by interpolating in the latent space. Also we will evaluate the effect of the KL weight term on both the quantitative and qualitative analysis.

7.1 Training progress

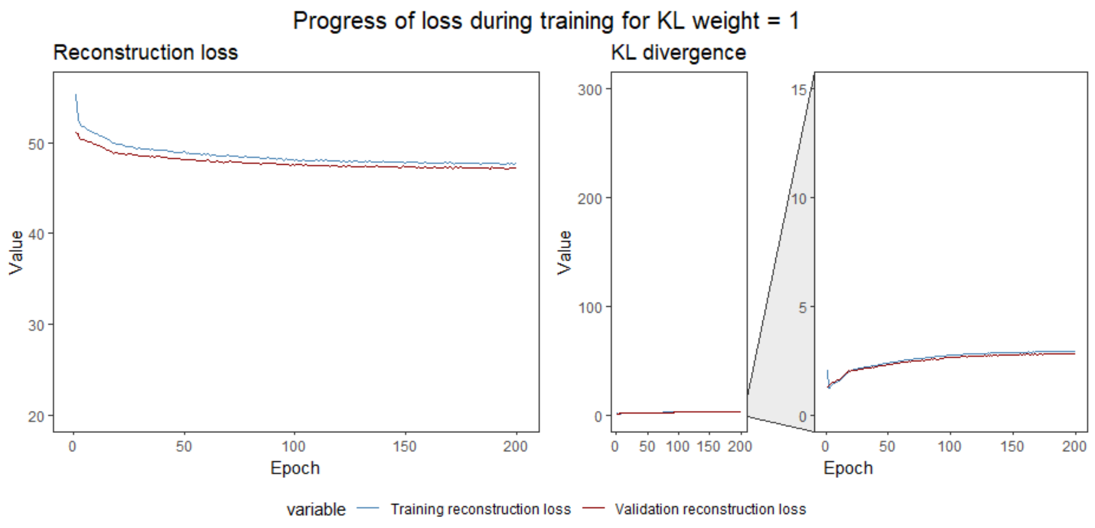
We started with training for 200 epochs, and tracked the reconstruction loss and KL divergence on both the training and validation dataset. The results are shown in Figure 7.1.

As is shown, for a balanced loss function (KL weight = 1), the reconstruction loss stabilizes after about 50 epochs to a loss of 48. The KL divergence stabilized to a value of about 2.5.

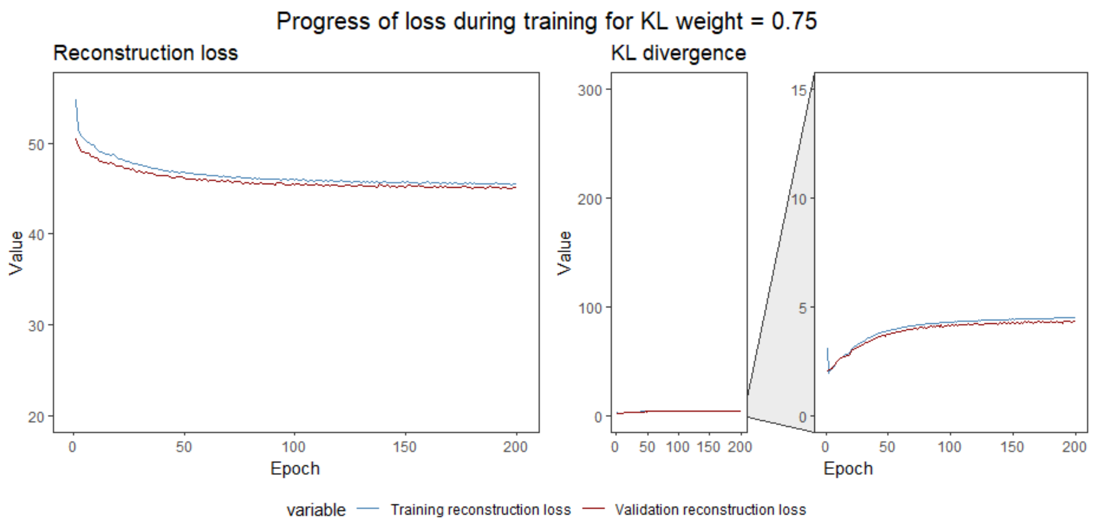
As the penalization weight decreases, one can see that the reconstruction loss becomes lower and the KL divergence increases. This makes sense, as the KL divergence term becomes a smaller part of the loss function, which we are trying to minimize during training.

