Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Deep Learning for Natural Language Processing (NLP) using Variational Autoencoders (VAE)

Master's Thesis

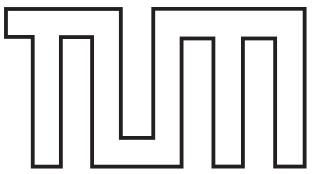
Amine M'Charrak

aminem@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Gino Bruner, Oliver Richter
Prof. Dr. Roger Wattenhofer

October 16, 2018



Technische Universität München
Chair of Media Technology
Prof. Dr.-Ing. Eckehard Steinbach

Master Thesis

Deep Learning for Natural Language Processing (NLP) using Variational Autoencoders (VAE)

Author:	Amine M'Charrak
Matriculation Number:	03655053
Address:	Universitätstrasse 23 8006 Zurich Switzerland
Advisor:	Gino Brunner and Oliver Richter
E-mail:	amine.mcharrak@tum.de
Begin:	16.04.2018
End:	16.10.2018

With my signature below, I assert that the work in this thesis has been composed by myself independently and no source materials or aids other than those mentioned in the thesis have been used.

Zurich, October 17, 2018 _____

Place, Date Signature

Zurich, October 17, 2018 _____

Place, Date Signature

Abstract

Observations of complex data in the real world might be caused by several underlying generative factors that account for specific perceptual features. However, these factors are usually entangled and can not be underlined directly from the data. Modeling such factors could generalize the learning of complex concepts through compositions of simpler abstractions. This enables us to understand the inner structure of the data, to process it efficiently and to control a meaningful generative processes which may eventually open up on artificial creativity and machine intelligence. Deep neural networks have been very successful at automatic extraction of features from various data distributions, making manual feature extraction obsolete. However, due to the complexity of these neural networks, the extracted features often themselves are highly complex. Which hinders reusing them for downstream tasks as humans can not interpret the extracted meaning of these features due to their *entanglement*. Often, neural networks are rather treated as a black box and we have to trust external evaluation metrics such as train and test error. It would be beneficial to understand what kinds of hidden representations the model has learned. Interestingly, several methods exist that are particularly suited for learning meaningful hidden representations. In the image domain an extensive body of research has been carried. Through various deep generative models such as Generative Adversarial Networks (GAN) and Variational Autoencoders (VAE). For example, when being fed with images of faces, a VAE might automatically learn to encode a person‘s gender and beard length/existence into two separate hidden variables. These *disentangled* features we could then use to generate new images which are similar to the underlying image distribution of the images the network was trained with. The goal of this research is to extend these promising results into the natural language text domain. By performing experiments with different neural network architectures for the feature extraction and sample generation in order to find a possible disentangled hidden representation for sentences. State of the art for representation learning in NLP is limited, therefore we are constructing our own new dataset, the dSentences dataset. Thus, we are performing research from scratch by transferring and adapting knowledge and approaches from the image domain into the natural language domain.

Acknowledgements

This thesis was supported by many people. To some of whom I would sincerely like to thank at this point.

To begin with, I am thankful to both Gino Brunner and Oliver Richter for offering me such an interesting topic of investigation.

Gino and Oliver, as my advisors of my thesis work throughout the last half year, deserve special recognition for their always highly qualified comments, ideas and feedback. Their openness, even-tempered and kindly character helped me to stay motivated throughout the time. They have pushed me to think about the most fundamental problems and equipped me with many new tools and skills. I learned more than I was expecting from you. Thank you very much!

There are two other persons I want to thank for being by always by my side. First, my sister Jasmina M'Charrak. Thanks for your support and the nice rambling conversations we had. Second, I would like to thank my girlfriend Subin Park, who is my joy. She lets me see my work through new eyes and assisted me with the dataset construction.

Next, I would like to thank my supervisor Professor Roger Wattenhofer for hosting me at ETH Zurich and making this thesis work even possible. Moreover, do I want to thank Professor Eckehard Steinbach for being my supervisor at TU Munich. Giving me the chance to continue my thesis work at ETH Zurich.

Finally, but first in my heart, my parents Anna and Amal M'Charrak are due my deep gratefulness for their continued moral support and constant love throughout my studies, the latter being of much greater importance. They made me this way, so please direct all complaints to them.

Abbreviations

Abbreviation	Description
NN	Neural Network
FC	Fully Connected
ReLU	Rectified Linear Unit
ELU	Exponential Linear Unit
LSTM	Long Short-Term Memory
CNN	Convolutional Neural Network
MLP	Multilayer Perceptron
RNN	Recurrent Neural Network
GAN	Generative Adverserial Network
AE	Autoencoder
VAE	Variational Autoencoder
NLL	Negative Log-Likelihood
$H(X)$	Entropy of random variable X with probability distribution P
$H(P, Q)$	Cross Entropy between two probability distributions P and Q
$I(X; Y)$	Mutual Information between two random variables X and Y
$D_{KL}(P\|\|Q)$	Kullback-Leibler divergence (relative entropy) of two probability distributions P and Q
XE	Cross Entropy
RV	Random Variable
MSE	Mean Squared Error

ELBO	Evidence Lower Bound
PPL	Perplexity
GloVe	Global Vectors for Word Representation
NLP	Natural Language Processing
CV	Computer Vision
vanilla	standard, usual, unmodified
LM	Language Model
CL	Computational Linguistics
AI	Artificial Intelligence
POS	Part Of Speech
CBOW	Continuous Bag Of Words
Word2Vec	Mapping of sparse one-hot vectors to dense continuous vectors
Gensim	Open-source vector space modeling toolkit
NLTK	Open-source Natural Language Processing Toolkit
Pattern	Open-source toolkit for Natural Language Processing

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Context

Deep neural networks have been very successful at automatic extraction of meaningful features from data. Manual feature engineering is often not necessary anymore, and we can instead focus on designing the architecture of the neural networks. However, due to the complexity of neural networks, the extracted features are themselves highly complex and can often not be interpreted by humans. Instead, the neural network is treated as a black box and we have to trust external evaluation metrics such as train and test error. Often it would be beneficial to understand what kinds of hidden representations, or features, the model has actually learned. Several methods exist that are particularly suited for learning meaningful hidden representations. The model we will be mainly looking at in this research is the Variational Auto-Encoder(VAE) [1], a deep generative model. VAEs have been shown to be able to disentangle simple data generating factors from a highly complex input space. For example, when being fed with images of faces, a VAE might automatically learn to encode the direction of the lighting in a single hidden variable [2, 3]. After training, we can vary each hidden variable and observe the effect on the output, which let's us analyze, albeit manually, what kinds of features the model has learned. Several VAE extensions such as the β-VAE [4] focus mainly on the disentanglement aspect and also introduce data specific disentanglement metrics, making evaluation easier.

1.2 Motivation

Observations of complex data in the real world might be caused by several underlying generative factors that account for specific perceptual features. However, these factors are usually entangled and cannot be underlined directly from the data. Modeling such factors could generalize the learning of complex concepts through compositions of simpler

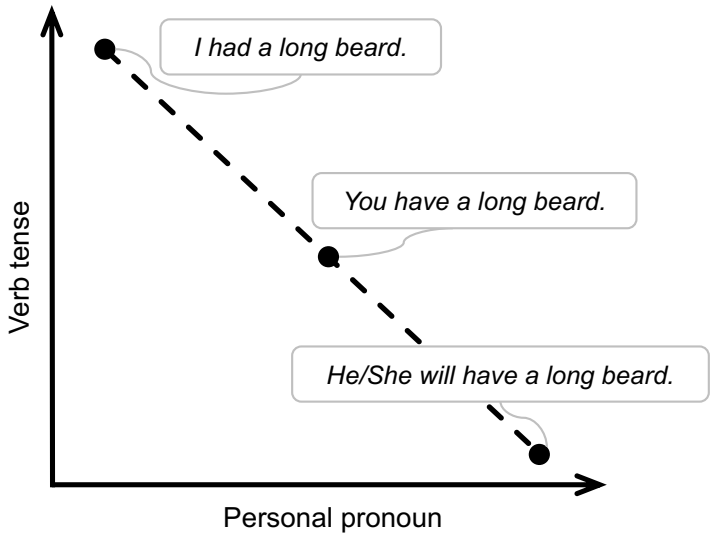


Figure 1.1: Our vision of a disentangled latent space

abstractions. This enables us to understand the inner structure of the data, to process it efficiently and to control meaningful generative processes which may eventually open up on artificial creativity. An extensive body of research has been carried in the field of computer vision and image domain through the application of VAEs on various datasets such as MNIST and CelebA [5]. The goal of this research is to extend these promising approaches to natural language data for natural language processing (NLP), by creating a similar toy dataset with known generative factors similar to the construction of the dSprites dataset by Google Deepmind Research with sophisticated syntactical, grammatical and semantical properties and then applying the recent methods such as β- VAE [6], Factor-VAE [7] and Joint-VAE [8] with variations in encoder and decoder architectural structure blocks and exploration of the latent space. The ultimate goal would be to discover a meaningful latent space which we could envision to look like as shown in Figure 1.1. Thus, we would like to close the gap between representation learning and NLP. With this goal in mind one application we could easily come up with is to simply collect a set of police reports, these are most of the time written in past tense. Next, we could extract the latent representation of each report using a VAE architecture which disentangled generative factors. Finally, by simply flipping the generative factor corresponding to *verb tense* we could reconstruct our report into a short story as these are written in present tense. This demo application is shown in Figure 1.2.

In the following, we propose a few leading questions which represent the interest of this work and which we used as guidance throughout the project:

1. How do different neural network architectures and their disentanglement capabilities perform when using them as building blocks for VAEs?

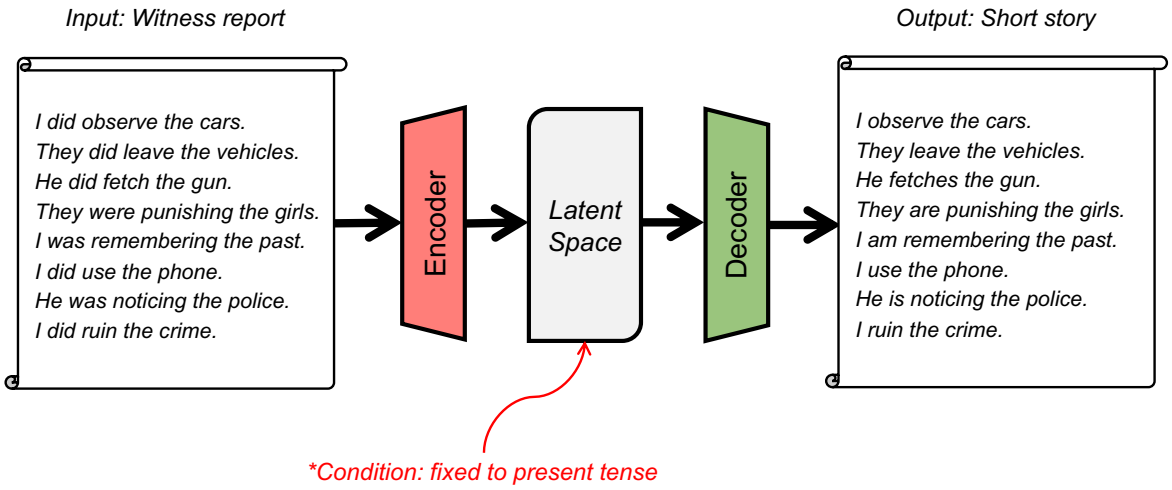


Figure 1.2: Application idea: Flipping a single dimension automatically transforms text written from past into present tense

2. What embedding space should we use as input to our architecture pipelines and does embedding size matter; if this is the case, can we find out why?
3. Does a disentangled representation for natural language even exist and would this representation be human interpretable and reusable for downstream tasks?
4. Can we apply current disentanglement metrics for natural language, if not, what might be a method to measure disentanglement in the text domain?