When the KL weight term is zero, i.e., there is no regularization at all, the KL divergence keeps on increasing until almost a value of 300. The reconstruction error lowers until around a value of 23. The reconstruction error also shows a bit more spiky behavior compared to the other KL weight parameter settings.

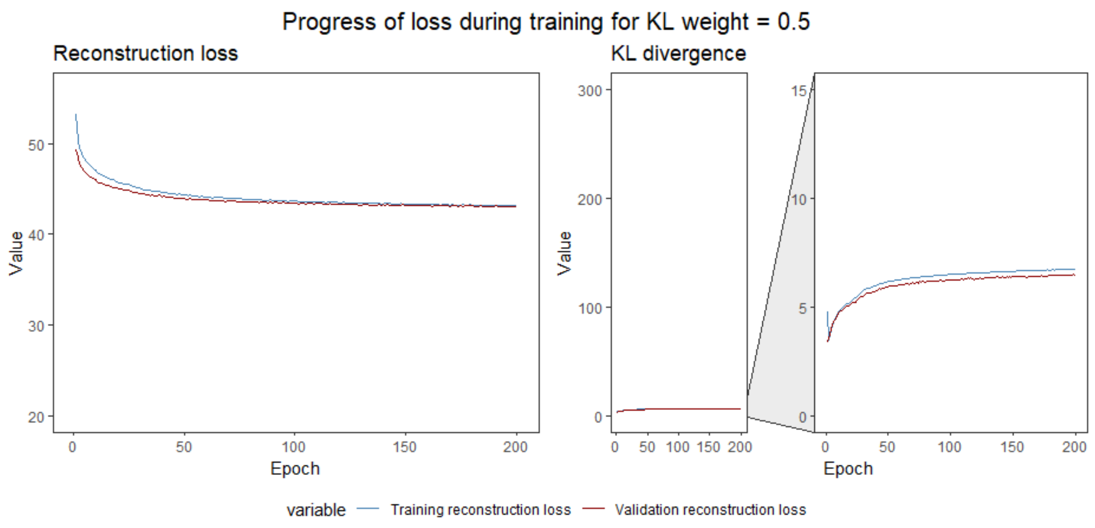
Furthermore it is noted that the error on the validation dataset is a bit smaller compared to the training dataset. This might be due to the fact that we use dropout layers in our network, which function as regularization layers. Because dropout is activated during training, but deactivated when evaluating on the validation set, it might result in a lower loss on the validation dataset.



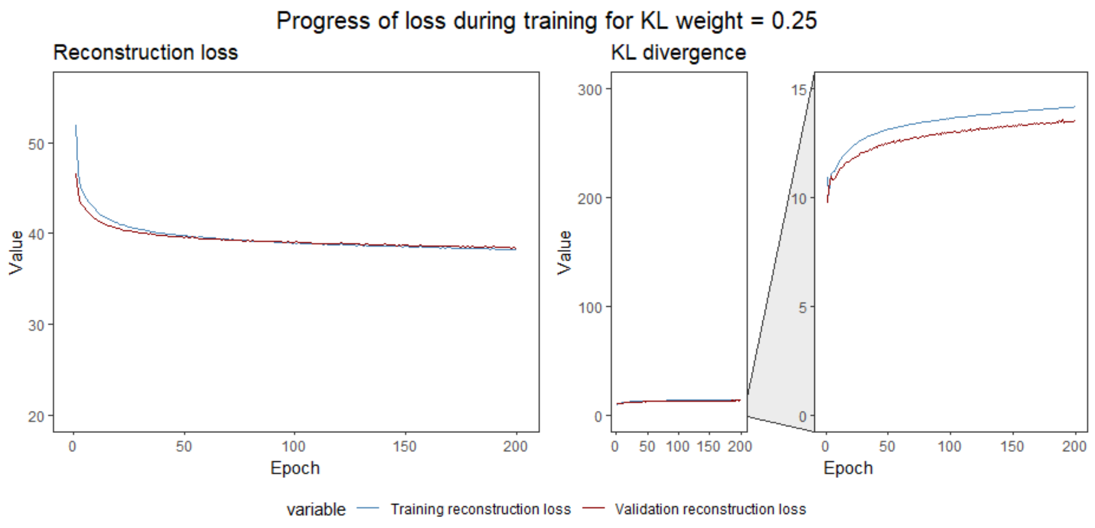
(a) KL weight = 1



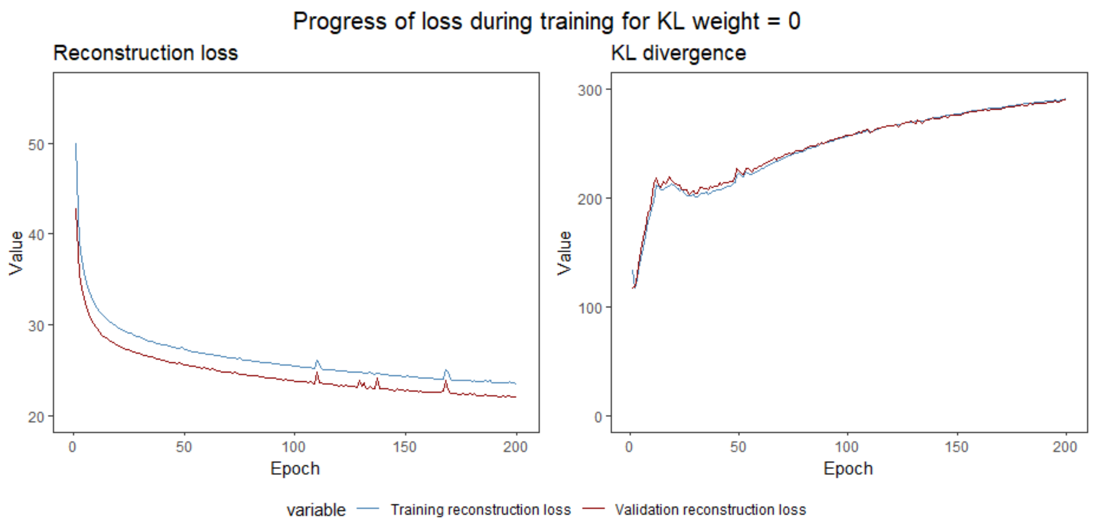
(b) KL weight = 0.75



(c) KL weight = 0.5



(d) KL weight = 0.25



(e) KL weight = 0

Figure 7.1: Progress of loss during training

7.2 Results on test dataset

As explained in the previous chapter, we used 10.000 instances as test dataset. We ran all parameter settings 10 times, and took the mean of the results in order to obtain a robust estimate of the models' performance. The results (reconstruction error and KL divergence) plus the standard deviation of the results are shown in Figure 7.2.

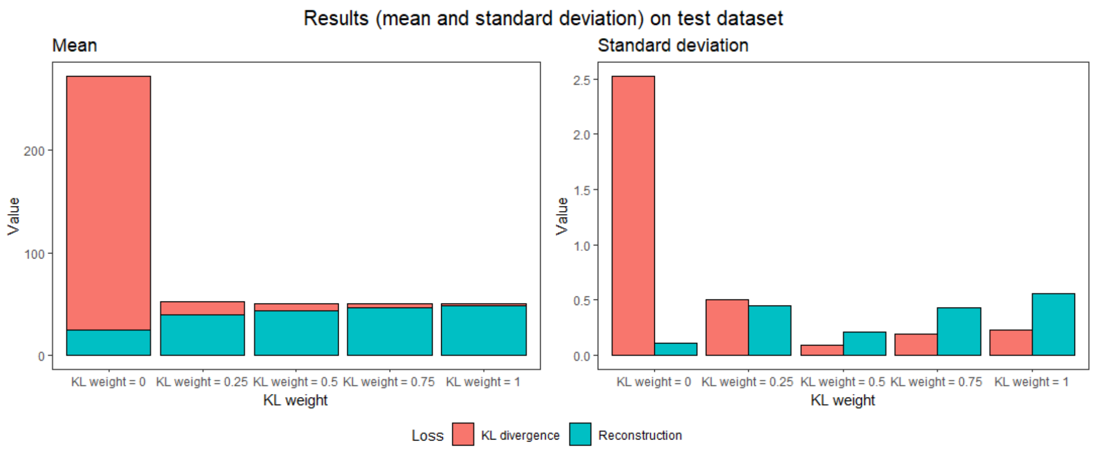


Figure 7.2: Results on test dataset

As is shown, the KL divergence becomes larger as the weight term decreases. The KL divergence ‘explodes’ when the weight term is set to zero. Furthermore, it is noted that the total loss for a KL weight of [0.25, 0.5, 0.75, 1] does not differ that much. The standard deviation of the results is not very large.