The original VAE [1] has led to several implementations. These different techniques relate to representation learning derived from variational inference and may support the association process to symbolic modeling. Unsupervised learning methods model data and shape representations without explicit targets or direct assumptions on the generative processes. Amongst them, a key model is the VAE and its variants. In these, an encoded space is learned under certain constraints and regularizations. Encoder and decoder networks are jointly trained in order to fit such objectives while being able to retrieve the original data. Recently, the β-VAE [4] was developed to disentangle factors by learning independent latent variables accounting to generative processes. So far, VAEs (and other generative models) have been most successful in the image domain, where most architectures are based on Convolutional Neural Networks (CNNs), which by themselves are known to be powerful feature extractors. Unfortunately, in the sequential and discrete data domains, such as music and language processing, there has been limited success. The only successful application of VAEs used in NLP are task such as sentiment control and sentence length manipulation [5] as shown in Figure 1.3. Finally, the task of measuring disentanglement is a current state of the art challenge as recent approaches always assume, that the generative

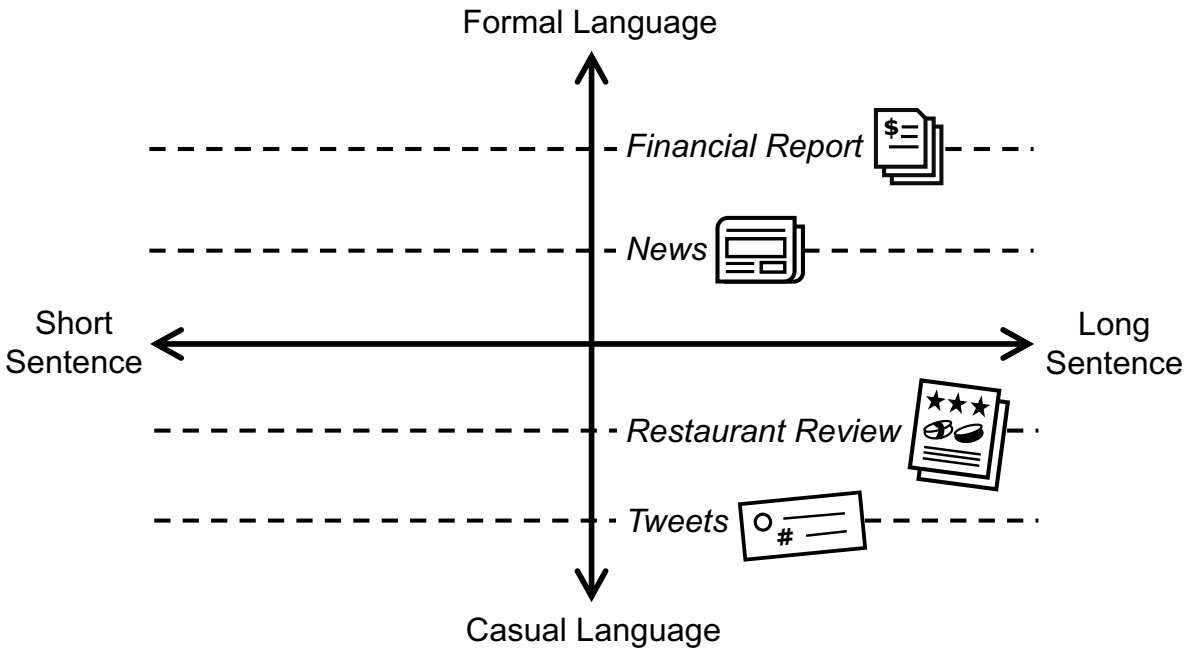


Figure 1.3: State of the art NLP latent space

factors of a dataset are well known in advance and in addition a computationally costly classifier often needs to be trained in order to use such disentanglement metrics [7].

1.3 Contributions

In summary, the main contributions of this thesis work are:

1. Construction of a new dataset, called dSentences, which we will open source . The generation was inspired by the state of the art dSprites dataset published in 2017 by Google DeepMind. It is being heavily used in the image domain for experiments on representation learning
2. Evaluation of different embedding methods for natural language text, in order to determine which approach is most promising to representation learning and disentanglement in NLP
3. Implementation of a fully discrete latent space VAE for natural language datasets, for which the VAE outperforms the AE in reconstruction and disentanglement performance
4. Implementation of modular VAE architectures with interchangeable encoder and de-

coder blocks such as MLPs, CNNs and RNNs

5. Proposing and implementing two new quantitative metrics, which help to keep track of the disentanglement and representations the network has learned. The first is a disentanglement metric which can be used on continuous and discrete latent space VAEs. The second is called, *generative factors accuracy* and helps to understand in which epoch what generative ground truth factors are learned

1.4 Document overview

The thesis is structured as follows. In the second chapter, we will introduce the basic background knowledge which is necessary in order to understand the formalities such as ML and NLP specific expressions and terms. In the subsequent chapter we will do a deep dive into related work not only in the NLP domain but also for the image domain. This is because the original ideas of this thesis topic heavily rely on several VAE models which successfully disentangled the latent space on toy datasets. In the fourth chapter, we will explain why we need a novel dataset with certain attributes, how we constructed this dataset and what its properties are with regards to other datasets in natural language processing and computer vision. Following that, in chapter five we will derivate several bottleneck models and transition from one architecture to the next by adjusting the objective function and adding new terms and concepts. These models were used in the experiments, which are described in the penultimate chapter. We will apply two distinctly different VAEs onto our dataset and observe and describe their behaviour. At the end, in the last chapter, we will make our conclusion and give an overview what needs to be done in order to push the state of the art in representation learning to follow up with the success made within the computer vision domain.

Chapter 2

Background

2.1 Deep Learning (DL)

2.1.1 Overview

In this chapter we want to introduce and explain the most important deep learning neural networks NN blocks, machine learning principles and expressions as they will be heavily used in the subsequent chapters. Deep learning is a subfield of AI and ML. The difference between ML and DL is the fact, that when we have a large amount of data, we do not need to extract manual features as the feature extraction is done by the deep learning model itself. Figure 2.1 The larger the amount of data, the better performances we can expect from a given model. Therefore, deep NNs are powerful black box predictors that achieve impressive performance on a wide spectrum of tasks from natural language processing to computer vision. There learning process in deep learning can be of two types:

1. **Supervised learning**,
   where we have labels for each data sample and the goal is to learn a mapping between raw inputs and the ground truth output label. This label serves as a teaching signal from which the deep learning can learn through feedback
2. **Unsupervised learning**,
   where we have no labels for any data sample and the goal is to learn hidden structures as relationships of raw data inputs. This enables us to solve tasks like density estimation of the underlying distribution [9] or clustering of samples into specific groups e.g. deep k-means [10].

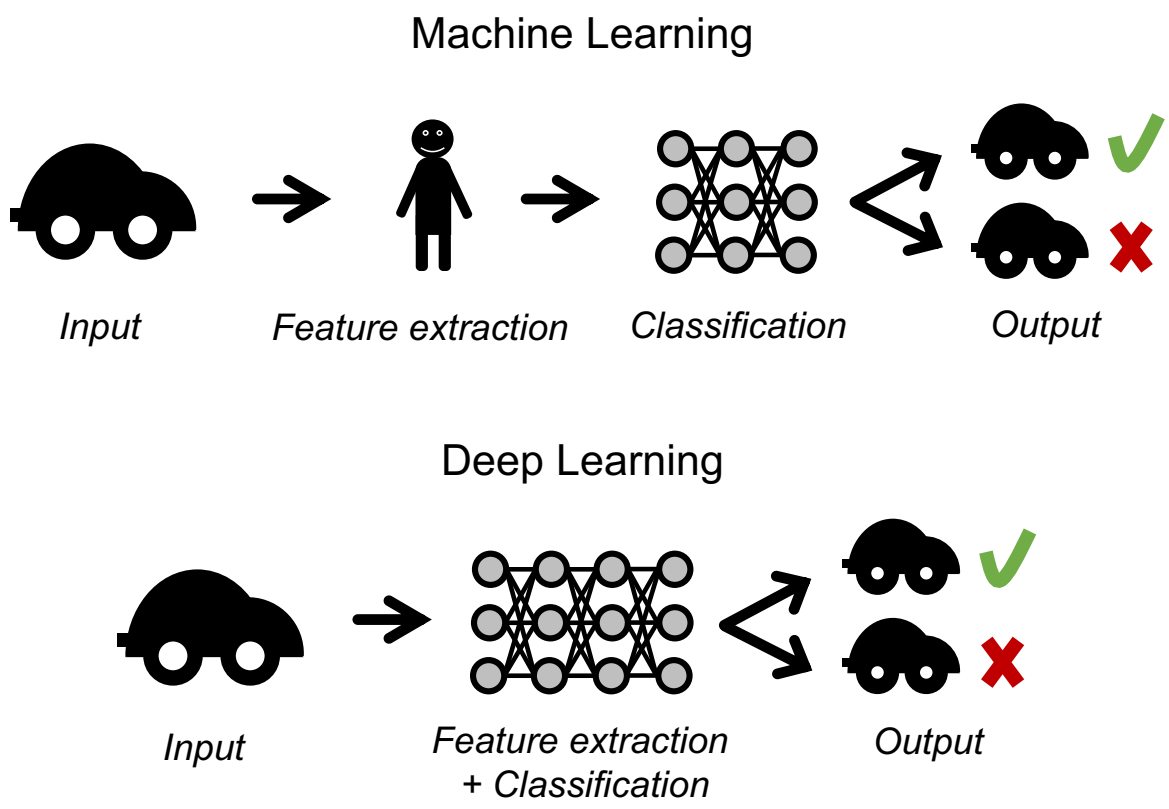


Figure 2.1: Machine Learning compared to Deep Learning

In this thesis work we will be using an unsupervised deep learning model to learn meaningful hidden structures in text data. In specific, we will work with VAEs which do not have labels but as one of the main goals of VAEs is to reconstruct the input, such models use their own input as teaching signal, so we can think of VAEs as a supervised unsupervised learning model.

2.1.2 Deep learning expressions

The following list gives an overview of often used and relevant expression when dealing with neural network models and architectures as we will be using all of them in the next chapters:

- **Backpropagation (BP):** A method which calculates a gradient in order to update neural network parameters. It requires the derivation of the objective/ loss function with regards to the model parameters (weights and biases) on any layer within the network. It uses the chain rule of calculus because derivations have to be done

for several layers from output signal layer back to the initial input layer which is called a backward pass. It is important to mention, that backpropagation can only be performed in layers which consist of deterministic nodes/units. Eventually, the calculated gradients give information about how the NN‘s parameters should be updated in order to optimize the overall performance of the task at hand.

- **Objective:** The objective is the loss function which we try to minimize with an algorithm of our choice. It is used to define the error of the model which can than be used as feedback to improve upon it. The loss function is evaluating the difference between the model‘s output prediction and the actual ground truth value.
- **Overfitting**: A model is overfitting if it performs well on the samples is was trained on but fails to maintain a good performance on unseen and new data samples. We can prevent models to overfit by removing some of the features of a sample (also known as *dropout*), stopping the training process as soon as we see the first signs of overfitting (also known as *early stopping*) or by making the model less complex. This can be achieved in VAEs through *capacity reduction*, where we for instance simply reduce the available units in the latent code z.
- **Epoch**: Given a dataset of n samples, an epoch is completed when the entire dataset is passed forward to the NN output and the gradients from backpropagation through all layers of the NN model back to the input has been completed. After one epoch, the model has seen every sample in the dataset only once.
- **Batch size**: The batch size b is a deep learning hyper parameter which defines the amount of samples we feed and process with the NN concurrently in a single iteration (forward + backward passing). The biggest effect batch size has on a NN is the number of epochs the model needs to achieve some threshold of performance. In general, we need less training epochs when using smaller batch sizes as we converge faster to a certain performance threshold when comparing to larger batch sizes [11].
- **Batch**: The number of batches can be calculated by simply diving the total amount of dataset samples n by the batch size b. An epoch is finished after the model has processed b data batches.
- **Learning rate**: The learning rate is another hyper parameter when training NN models. It controls by how much we are adjusting the weight updates of our network parameters with respect to the loss gradients we receive through backpropagation. It can be seen as a step size along some function. In general, the lower the learning rate, the slower we move on our objective function surface in the direction of the downward slope. This suggests, that the whole learning process of converging to a local minimum will take more time or we might get stuck at a plateau and not learn anything at all. On the other hand, a learning rate which is too large can cause non-convergence as well; for example by oscillating on the objective function or causing weight updates to get so large the our gradients explode. Within this thesis the learning rate was determined via trial and error in an engineering manner.

- **Train and test set**: When developing NN models it is state of the art to split the whole dataset into two portions; a train set and a test set. The model is then being trained only using the train set. After successfully having fitted a model, we then use the test set as as sanity check to figure out if our model generalizes to unseen and new data samples which originate from the same data distribution as the samples it was initially trained on. Therefore, the test set can be seen as a way to evaluate the final performance of a model and to determine if the model has actually 'learned' something or if it only mimics the training samples. This is of key importanbce in our model implementation as we do not want the VAE, which has the task of reconstructing the input x as output $\hat{x}$ to simply learn the identity function $f(x) = x$, which would lead to optimal reconstruction performance but absolutely no generelization.
- **Activation function:** The activation function is crucial in deep learning. Choosing the appropriate activation function decides if the NN model converges or not and whether we are able to learn a non-linear mapping between input and output. In effect, activation functions are often referred to as *non-linearities* as they are the main source of non-linearity of any DL model. All other operations such as convolutional layer or fully-connected layer make the final mapping function of the model remain a complex but linear transformation. Popular activation functions, which we use for our VAE architectures are:

2.1.3 MLP, CNN and RNN as basic neural network blocks

In this subsection we will explain the key principals of three major and fundamental building blocks of every neural network architecture. All of these models are widely applied in ML and DL but each of them is preferred for different data domains such as computer vision or NLP. For us in representation learning is interesting to examine how different *encoder* and *decoder* blocks within the VAE perform and which of these models performs best reconstruction or achieves the highest disentanglement score. All of these blocks are computational graphs which perform a sequence of linear and non-linear transformations. Every block consists of neurons often referred to as units within a layer. These neurons are connected to the input vector $\mathbf{x}$ or previous layer over some weight edges $\mathbf{w}$ and the weighted sum is calculated as the dot product between $\mathbf{x}$ and $\mathbf{w}$ as $\sum_i x_i w_i$. Finally, a bias term b is added to the dot product and a non-linearity is applied onto the final result. All weights and values are $x_i, w_i, b \in \mathbb{R}$. The output of each neuron can be the input of another neuron within the next layer. Stacking up several layers of units will make NN models deeper. There are two types of NN graphs. Firstly, feedforward graphs and secondly graphs which contain cycle edges which connect a neurons output with itself.

Name	**Equation**
Sigmoid function	f(x) $= \sigma(x) = \frac{1}{1+e^{-x}}$, which squashes values into the range $[0, 1]$
Rectified linear unit (ReLU)	$f(x) = max(0, x)$, which introduces some sparsity into NN models and solves common issues with vanishing gradients when using the sigmoid function
Exponential linear unit (ELU)	f(x) $= \begin{cases} \alpha\,(e^{-x} - 1) & for\ x < 0 \\ x & for\ x \geq 0 \end{cases}$, which has the advantage over ReLU in that it tends to converge the loss function faster towards zero
Softmax function	$f(\mathbf{x})_i = softmax(\mathbf{x})_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$ (a generalization of the logistic function) often used for output layers in classification tasks as it directly corresponds to the probability of each class
Hyperbolic tangent function	$f(x) = tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ which squashes values into the range $[-1, 1]$ which is used in each LSTM cell of our LSTM-VAE architecture

Table 2.1: Overview of activation functions used in this project

MLP

Multilayer Perceptrons (MLPs) are often presented in basic ML introduction classes. The basic structure of such an MLP is shown in Figure 2.2. They represent the simplest vanilla artificial neural network. An MLP consist of one or more hidden layers which learn intermediate representations before finally producing some output after the last, deepest layer. The network is a directed graph with processing units and inter-connecting weight edges. Each weight is part of the set of model parameters Φ we have to learn. The units between two layers are densely connected, which is the reason why MLP layers are called as *fully connected* (FC) layers. Each MLP consists of a minimum of three layers, which are input layer, hidden layer and output layer. MLPs are universal function approximators. They are able to project the input into an entangled representation space, in where the data becomes linearly separable, this layer is called the hidden layer. The drawback of MLPs is their computational complexity due to the fully connected forward graph. They can be used on any type of data and for any task such as image classification for example

on MNIST [12].

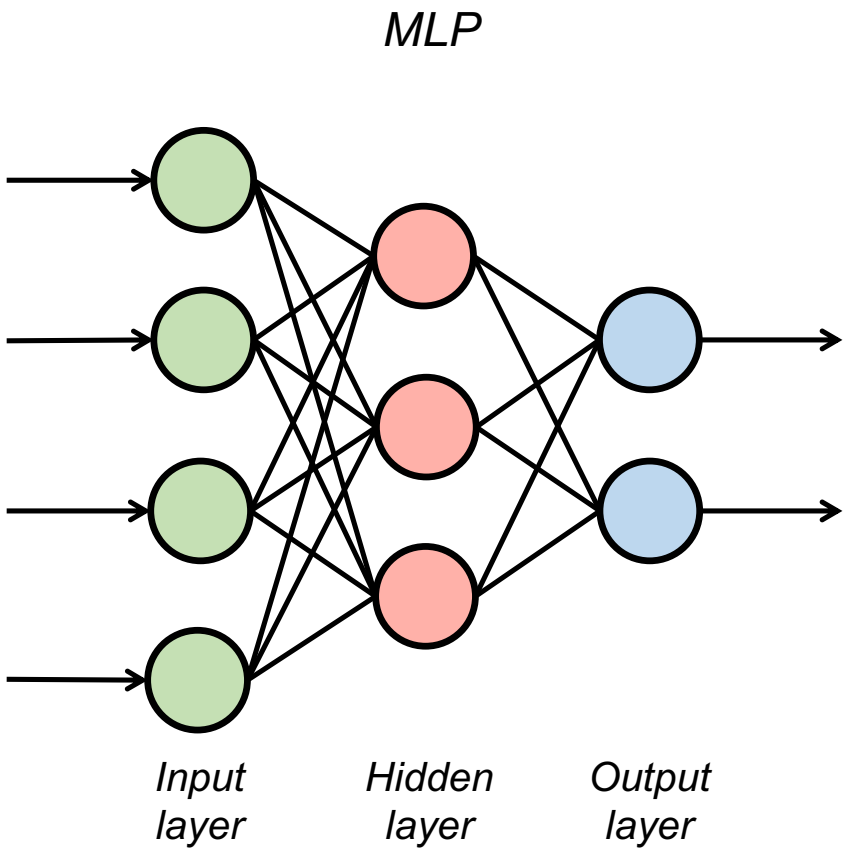


Figure 2.2: Regular Multilayer Perceptron (MLP)

The input of sentences into the VAE MLP encoder can be achieved by first embedding the sequence of tokens using either *one-hot* or *word2vec* embedding and next concatenating all embedded tokens within the sequence in correct order. Now we have a single feature vector of constant length for each sentence. This is very similar to the way we would feed an MNIST digit image into an MLP, by simply flattening the two dimensional image array into a one dimensional array of pixel values. We have implemented a VAE model with MLP encoder and decoder in the following way as shown in Figure 2.3 below.

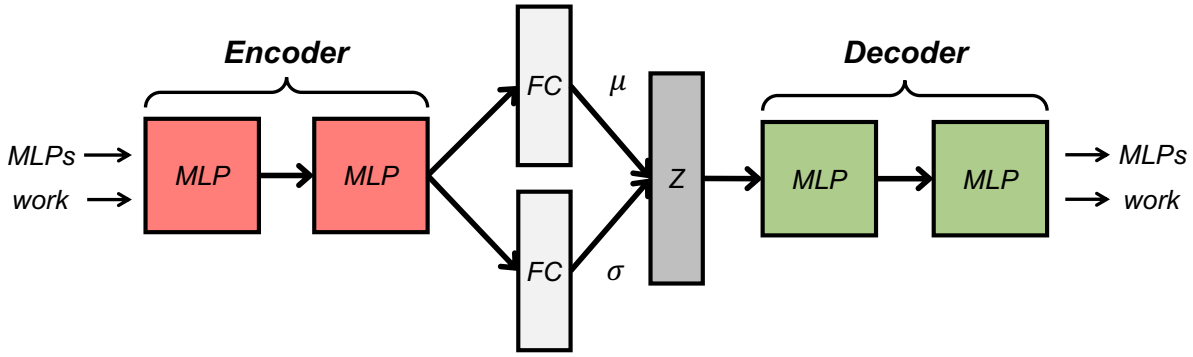


Figure 2.3: MLP based VAE

CNN

Convolutional neural networks (CNN) [13] is a neural network variation that is optimized for extracting important features from local regions in data such as a region of interest (ROI) within an image. The basic structure of such an CNN is shown in Figure 2.4.

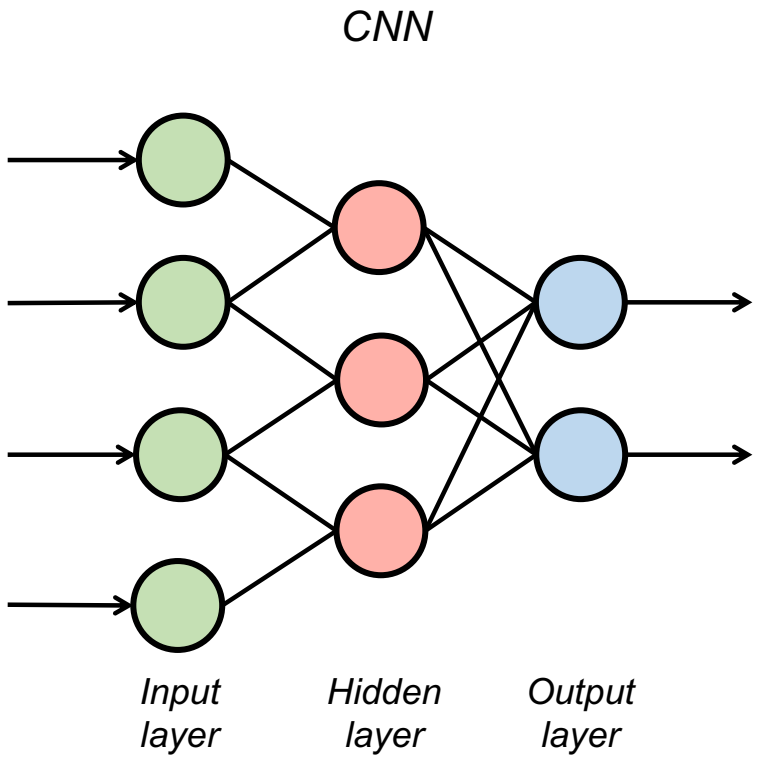


Figure 2.4: Simple CNN - with a pooling-like ‘operation‘

They are advantageous over MLPs for the task of feature extraction on images, as images themselves are a of high dimensionality which is problematic for MLPs which dedicate a single processing unit for each input pixel within an image and its own set of weights. Instead of learning weights for each unit, in a CNN weights can be reused, as they are contained and represented as convolutional *filters*. The convolution within a CNN is implemented using a filter or kernel, the terms are interchangeable. A filter is a learned function that transfers a spatial representation of input e.g. subset of pixels in an image, into a single value. This is done for all subset of pixels within the image until the filter has been moved over all pixels within the image. Moreover, CNNs use *pooling* in order to make the processing computation faster. Pooling is a subsampling method to reduce the dimensionality within the subsequent layers. The two most prominent pooling methods are *max-pooling* and *average-pooling*, yet for this thesis project we implemented CNNs without any pooling as the dimensionality remains manageable for our short sentences. Two-dimensional convolutions are employed for feature extraction of data samples with a spatial characteristics such as images. One-dimensional convolutions can be used for sequential data such as audio from speech or words in text which represent a sequence as it has be shown in [14, 15]. By stacking up different convolutional and pooling layers, the network will extract high level features which will serve the task at hand for instance face

recognition or object classification. The extracted features are invariant to translation within the input, such that the task can be solved regardless of the position of an object within the input image.