Now that we have seen the quantitative results for each parameter setting, we will evaluate if in fact the models are capable of creating novel sentences.

7.3 Qualitative analysis

After the first iteration of training the models, we found that the generated sentences only contained so-called *stop words*, i.e., *a, the, that*, etc. This was due to the fact that our target vocabulary was very big (116.617 unique words). This large vocabulary makes it impossible to use the regular softmax in our network, because then a probability for every word needs to be calculated. To circumvent this problem, we used *sampled softmax* in our models. By doing this, you only take a subset of words from your vocabulary into account when calculating the probability.

After implementing the sampled softmax, the models were capable of generating words other than just stop words.

Also, as explained in the previous chapter, we experimented with early stopping to speed up the training process. Training a model for 200 epochs took almost 9 hours, which makes it difficult to do quick iterations. However, if we stopped the training process when the loss had not improved during 10 consecutive epochs, the models degraded again to only generating stop words. Therefore, all our models were trained for 200 epochs.

To generate novel news headlines, we took 2x2 headlines from the dataset, and interpolated in the latent space. For exemplary purposes, we will show the results of the models with a KL weight of 1 and 0. The other homotopies are shown in the

Appendix. The used headlines are:

1. - **'Trump Says U.S. Has Had High-Level Talks With North Korea'**
   - **'Trump Says U.S. Will Use Military To Protect Border With Mexico'**
2. - **'Facebook Will Let Users See Which Sites Are Tracking Them'**
   - **'Facebook Pledges To Add More Local News To Newsfeeds'**

We created six new headlines by interpolating in the latent space. The results for a KL weight of 1, are shown in Figure 7.3. As one can see, the generated sentences are not plausible English. The sentences are nor grammatically correct, nor show any kind of syntactic similarity with the two input sentences.

'Trump Says U.S. Has Had High-Level Talks With North Korea'
asleep represents smiles looking channing smiles costing dictator schedules schedules
asleep represents smiles asleep woods smiles entrepreneur dictator schedules prague
asleep represents smiles smiles smiles adoption virginia costing woods dictator
nuptials represents smiles wrapping smiles storms off embattled coupled costing
nuptials 'breaking smiles woods 'breaking condoms virginia costing costing off
asleep 'breaking drawn woods 'breaking dictator embattled embattled costing informed
'Trump Says U.S. Will Use Military To Protect Border With Mexico'

'Facebook Will Let Users See Which Sites Are Tracking Them'
terms whistleblower grateful midst infamous 100 funding begun insecurity deeper
terms whistleblower grateful midst begun selena begun trains begun literary
terms whistleblower bon gulf midst bon 100 funding grateful emmanuel
terms whistleblower bon damaged begun revived solitude damaged evening evening
terms whistleblower bon eddie evening grateful midst midst funding arrangement
terms whistleblower bon midst cyberbullying aggression midst midst universe trains
'Facebook Pledges To Add More Local News To Newsfeeds'

Figure 7.3: Sentences produced by model with KL weight = 1.

The results of the model with a KL weight of zero, are shown in Figure 7.4. As shown, the generated sentences even make less sense compared to the sentences generated by the model with a KL weight of 1. Words are often repeated in the sentences, and for some sentences the sentence consist of one word that is repeated 15 times.

‘Trump Says U.S. Has Had High-Level Talks With North Korea’
immigration immigration brzezinski conan brzezinski eerie eerie contribution contribution build
immigration immigration conan conan infants eerie eerie contribution contribution contribution
follow rats conan conan conan petty rats son’s son’s son’s
follow rats rats conan conan rats rats rats rats rats
rats rats rats rats rats rats rats rats rats rats
rats rats rats rats rats rats rats rats rats rats
‘Trump Says U.S. Will Use Military To Protect Border With Mexico’

‘Facebook Will Let Users See Which Sites Are Tracking Them’
distribution bedtime bedtime bedtime distribution distribution distribution distribution debris debris
distribution ’i bedtime bedtime distribution distribution distribution distribution recruit writer
distribution scenarios finalized lisa distribution distribution distribution recruit writer writer
follow follow follow boy’s follow follow follow follow writer writer
follow follow follow follow follow follow follow follow writer writer
follow follow follow follow follow follow follow writer writer writer
‘Facebook Pledges To Add More Local News To Newsfeeds’

Figure 7.4: Sentences produced by model with KL weight = 0.