We have implemented a VAE model with CNN encoder and decoder in the following way as shown in Figure 2.5 below.

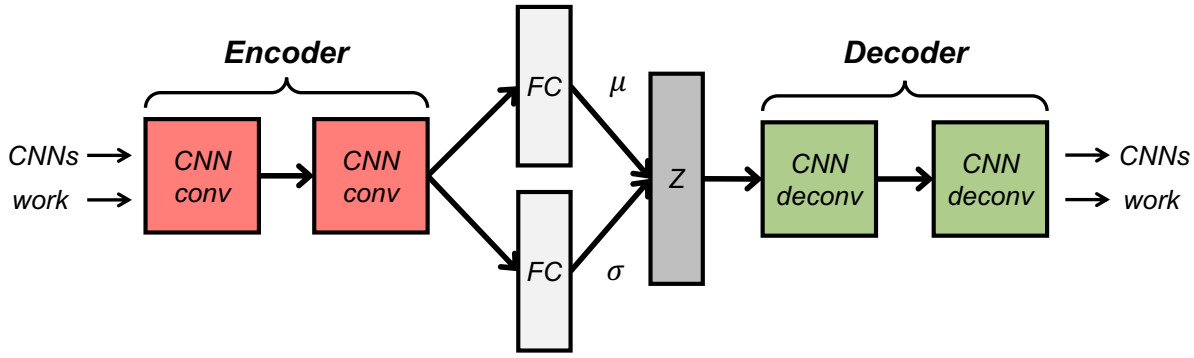


Figure 2.5: CNN based VAE

The input of sentences into the VAE CNN encoder can be achieved by reshaping the data into an appropriate shape which mimics the spatial nature which images usually have. That means, that when given with a sentence of length s and embedding dimension for each token within that sentence of d_{emb}, we have to reshape our feature vector from a single dimensional array of shape $(d_{emb} * s, 1)$ into a two dimensional array of shape (d_{emb}, s). In this representation form, we are feeding 'sentence-images' into the CNN encoder, where each column represents a single token. This means, that the input is a 2D object. Now that we know how to construct our data, we show how we apply the CNN, usually used on images, onto the 2D spatial text document as shown in the following Figure 2.6:

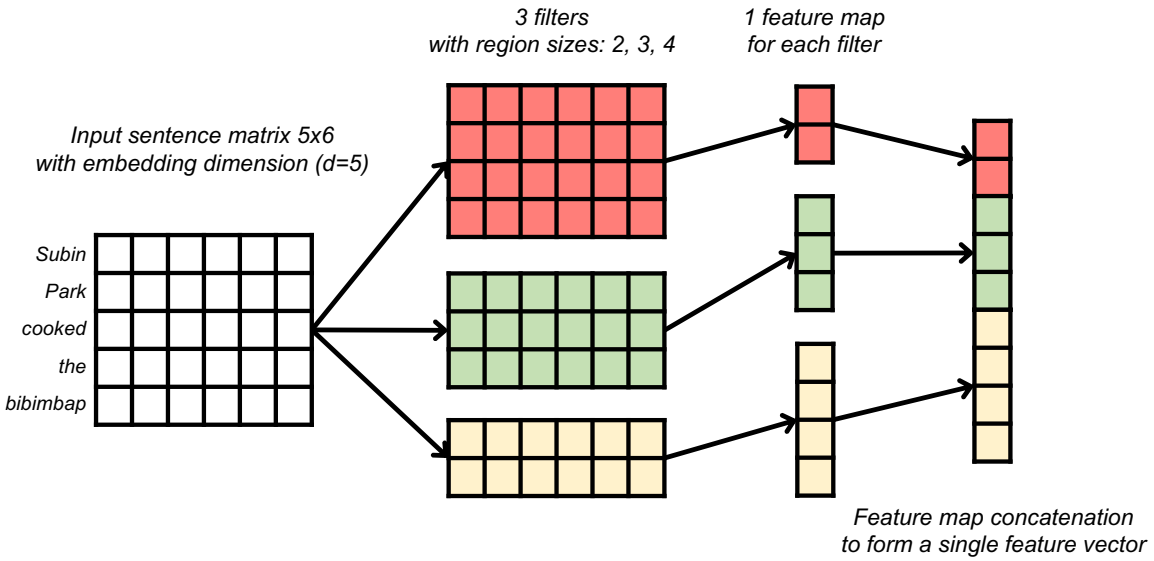


Figure 2.6: Our convolutional approach to sentences in 2D representation

This model makes us of 1D convolution as shown in 2.7 instead of the classical 2D convolution as shown in 2.8 and often applied on images for efficient feature extraction.

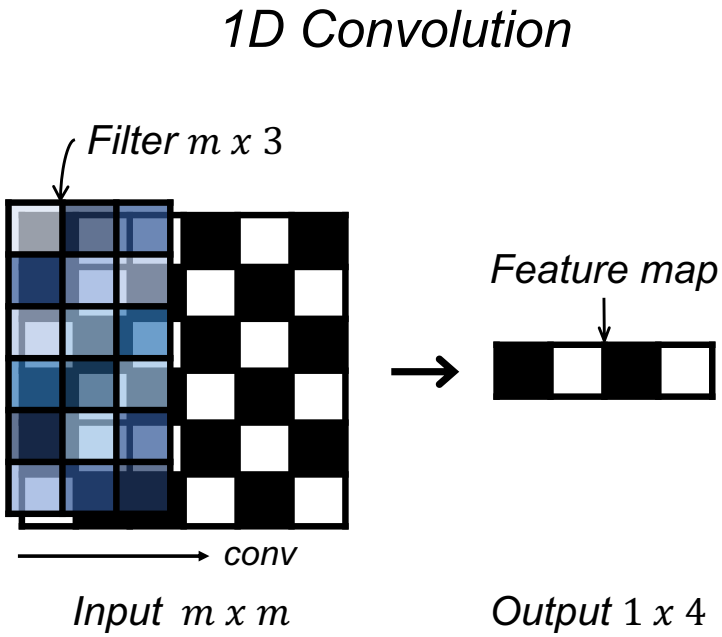


Figure 2.7: Text specific approach to 1D convolution with filter of height equal to embedding dimension of vocabulary words

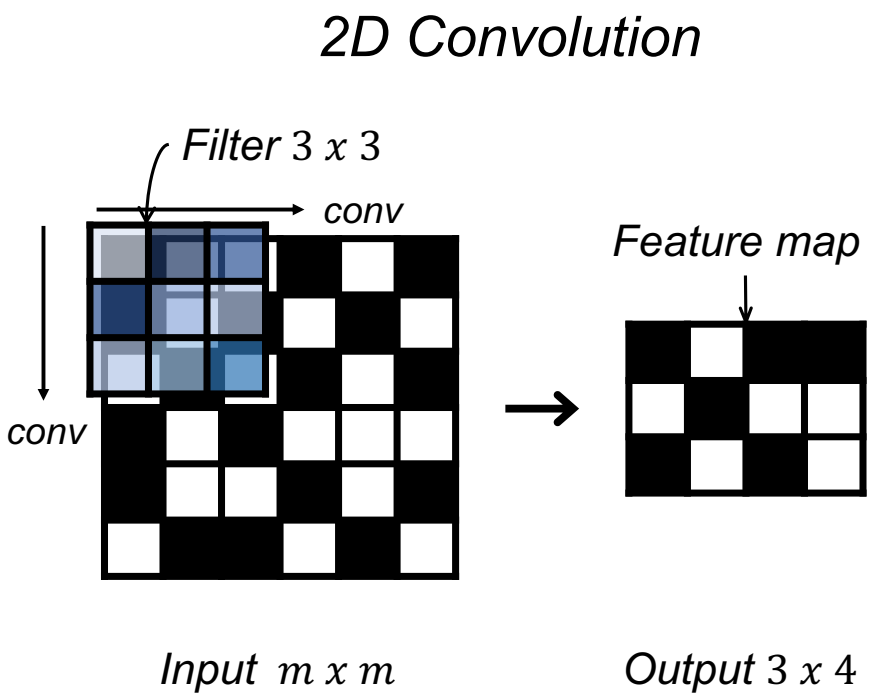


Figure 2.8: Regular 2D convolution with a quadratic filter

RNN

Recurrent Neural Networks (RNNs) are directed graphs dedicated to processing data, which has a sequential property, such as audio (represented as a time series) or text (represented as a sequence of tokens and steps). They are able to extract features which are changing over time and they can do this for an arbitrary input sequence length. In general, the goal of RNNs is to predict the next element within a sequence based on all previously seen elements of this sequence. We can build language models (LM) with RNN cells, where the next word of a text is predicted given all previous words of the text the model has already seen. RNNs are cyclic graphs, as they feed their output at step t as an additional input for the next step $t + 1$ over a self-connecting edge. One can think of RNNs as feedforward neural networks if we take these cyclic edges and unroll them by duplicating the RNN cell for each input step. The advantage of recurrent neural networks over CNN and MLP is the fact, that they can process different sized input sequences because they are not limited to a finite number of input nodes. The disadvantage of RNNs is the fact, that they process each token at a time and therefore can not make use of parallel processing as it is the case for CNNs and MLPs which expect the whole input sequence at once. This concludes, that due to sequential operation RNNs are slower because we have to wait for the current token to be processed and get the RNN cell‘s output in order to process the following token. Yet, there

are first approaches and solutions in getting RNNs to train as fast as CNNs as shown in [16].

The basic structure of such an RNN is shown in Figure 2.9.

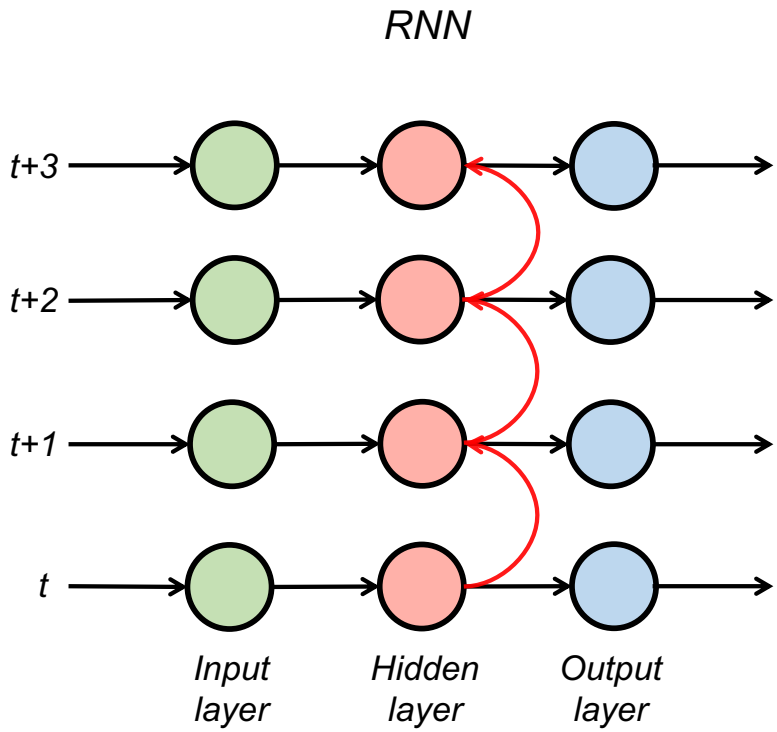


Figure 2.9: Simple RNN - with a cycle weight

The input of sentences into the VAE RNN encoder can be achieved by adjusting the representation of each token and by reshaping the inputs to an appropriate shape necessary to use recurrent neural network cells. First, we need to create a dictionary, which contains an index i for each word w within our dataset vocabulary V. With this index dictionary, we can represent each sentence of length L as a sequence of indices $i_1, i_2, ..., i_L$. Next, we use a pre trained *embedding lookup matrix* in order to convert the discrete indices into the corresponding row-vector $\mathbf{v}$ within the embedding lookup matrix of shape $(|V|, d_{emb})$, where again d_{emb} represents the dimension of our embedding space and $|V|$ is our vocabulary size. Finally, we can input a batch of b sentences into our RNN by parsing the batch of words in n-th position in each sentence at a time. This means, that the input of our RNN encoder is a 3D object of shape (b, L, d_{emb}). We have implemented a VAE model with LSTM encoder and decoder in the following way as shown in Figure 2.10 below.

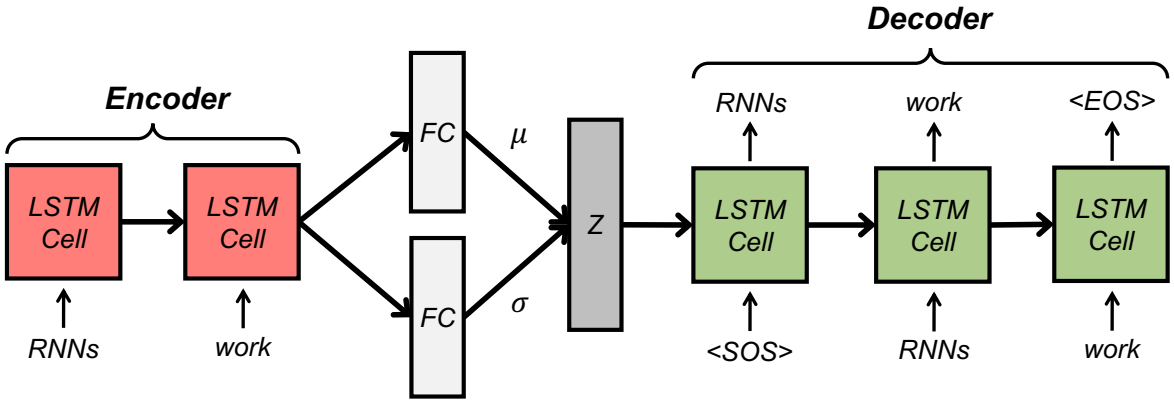


Figure 2.10: LSTM based VAE

There is a variety of RNN cells as listed below:

- vanilla RNN cells [17]
- Long short term memory (LSTM) cells [18]
- Gated recurrent unit (GRU) cells [19]

Some of them provide the ability to save information about the sequence in a dedicated ‘memory‘ which can be saved to and erased from at any given processing step. For text, this means that the recurrent unit has a stored memory of previously observed words. For our VAE RNN model we rely on LSTM cells, as they tend to perform best when comparing to the state of art of VAEs applied to NLP text generation problems [20, 21].

2.2 Natural Language Processing (NLP)

2.2.1 Overview

NLP is a sub field of computer science and artificial intelligence (AI) with roots in computational linguistics (CL). It deals with computational processing, automatic analysis and representations of natural language in variant forms such as text or audio. A few important tasks which can be solved with NLP are:

1. text summarization (e.g. generating a shorter version of a longer text document)
2. document classification (e.g. classifying news articles into a specific genre such as sport, politics or finance)
3. sentiment analysis (e.g. classifying a restaurant review as positive or negative)
4. machine translation (e.g. translating from source language A into target language B)

The field of NLP is moving from established statistical based methods [22, 23] to neural network (NN) based methods such as [24, 25]. For NLP problems many solutions use manual feature engineering of which special care must be taken. We need to make sure that the features encode everything that is informative, predictive, and meaningful for the final task we are trying to solve. Traditional approaches to NLP often apply a pipeline of preprocessing algorithms, each performing a different task such as part of speech (POS) tagging, removal of stopwords (e.g. 'the', 'is' or 'are') and stemming of words to their root word which we would usually find in a dictionary. We hope to produce a useful input for a downstream task, which we can subsequently feed our model with. The truth that different tasks may require information about different aspects of the raw input is an important reason for the motivation behind learning unbiased representations for natural language.

2.2.2 Representing words

Words can be represented using an embedding method of choice. An embedding of a word can be understood as a mapping from a discrete token to a vector representation. This word mapping into a vector space is useful, as it allows to work with continuous value based machine learning models using discrete data points such as text. In the following we will present two types of ways, in which words can be embedded and which we further examine within this project work.

One-hot embedding

For one-hot embedding, the representation of each word is a sparse vector. The encoding of a given word is simply the vector in which the corresponding element is set to one, and all other elements are zero. The dimension of the embedding space will be equal to the number of words existing in the dataset vocabulary. Using such an encoding, there is no meaningful comparison we can make between word vectors other than equality testing. One-hot embeddings have many other drawbacks despite the curse of dimensionality and the fact that the embedding space grows linearly with the amount of words in the vocabulary, one of themPutting together the one-hot representation of words and an embedding model, we can generate more meaningful representations of words by achieving dense embeddings.

Dense embeddings

One of the biggest achievements in current NLP advances is the idea and generation of continuous and dense word embeddings. One can think of them as shown in Figure 2.11 below:

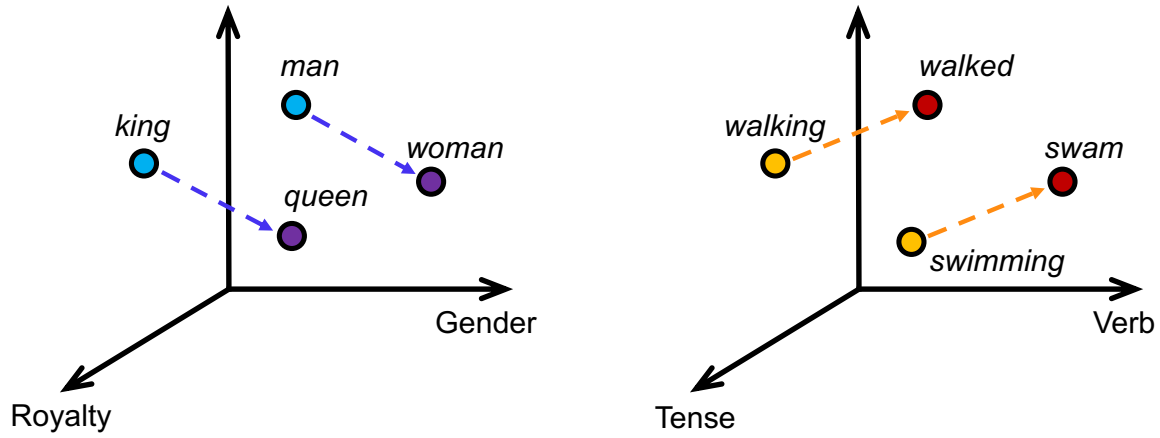
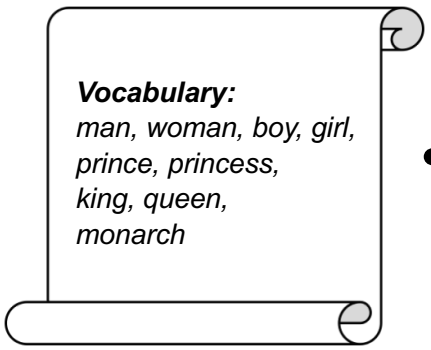


Figure 2.11: Visualization of the *word2vec* word embeddings

They solve the issue of dealing with text representations which are high dimensional and sparse. Thus, reducing the computational complexity of many NLP related problems as mentioned in the previous section. *Word2vec* is a particularly computationally-efficient model to train word embeddings from raw text documents. It comes in two variations namely, the Continuous Bag-of-Words model (CBOW) [26] and the Skip-Gram model [27] both proposed by Mikolov et al.. Word embeddings have the ability to generalize, due to semantically similar words having similar vectors, which is not the case in one-hot vectors (each pair of such vectors w_1, w_2 has a cosine similarity of $cos(w_1, w_2) = 0$, which means, that all words within the vocabulary are independent from each other, even if we would have synonyms or variations of a word which for instance are the result of conjugating a verb in different verb tenses. On the one hand, if a feature vector of a sentence is the concatenation of one-hot vectors of the sentence‘s words, we will only be able to consider features which we have seen during training of our model. On the other hand, when we use trained dense embeddings, semantically similar words appearing in the test set will share a similar representation with a synonym or related word within the training set.

In order to highlight the difference between what one-hot and *word2vec* embeddings look like for the same dataset. The following two Figures 2.12 and 2.13 will make the difference crystal clear.

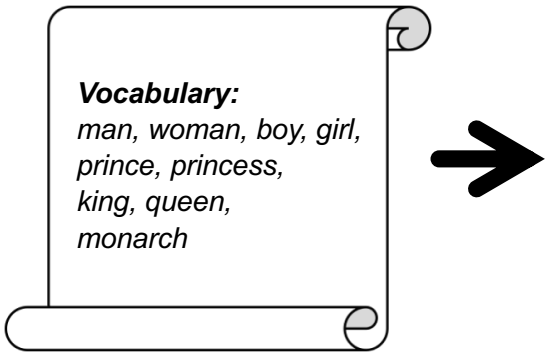


Each word gets a 1x9 vector representation

Dimension	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
king	0	0	0	0	0	0	1	0	0
queen	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Figure 2.12: One-hot embedded dataset consisting of 9 words

and the more dense variant



Each word gets a 1x4 vector representation

Dimension	1	2	3	4
man	-0.9	0.5	0.3	0.2
woman	0.9	0.5	0.3	-0.7
boy	-0.9	0.2	0.3	0.1
girl	0.5	0.2	0.3	-0.8
prince	-0.3	0.1	0.4	0.3
princess	0.6	0.2	0.4	-0.2
king	-0.9	0.6	0.7	0.8
queen	0.9	0.6	0.7	0.6
monarch	0.4	0.3	-0.3	-0.8

Figure 2.13: *Word2vec* embedded dataset consisting of 9 words

Looking at both Figures 2.13 and 2.12, we see two advantages. First, our representations are much shorter, in effect of smaller dimension. Second, similar words in our toy dataset have close or similar values when comparing them in the encode *word2vec* space.

For our work we used the helpful Gensim [28] Python package in order to create and

load the embedding matrix model of the pre-trained GloVe [29] word embeddings in a dictionary-like way. Gensim enables us to build a model with which we can simply look up the nearest neighbour word embeddings and it lets us easily calculate the *cosine similarity* as distance measure between two word embeddings which can be fed into the model by directly providing the corresponding strings of each word. For our models and experiments we used two different pre-trained word embedding models:

- 50 dimensional word vectors
- 300 dimensional word vectors

Both, the 50D and the 300D word embedding models are trained on a combination of Gigaword5 (4.3 billion tokens) [30] and Wikipedia2014 [31] (1.6 billion tokens) datasets which together contain a total amount of 6 billion tokens and a vocabulary size of $|V| =$ 400,000, which was determined by keeping the 400,000 most frequent words and filtering out all other less frequent words.