The results of the models with a KL weight term of [0.25, 0.5, 0.75] are shown in the Appendix. The generated sentences do not show any improvements compared to the other parameter settings. The model with a KL weight of 1 suffered the least of the phenomenon of repeating words in the sentence.

Chapter 8

Conclusion and discussion

In this research paper, an extensive overview on the working of VAEs is presented. A VAE model was proposed to generate novel text by interpolating in the latent space. The results have been presented in the previous chapter. In this chapter, we will evaluate the results and connect the results to our research question. Also, we will reflect on the limitations of this research and give suggestions for future work.

8.1 Evaluating the results

We studied the effect of the KL weight term on the behavior of the loss function. We saw that when the penalty on the KL divergence was relaxed, in effect the reconstruction error lowered at the cost of a higher KL divergence.

The effect of the KL weight term on the quality of the generated sentences was not very clear. The only conclusion that we can draw is that a balanced loss function resulted in sentences without words being repeated.

Although related work showed the potential of VAEs for generating text, we were not able to produce coherent and grammatically correct sentences.

8.2 Research goal evaluated

After evaluating the results, we can answer the research goal which is repeated below.

HOW CAN WE USE VARIATIONAL AUTOENCODERS TO ENCODE TEXT TO A LATENT SPACE AND HOW CAN WE GENERATE NOVEL TEXT BY INTERPOLATING IN THE LATENT SPACE?

The proposed VAE model is the answer to the first part of the research question. In short, we first need to tokenize the text and feed it to our Neural Network through a word embedding layer. After several hidden layers, our encoder network outputs latent variables z_{mean} and z_{log_var}. We feed the output as input to our decoder, and the decoder will output the reconstructed mean of our original input text.

To answer the second part of the research question, we need to look at Chapter 6. Here, we explained that one can encode two sentences to a latent space. One can then draw a line between the two sentences, and find new points on that line. The points can be decoded back using the decoder, with a new sentence as a result.

Although we can thus answer the research question, this only is the technical side of the story. Meaning, it is technically possible to encode text to a latent space and create ‘novel’ text by interpolation. However, the results show no resemblance after real language. The results as they are now, are not suitable for any kind of task.

It is also worth mentioning that the data used in this research might be difficult for a VAE to learn. News headlines contain a lot of names, and do not necessarily qualify as ‘easy’ sentences. Besides that, the research discussed in the Related Work Chapter carried out their experiments on much simpler simpler sentences. For instance, they interpolated between sentences like *‘He was silent for a long moment’* and *It was my turn’*.

8.3 Future work

As mentioned, it is technically possible to use VAEs to generate new text. However, although we experimented with different settings, we could not identify factors that have a positive effect on the quality of the generated sentences.

Therefore, for future work it might be desirable to spend time on trying to pinpoint what factors contribute to the quality of the created sentences. An easier/more homogeneous dataset might be a good starting point.

Since VAEs are very flexible in the sense that one can basically use any kind of encoder and decoder network, experimenting with a different kind of Neural Network might also give better results. Also, no time on parameter tuning was spent in this research, however, the results might benefit from doing so.

Bibliography

[1] Jaan Altossar. Tutorial variational autoencoders. `https://jaan.io/what-is-variational-autoencoder-vae-tutorial/`, 2018.

[2] Aditya Grover. Variational autoencoders. `https://deepgenerativemodels.github.io/notes/vae/`, 2018.

[3] Mark Chang. Variational autoencoders. `https://www.slideshare.net/ckmarkohchang/variational-autoencoder`, 2016.

[4] Sergios Karagiannakos. How to generate images using autoencoders. `https://sergioskar.github.io/Autoencoder/`, 2018.

[5] Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[6] Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. Improved variational autoencoders for text modeling using dilated convolutions. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3881–3890. JMLR. org, 2017.

[7] Christopher Olah. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015.

[8] Manik Soni. Understanding architecture of lstm cell from scratch with code. `https://hackernoon.com/understanding-architecture-of-lstm-cell-from-scratch-with-code-8da40f0b71f4`, 2018.

[9] Ceshine Lee. Understanding bidirectional rnn in pytorch. `https://towardsdatascience.com/understanding-bidirectional-rnn-in-pytorch-5bd25a5dd66`, 2017.

[10] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494, 2015.

[11] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.

[12] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *SSW*, 125, 2016.

[13] Volodymyr Kuleshov, S Zayd Enam, and Stefano Ermon. Audio super resolution using neural networks. *arXiv preprint arXiv:1708.00853*, 2017.