2.3 Representation Learning (RL)

Representation learning is a subfield of machine learning, where we try to automatically learn good features without a human being being involved as a feature designer. Good features can mean many things for instance removal of collinear features, denoising the raw inputs and achieving a better signal to noise ratio (SNR) while reducing the input dimensionality. Thus, learning representations of the raw data will make it easier to observe useful information when building any downstream model for task such as classification, generation or prediction. In the case of probabilistic models, a successful and meaningful representation is often one that captures the posterior distribution $q_\phi(z|x)$ of the underlying ground truth factors of the observed raw input. Currently, there is a strong interest in deep generative models such as VAEs [1] and GANs [32], where VAEs in specific try to learn such a posterior distribution which is a concise probabilistic formulation. These models use neural networks as feature extractors to capture complex and nonlinear relationships between raw input features. Both of these model architectures are unsupervised and expressive latent variable models. These latent variables are also called hidden representation, as they are the fundamental the core of representation learning. One of the main interests within this thesis work is to analyze and learn a representation for text from the sentence level as opposed from the word level as it is being done with *word2vec* and *fastText*. It is reasonable to assume, that each data sample within a dataset is constructed by some underlying set ground truth factors which represent the data sample in its most compressible form without any loss of information due to compression. In representation learning those ground truth factors are referred to as *generative factors* or factors of variation. There are three properties we are interested in when looking at any learned representations of a trained model:

2.3.1 Disentanglement

A representation can be called *disentangled* when for instance one single latent variable dimension of the latent representation is only sensitive to a single generative factors, while remaining almost invariant to changes in other generative factors of the input [33] as it is shown in the Figure 2.14 below.

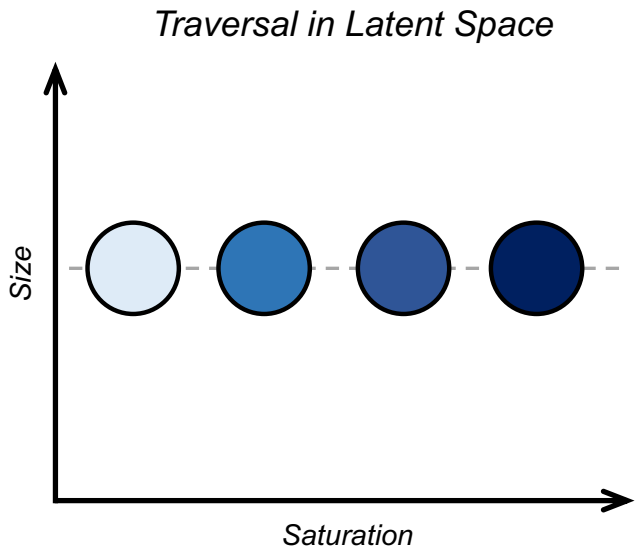


Figure 2.14: Disentangled Latent Space Traversal

Another example is, when given a set of images of a white circle in a black box, where the white circle can have a different location in each image. The model could learn, that the circle is always of the same size, therefore ignoring this information and simply encoding two two factors of variation which are x-position and y-position of the white circle within the black box. A disentangled representation would have the characteristic, that when changing the y-position of the white circle in the input image, the model‘s learned representation should only reflect this change in one of its latent variables and more than one. Therefore, for a representation to be disentangled it assumes, that the learned representation factorizes the latent causes of variation. The maximum disentangled representation for some dataset and its underlying manifold distribution can be achieved if and only if the ‘true‘ generative factors which are used to construct each data sample are equal to the latent representation. In short, a learned representation which factorises the underlying factors of variation, such that each latent (or latent variable) at maximum captures one generative factor can be called disentangled. *Disentanglement metrics* are a way to evaluate disentanglement quantitatively and will be explained in the following

chapter. On the other hand, an entangled latent space interpolation is shown in the Figure 2.15 where we can see, that it is difficult to tell the effect of dim_1 apart from dim_2.

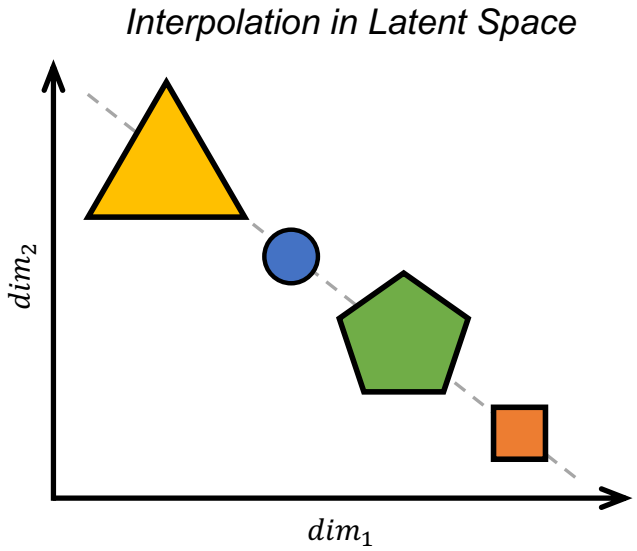


Figure 2.15: Entangled Latent Space Interpolation

2.3.2 Interpretability

A representation is meaningful and interpretable if human beings are able to understand the extracted features in a 'linear' way. This concept is best described by giving the following two graphical ideas.

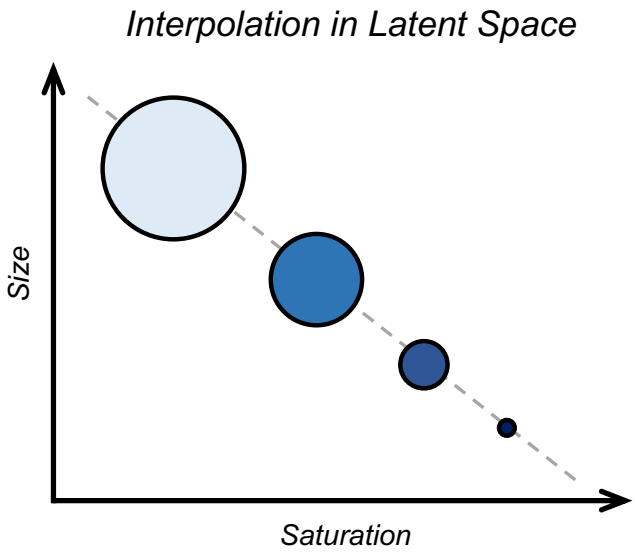


Figure 2.16: Interpretable Latent Space Interpolation

In Figure 2.16 the latent sapce is easily interpretable as one dimension clearly affects the saturation while the other changes only the size of the ball.

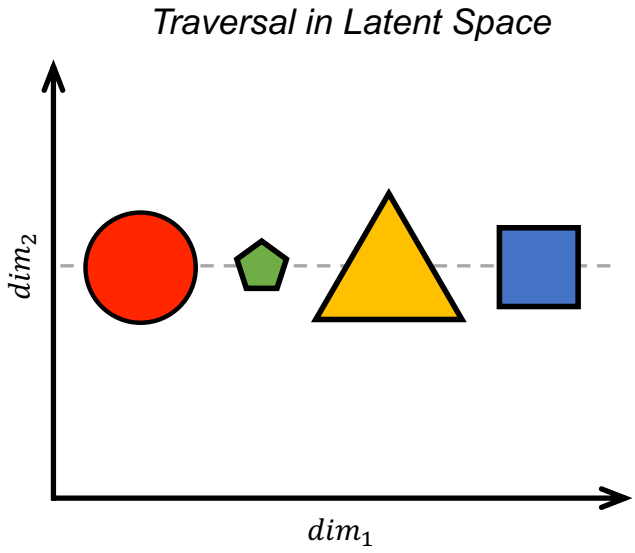


Figure 2.17: Uninterpretable Latent Space Traversal

Whereas in Figure 2.17, the latent space has no clear meaning, as we are only changing one latent dimension but despite that, many attributes changes at the same time. Most of deep learning and machine learning models extract entangled representations, which are highly complex and only helpful for the complex NN model which is able to process such an entangled representation with nonlinear functions to construct a final output and interpretable output for a specific task, such as image classification or text generation. To a certain degree, these entangled representations can be made interpretable by using machine learning visualization tools such as t-distributed stochastic neighbour embedding (t-SNE) plots [34] or performing simple vector arithmetics by adding two learned representations and reconstructing the resulting representation [35]. Finally, a representation which is interpretable should have live in a robust and smooth space. This means, that a smooth representation of a generative factor is one, that allows us to perform monotonic transformations in the representation space. An example of such a monotonic transformation would be moving up and down the white circle in the black box in a continuous, smooth and linear way without any abrupt changes when adding some small value to the latent variable. *Latent traversals* are a way to judge and evaluate the interpretability qualitatively and we will be explain them in the next chapter.

Chapter 3

Related Work

In this chapter we will uncover the source of our motivation, namely the current state of the art in the image domain with regards to generative models and disentanglement. Next, we will examine the topic of disentanglement and representation learning in NLP, what achievements have been made so far and what the next steps for generative models and NLP will most likely be.

3.1 Disentanglement in the image domain

3.1.1 Overview

Ground breaking ideas and state of the art Deep Learning models often originate from research within the image domain. One of these recent achievements was made in representation learning towards discovery of hidden factors of variation which help in understanding latent representation of CNN models. This is important, as there has been no incentive in exploring any factors of variation with all the success from CNNs and RNNs pushing state of the art performance for many engineering problems such as speech recognition and image classification. Yet, it is important to learn an explicit latent representation with features which are reusable, interpretable and independent, such that they could be used on other downstream tasks beyond classification. An example would be the generation of new labeled data as pair of (‘sample‘, ‘label‘). In that way, it is possible to leverage and make use the unlimited amount of unlabeled data. Of course, generative latent models will not factorize their latent representation without any explicit penalty which forces the model to find a compact and efficient representation of the input data distribution. Obtaining an abstract visual representation has been successfully shown with the DCGAN [36] model, which learns a hierarchy of representations from object parts to scenes in both the generator and discriminator of the GAN network as its basic structure. GANs other than

VAEs offer a powerful alternative to maximum likelihood techniques. Moreover, GANs do not need a specific cost function depending on the input domain as the discriminators task always stays the same for any given data domain, namely distinguishing between artificially and naturally (from original dataset) generated samples. VAEs perform worse in reconstruction and therefore generated samples are often blurry. In DCGAN, the authors present a model which allows to perform vector arithmetic within the latent representation to manipulate learned and meaningful properties of the input object. For example, the following arithmetic on images becomes $woman_{smile} - woman_{neutral} + man_{neutral} = man_{smile}$, so far we have only seen this for vector arithmetics on word embeddings such as the famous example $king - man + woman = queen$. Moving forward to more interpretable representation of images with notions of disentanglement, the work presented in DCIGN [37] shows how to learn a graphics code latent representation in a 3D graphics engine like fashion from unlabeled data with factors of variation such as rotation, lighting and pose. DCIGN uses the VAE formulation as architecture skeleton. All these models proved to disentangle as it was shown qualitatively through latent code manipulation and generation of new images. Still, the work done does not involve address the research problem of measuring the success of disentanglement and interpretability. Finally, InfoGAN [3] is an information-theoretic extension to the Generative Adversarial Network (GAN) which employs information theory to extract a meaningful representation by using a mutual information (MI) regularization term such the MI score between latent codes c and generator distribution $G(z, c)$ is optimized to be high.

Common datasets for analyzing learned latent representations and performing latent interpolations are shown in Table 3.1.