[14] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[15] Iulian V Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. Building end-to-end dialogue systems using generative hierarchical neural network models. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[17] Matt J Kusner and José Miguel Hernández-Lobato. Gans for sequences of discrete elements with the gumbel-softmax distribution. *arXiv preprint arXiv:1611.04051*, 2016.

[18] Sai Rajeswar, Sandeep Subramanian, Francis Dutil, Christopher Pal, and Aaron Courville. Adversarial generation of natural language. *arXiv preprint arXiv:1705.10929*, 2017.

[19] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[20] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[21] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[22] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[23] Stanislau Semeniuta, Aliaksei Severyn, and Erhardt Barth. A hybrid convolutional variational autoencoder for text generation. *arXiv preprint arXiv:1702.02390*, 2017.

[24] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. Toward controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1587–1596. JMLR. org, 2017.

[25] Rishabh Misra. News Category Dataset. `https://rishabhmisra.github.io/publications/`, 2018.

[26] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[27] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610, 2005.

[28] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[29] Seq2seq sequence loss. `https://www.tensorflow.org/api_docs/python/tf/contrib/seq2seq/sequence_loss`.

[30] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[31] François Chollet et al. Keras. `https://keras.io`, 2015.

Appendix A

Appendix

‘Trump Says U.S. Has Had High-Level Talks With North Korea’
posed posed posed posed posed posed posed posed debacle debacle
posed posed posed posed posed posed posed posed posed debacle
posed posed posed posed posed posed posed posed posed debacle
posed posed posed posed posed posed posed posed posed debacle
posed posed posed posed posed posed posed posed posed posed
posed posed posed posed posed posed posed posed posed posed
‘Trump Says U.S. Will Use Military To Protect Border With Mexico’

‘Facebook Will Let Users See Which Sites Are Tracking Them’
debacle 120 120 locals locals locals locals locals locals locals
honors honors honors honors honors honors honors locals locals locals
posed honors honors honors honors honors honors honors honors honors
posed posed posed honors honors honors honors honors honors honors
posed posed posed posed posed posed honors honors honors honors
bug posed posed posed posed posed posed posed debacle debacle
‘Facebook Pledges To Add More Local News To Newsfeeds’

Figure A.1: Sentences produced by model with KL weight = 0.75.

‘Trump Says U.S. Has Had High-Level Talks With North Korea’
kushner sheen sheen sheen sheen competing competing competing competing competing
sheen sheen sheen sheen sheen competing competing competing competing competing
sheen sheen sheen sheen sheen competing competing competing competing competing
sheen sheen sheen sheen sheen competing competing competing competing competing
sheen sheen sheen sheen sheen competing competing competing competing competing
sheen sheen sheen sheen sheen sheen competing competing competing competing
‘Trump Says U.S. Will Use Military To Protect Border With Mexico’

‘Facebook Will Let Users See Which Sites Are Tracking Them’
competing competing competing competing competing competing competing competing competing competing
large competing competing competing competing competing competing competing competing competing
competing competing competing competing competing competing competing competing competing competing
stated explosion explosion explosion unfortunate competing competing competing competing competing
stated explosion explosion explosion explosion explosion competing competing competing competing
stated explosion explosion explosion explosion explosion explosion explosion large large
‘Facebook Pledges To Add More Local News To Newsfeeds’

Figure A.2: Sentences produced by model with KL weight = 0.5.

‘Trump Says U.S. Has Had High-Level Talks With North Korea’
parents oz oz oz oz oz oz oz oz oz
parents oz oz oz oz oz oz oz oz oz
parents oz oz oz oz oz oz oz oz oz
parents oz oz oz oz oz oz oz oz oz
fallon fallon fallon fallon fallon fallon fallon fallon fallon fallon
fallon fallon fallon fallon fallon fallon fallon fallon fallon fallon
‘Trump Says U.S. Will Use Military To Protect Border With Mexico’

‘Facebook Will Let Users See Which Sites Are Tracking Them’
oz oz oz oz oz oz oz oz oz oz
fallon oz oz oz oz oz oz oz oz oz
fallon fallon oz oz oz oz oz oz oz oz
fallon fallon oz oz oz oz oz oz oz oz
fallon fallon oz oz oz oz oz oz oz oz
fallon muffins fallon oz oz oz oz oz oz oz
‘Facebook Pledges To Add More Local News To Newsfeeds’

Figure A.3: Sentences produced by model with KL weight = 0.25.