Dataset	**Factors of variation**
3D faces [38]	azimuth, rotation, lighting each attribute comes with 11 variations
2D sprites [39]	672 identities and body, gender, hair, armor, arm, greaves, weapon each attribute comes with 20 variations
CelebA [40]	10k identities, 40 binary attributes, 5 landmark locations
3D shapes [41]	4 shapes, 8 scales, 15 orientations, 10 floor, wall and object colours

Table 3.1: Commonly used datasets for unit testing of generative models

This is because for these datasets the ground attributes are well known in advance. Which becomes important when we want to quantify the disentanglement success and compare different generative models‘ performances against each other.

3.1.2 VAEs applied in the image domain

On the other hand, when we look at autoencoders (AE); their latent representation is compressed but unstructured and highly entangled. Only the autoencoder‘s decoder itself is able to understand its own latent representation as it can transform the learned features into a new space, where the data might become linearly separable. The basic autoencoder model is separated into two stages, namely inference and generation. The output of the inference model is our latent representation. When using an appropriate methods on the encoder output it becomes possible to find meaningful features of the input data. The main source of inspiration for this thesis work originates from work done with VAEs as generative models. The following gives an overview of state of the art generative models using VAE as skeleton structure. Mostly they differ in terms of encoder and decoder variations e.g. MLP versus CNN or additional terms in the VAE objective (ELBO) such that factors which encourage disentanglement are not being minimized but either maximized or taken out of the objective [42, 43] in order to improve the VAE representation.

- **InfoVAE:**
  The work in InfoVAE [2] tackles the issue of degraded inference (in effect the posterior $q_\phi(z|x)$) quality and the common problem of the latent variable which is ignored by the decoder. They mitigate these difficulties by modifying and proposing a new ELBO objective which enables to balance the trade off between correct inference and fitting the data distribution (likelihood). Both lead to a better generalization in the latent space and therefore less susceptibility to overfitting the train data set distribution. The key essence of this work is that the model always prefers to maximize the mutual information between the input x and the latent variable code z; regardless of the capacity of encoder and decoder. InfoVAE outperforms vanilla VAE by far when training on a small amount of data samples.

- **β-VAE:**
  In β-VAE [4] the authors introduce a hyper parameter β for the KL divergence term in the ELBO objective. In the case $\beta = 0$, the models tracks back to the classical AE setup and in case of $\beta = 1$ we get the vanilla VAE model with usual ELBO objective. β-VAE discovers totally new factors in the CelebA dataset such as *skin color*, *saturation* and a mix of *gender/age*. The latent space is improving its smoothness which can be seen through latent traversals and generation of new samples. A higher β value results in better disentanglement but also blurrier reconstruction and loss of representation of some factors of variation. This paper is the first to propose a quantiative evaluation tool of disentanglement by introducing a *disentanglement metric*. Having a quantitative measure of disentanglement is important in order to be able to compare generative models. Clearly, β-VAE shows that there is a trade-off

between disentanglement score . Moreover, their work elucidates the importance of data continuity and its effects on the quality of a disentangled representation.

In the follow-up work of β-VAE [6] the authors try to understand the disentanglement of their model. They adjust the ELBO objective and implement a target KL-divergence term, which defines the encoding capacity C of the model. This capacity C is increased during training to learn new factors of variation as soon as the reconstruction performances plateaus; for that they implemented the capacity increase with a scheduling scheme after a certain amount of epochs. The effect of this capacity term is, that it enables the model to learn *axis-aligned* disentangled representations of the generative factors of variation. Increasing the capacity C increases the KL-value for latent variable within the code and therefore also the total KL-loss term in the ELBO objective. Another interesting effect of capacity increase is, that the increase helps to make the reconstructions better, which means that reconstructions get sharper. The capacity value C can be explained as a measure of information content of the which is allowed to flow through the VAE bottleneck.

- **Factor-VAE:**
  Factor-VAE [7] improves upon β-VAE providing a better trade-off between disentanglement and reconstruction performance. They introduce a discriminator network (MLP), jointly trained with the VAE, in order to make use of the *density ratio trick* which enables to approximate the KL-term in the ELBO objective which now contains a *Total Correlation* [44] TC-term which directly encourages independence in the latent code distribution as it measure dependance of multiple random variables RVs. A low TC is necessary but not sufficient for disentangling of independent factors of variation. Moreover, they point out the weakness of the disentanglement metric proposed in β-VAE and suggest a new and more accurate disentanglement metric. The hyper parameter γ in front of the KL-term evaluates the divergence between two distributions namely the aggregate posterior $q(z)$ as expectation marginal distribution over the posterior $q(z|x)$ and the distribution which results from the product of marginals as $\bar{q}(z) = \prod_{i=1}^{d} q(z_i)$, where d denotes the number of hidden units in the latent representation z. This γ hyper parameter is used to encourage code independence. The work pinpoints that disentangling effects are sensitive to the number of code latents d. Finally, the new metric is free of hyper parameters because the classifier is a simple majority-vote classifier.

- **β-TCVAE:**
  In the work of β-TCVAE [5] the authors revisit and decompose the ELBO objective to show the existence of a term which measures the total correlation (TC) between latent variables. The argument is that the root cause of posterior collapse is the ELBO and thus the authors decompose the ELBO objective in order to understand certain behaviours when optimizing the KL divergence term hyper parameter in β-VAE. Their proposed model is a refinement and plug-in replacement of the vanilla β-VAE for learning a disentangled representation without introducing any new hyper

parameters during training. Furthermore, this work proposes a third a novel disentanglement metric which as opposed to the ones presented in Factor-VAE and β-VAE is a classifier-free measure. As their disentanglement metric is an information-theoretic based approach employing the mutual information between the ground truth factors $\mathbf{v}$ and the latent code $\mathbf{z}$ they call it the *Mutual Information Gap* (MIG metric). Metrics which are based on classifiers are ad hoc, meaning they only a solution designed to work for a specific problem or task, and thus not holistically applicable plus classifiers often introduce new hyper parameters which one needs to optimize for every new dataset and task. The newly introduced disentanglement metric is generalizable as it allows to measure disentanglement between any type latent-variables e.g. scalar or non-scalar. Their proposed MIG metric also detects axis-alignment, is unbiased for all hyper parameter settings, and can be generally applied to any latent distributions provided efficient estimation exists. Therefore we can use it onto discrete latent distributions. When a set of latent variables is not axis-aligned, each variable can contain a decent amount of information regarding two or more factors. The gap feature in the MIG metric heavily penalizes unaligned variables, which is an indication of entanglement. Next, this work points out the problem of β-VAE which is focused on holding the KL divergence term small. On the one hand, β-VAE penalizes the index-code mutual information, which is detrimental to learning a factorized latent code. On the other hand, β-VAE penalizes the total correlation and dimensionwise KL divergence, which is beneficial towards a meaningful representation. Therefore, the difference between those two β-VAE model variations is that with higher values of β this model consistently results in higher disentanglement score compared to to β-VAE. The key difference between the Factor-VAE and β-TCVAE is that β-TCVAE does not need an additional discriminator in order to evaluate the total correlation TC-term as FactorVAE does by leveraging the density ratio trick. The β-TCVAE model shows a clear and strong relation between total correlation and disentanglement score. The penalty on the TC-term forces the model to find statistically independent factors in the data distribution. They argue that a heavier penalty on this term causes each dimension of the aggregate posterior to learn statistically independent semantics in the data distribution, which induces a more disentangled representation. Finally, the authors show evidence for the hypothesis that large values of β can be useful as long as the index-code mutual information is not penalized! A lower penalization of the index-code MI in β-TCVAE helps finding new interpretable and semantic latent properties.

- **Categorical-VAE:**
  The work presented in [45, 46] approaches the problem of backpropagation through discrete nodes in the computation graph by introducing a new reparameterization trick for discrete latent bottlenecks. This is achieved by introducing soft discrete, also called *concrete*, a word composition of continuous and discrete. Both models use a one-hot bit representation for a single latent variable which is treated continuously during the training phase. Sampling from this discrete distribution as in VAEs boils

down to the Gumbel-softmax trick. In the beginning of training the variance of the computation graph gradients is low and biased and close to the end of the training time, the gradients‘ variance become more larger but unbiased by introducing temperature scheduling in the within the softmax operation. Being able to learn a latent representation which is discrete is meaningful as a lot of objects and problems in the real world are of categorical and discrete nature such as object classes when doing object classification but also tokens in a dictionary of vocabulary when working with text for NLP tasks such as sentiment analysis and topic modeling [47]. The only drawback of the presented categorical VAE models is that each latent code dimension must have the same amount of possible categories is can take on, which means, that all variables in the latent code are symmetric. This is a challenge where we have to work with categorical data which might consist of generative ground truth factors with highly asymmetric number of categorical values for each generative factor.

- **Joint-VAE:**
  The Joint-VAE [8] is a VAE model which consists, as the name suggests, of two types of latent variables namely continuous ones as we know them from vanilla VAE [1, 4, 6, 7, 5, 2] and discrete VAEs as first presented by Rolfe [48]. The framework proposed in this work clearly shows, that VAE are able to recognize generative factors and their underlying distribution for example categorical distribution or continuous gaussian distribution. The model is applied on MNIST [12] where it clearly shows, that the first generative factor to be discovered is the digit type, followed by digit angle and digit stroke width. This is because the order of detecting generative factors is dependent on the effect a learned latent factor has on the loss function and its improvement. With that regard, using a Joint-VAE with 10 continuous latent variables and one discrete 10-dimensional latent variable; the VAE learns to save information about the digit type, which is a categorical factor, into the single discrete latent variable where each digit type between 0 and 9 will be saved as a one-hot representation. This approach is capacity saving and an efficient encoded representation. Manipulating and generating new digit samples now becomes as simple as permuting the discrete latent variable during testing mode. Again, for reparameterization of the categorical posterior the Gumbel-softmax, as continuous approximation of a categorical distribution. The Joint-VAE comes introduces an inference network which makes it possible to infer the digit type on test samples by simply looking at the one-hot representation within the discrete latent variable and thus achieves an accuracy of up to 88.7 %. This enables to classify digits into clusters in a completely unsupervised fashion. Finally, it is important to note, that all fully continuous latent space VAE flavours are not able to achieve a disentangled representation for MNIST as the turn out to be unable to clearly map digit type into a single continuous latent variable. This might be due to the fact, that MNIST has an inherently discrete generative factor which is the digit type.

- **VQ-VAE:**
  The work in VQ-VAE [49] is another discrete latent generative model which learns

a discrete representation. The encoder outputs a continuous code and only by subsequent mapping with vector quantization (VQ) the code becomes discretized. The prior $p(z)$ in this model is autoregressive, which means, that it has to be learned during training rather than staying static as in most VAE models e.g. vanilla VAE with a $\mathcal{N}(\mu = 0, \sigma^2 = 1)$ prior. Samples are generated by first sampling a latent factor from the prior distribution and next decoding the sampled latent vector in order to generate a 'new' sample as output of the decoder network. Due to the discrete latent space, VQ-VAE is able to abstract away redundant information such as noise and features which remain locally same and do not change much a lot locally. This is achieved by aggressive compression of the continuous latent codes, thus reducing the dimensionality. This is implemented by a two-stage discretization, stacking two VQ-VAEs on top of each other connected via the output of the first VQ-VAE. The output of the decoder gets vector-quantized by mapping it onto the embedding space e which contains D rows as embedding dimension and K columns which define the possible values a single entry in the embedding vector can contain. For example, $K = 9$ means that a value can take on a number between 0 and 512 ($= 2^9$). Having discrete latents is relevant because many real world objects and phenomena are discrete. Finally, the model is less susceptible to posterior collapse, which happens, when latents from the encoder are ignored as the decoder is complex enough to model the input x and achieve a low reconstruction error. In this case, the model is not any more generative because the learned latent z has no real variational data generation effect. The drawback of this model is the fact, that the prior $p(z)$ needs to be trained in addition to the VQ-VAE architecture which is time inefficient.

3.2 VAEs and representation learning in NLP

3.2.1 Overview

The first thing to note is that within the text domain, there exists no dataset which has the property of knowing all ground truth generative factors of variation in a graphics rendering fashion way as mentioned in the previous section such as 2D and 3D shapes datasets or the dSprites dataset which is the equivalent of 'MNIST' of representation learning and disentanglement research. Therefore, it is clear, that disentanglement and representation learning in NLP is not on par with the state of the art within the computer vision domain. Yet, the success of ML highly depends on the right choice of the feature representation, thus we assume that this must be the same case for NLP. In effect, feature engineering and statistical preprocessing is very common in NLP. Therefore, the task of automating the task of extracting meaningful features from natural language as raw text data is the next step. One of these approaches is using word embeddings, which is an effective way of representing words in a low-dimensional vector space. These word embeddings are trained by learning in what context a specific word often appears and are trained on a

massive amount of text documents. Still, it turns out, that some linguistic features are kept better than other e.g. those word embeddings are helpful for POS tagging. Whereas, when it comes to analyzing the sentiment of the whole message, in effect the combination and order of the words within a text document or sentence, state of the art models have difficulties in doing. Thus, sentiment understanding of natural language is one of the remaining problems in NLP. For a human being though, telling the difference between positive and negative sentiment is intuitive and an easy task. For example give the sentence:

'I liked your product as much as I like being poked in the eye with a lollipop stick.'

Using state of the art natural language processing techniques and machine learning, it is highly likely that this sentence and other similar constructed sentences are being interpreted as positive sentiment instead of negative sentiment. Not only because the sentence contains the word 'like'twice, also does it contain the word 'lollipop' which is associated with other positive words such as 'pink'and 'glitter'. This experiement was done using the FastText library [50], an open source library for efficient text classification and representation learning recently published by the Facebook AI Research (FAIR). How would a computer know that humans do not like getting poked in the eyes with sticks, even if it is a lollipop? In requires a better representation of words not only in their context but also in the sentence structure and word composition in order to parse information correctly, and knowing that humans use comparisons in a sarcastic way, and that given the same words but a different structure, that sentence could have a totally different meaning. In the following we will dive more into the application of VAEs for NLP problems and what current results look like and give some examples of task that can be solved with text based VAEs.

3.2.2 Generative models applied in the text domain

VAEs applied on text documents had some success in learning a latent space which can be used to generate new sentences of variation. It turns out the be more challenging than for image data, this is because written natural language is a sparse and discrete data space. Many adjustments have to be done as most generative models only accept dense and continuous data space input. In the following we give an overview of what type of model architecture have been studied and for what tasks they can be used.

- **Style transfer:**
  The model presented in [51] focuses on style transfer given a dataset of non-parallel text, where the task is to separate the content from other aspects such as style. In NLP most tasks that involve next text generation are developed based on parallel corpora. Parallel text can be thought of a dataset which consists of pairs of text corresponding to each other in terms of content, translation or meaning [52]. The

authors propose two new constrained AE variants, namely an aligned AE and a cross-aligned AE. The cross-aligned AE directly aligns the transferred samples from one style with the true samples from the other style. This approach uses two discriminator networks, one which distinguishes between real sample x_1 and transferred sample x_2 and the other discriminator does this between real sample x_2 and transferred sample x_1. The discriminators D_1 and D_2 are implemented using CNNs by adjusting them for the task of sequence classification. There is the assumption, of having two different corpora X_1 and X_2 with different styles but same content. The ultimate goal of this model is to learn the two mappings $f_1 : X_1 \to X_2$ and the reverted function $f_2 : X_2 \to X_1$. The transferred sentences from one style should match example sentences from the other style. The style transfer is demonstrated on three tasks: Sentiment manipulation, recovering word substitution as a type of manipulation of the source sentence and recovery of word order given a scrambled sentence. Finally, the authors point out, that is its crucial yet difficult to evaluate and compare different models in terms of style transfer quality and success due to the ambiguity and different subjective cognition of language for every individual. Due tho which the authors in addition to quantitative numbers also evaluate their results on human perception and opinions about the performance of the new model where they use TextCNN [14] as a baseline model. The model struggles to keep the content similar because the model architecture has no term which explicitly enforces content similarity between source and transfered sentence. Clearly, the latent space entangles content and style.

The work presented in [53] proposes multiple improvements upon encode-decoder frameworks for *linguistic attribute transfer* such as sentiment or source language. Current, state of the art machine translation models rely on large parallel datasets, which are expensive to generate and do not help in learning a disentangled representation for written language in general. Most of written language exist in form of non-parallel text which reflects real world scenarios more properly. The authors point out, that for many models, it is not possible to use *maximum likelihood estimation* (MLE) for such problems as there is no ground truth data existing. Moreover, textual data is sequential and often appears in varying lengths, whereas images have a defined constant size and are continuous valued. The key achievement of this work is the introduction of a new *noun preservation loss* to ensure proper content transfer. The goal is to transfer the style while preserving the content of the original sentence. Along with the content preservation loss, the authors propose a *content preservation metric* in order to compare with other style transfer models if and how much content was lost due to style transfer. To sum it up, the work is pushing forward the task of transferring the style while maintaining the source sentence content. They give an outlook what other transfers could be tackled such as *professional to colloquial* language or *pronoun possession switch* e.g. from ‘my car‘ to ‘your car‘.

Other ideas in style transfer have be presented in [54] where the goal is to sample a sentence and try to convert it into a more natural sentence. The problem with that approach is that the model does not preserve the intrinsic similarity plus the style transfer is not a direction process from style A to style B, it rather gradually changes the style from adjusted and more 'natural' style A to a subsequent adjusted and even more 'natural' style B in a recursive fashion.

- **Controlled text generation:**
  The authors of [55] present a conditional VAE model for text. The goal is to make a step forward in terms of being able to generate sentences with specific features, whereas previous models such as the one presented by Bowman et. al [20] are only able to generate sentences which are to high extent randomized and uncontrollable with respect to their attributes. A discriminator network is used to impose desired attributes on generated samples and disentangle them from the latent representation which is produced by the encoder network of the VAE. They show how to control two different attributes, sentiment and tense. Though no corpus with sentence tense and notations is available, this model is able to learn from labeled words and subsequently generates the desired attributes for the sentence. One has to have two separate datasets labeled with the desired attributes, here tense and sentiment. That means there is no need for a dataset or corpora which contains both attributes at the same time for each sample. Given only word-level annotations the model successfully transfers the knowledge into sentence-level. In their work, the generated samples are used to augment the sentiment dataset and train an additional sentiment classifier. The semantic of tense is learned only from a lexicon without complete sentence examples. This approach combines VAEs with attribute discriminators and imposes explicit independency constraints on attribute control, which promotes a less entangled latent code. The generator and the discriminator form a pair of collaborative learners and provide feedback signals to each other. The novelty of this model is to make use of a split latent representation and leveraging the newly generated samples during the training phase with the help the help of a VAE with a conditional latent space. Moreover, the latent space is split into a structured (disentangled) and unstructured (noisy and thus entangled) part. The structured part, denoted as c, of the latent space presents specific attributes, explicitly known because we have their corresponding labels in the two corpora. The unstructured part, denoted as z, is trained and will contain attributes, which are difficult to define explicitly and where it is unclear how to denote their specific attribute and existence.

- **Discrete and interpretable sentence representation:**
  The work in [56] focuses on introducing a discrete representation learning model for sentences because it is easier to interpret than continuous ones plus language is often divided into categories. For example topic, time, singular/plural. A continuous latent space hinders humans to understand the generation process of a sentence.

Their model is based on the VAE architecture. Moreover, they rely on a discrete latent space and therefore make use of the in the previous section mentioned method of Gumbel-Softmax and vector quantization to train discrete variables. The key contribution of their work is to mitigate the problem with posterior collapse which was first analyzed and explained in [57]. The authors argue, that posterior collapse issue lies in the ELBO objective and thus decompose the KL divergence term to understand its behaviour. They show, that the KL-term contains the mutual information MI between latent code $\mathbf{z}$ and input data $\mathbf{x}$. In the past VAE models were also reducing the mutual information latent variables and the input data, which explains why VAEs often ignore the latent variables. Especially, when the decoder $p_\theta(\mathbf{x}|\mathbf{z})$ is powerful model. A solution to correct the anti-information issue is to maximize both the data likelihood lower-bound and the mutual information MI between latent code $\mathbf{z}$ and the input data $\mathbf{x}$. Intuitively a good generative model should achieve low perplexity (PPL) and low KL divergence, and simultaneously achieve high mutual information $I((\mathbf{x}, \mathbf{z})$.

- **Alternative use of latent space as editing representation:**
  The authors of NeuralEditor [58], propose a model which differentiates between lexical and semantic similarity of sentences. The idea of lexical similarity can be used to describe the overlap between two sentences s_1 and s_2 and is measured by the word Jaccard similarity. For example, it is possible to construct a sentence with small lexical distance that still differs very much from the source sentence by simply inserting the word 'not'. The edit vectors in this work are a sum of word embeddings and therefore, measuring the distance between two edit vectors can be done in the same way as measuring the distance between two word embeddings, namely by using the standard cosine similarity. The authors use PPL as quantitative comparison tool between different generative models and for qualitative comparison they use human evaluation of quality and plausibility of generated sentences. The key achievement of this work is to learn a latent vector space in which editing operations can be performed and which therefore does not contain any information about the input sentence. The difference between sentence VAE (SVAE) proposed by Bowman et. al [20] and NeuralEditor is that SVAE encodes the semantic structure into the latent space. Therefore, the latent vector in SVAE represents the entire input sentence rather than just editing information. NeuralEditor is focusing on two goals, semantic smoothness and consistent bahaviour. Semantic smoothness means, that an edit operation should alter the semantics of a sentence only by a small and well-controlled amount, while multiple edits should make it possible to achieve a large semantic change. Consistent behaviour refers to the property, that when we apply the same edit vector on different sentences, the NeuralEditor should perform semantically analogous edits across sentences. The decoder $p(\mathbf{x}|\mathbf{x}, \mathbf{z})$ applies the edit vector $\mathbf{z}$ onto the sampled prototype sentence $\mathbf{x}$ using attention to output the edited sentence $\mathbf{x}$. NeuralEditor is compared with SVAE. Editing in SVAE is implemented as perturbation of the latent sentence vector representation of the input sentence.

The results show, that SVAE either produces the same output (which hints to the fact, that the RNN decoder actually is ignoring the latent representation z) or totally unrelated sentences. On the other hand, NeuralEditor produces consistently smoother sentences with a high frequency of 'good' paraphrases (human evaluated) of the sampled input sentence. The vectors $\mathbf{x}$ and $\mathbf{z}$ are concatenated into a single feature vector before being fed into the decoder.

- **Other text generating models with interpretable latent representation:** In the paper 'Learning to Generate Reviews and Discovering Sentiment' [59] the authors inspected the relative contributions of features on various datasets using an RNN with a single layer LSTM of 4096 hidden units. The latent representation in this work is therefore the last hidden state of the LSTM cell and their model does not employ any generative model such as VAE or GAN. Nonetheless, they discovered a single unit within those 4096 units of the LSTM cell, with which they are able to classify the sentiment with an accuracy of 92%. Without even considering the extracted feature values of all other 4095 units. The model recovers the concept of sentiment in a very precise, disentangled, interpretable and most of all easily manipulable way. Thus, the single unit can be used for a classifier to cluster reviews into positive and negative groups. Therefore, the active unit is denote as *sentiment unit*. It boils down to a concept in which negative reviews have a value around -1, whereas positive reviews have a value closer to +1 and optimizing a scalar threshold which separates both classes.

The model presented in [20] generates sentences from a continuous latent space using a VAE LSTM architecture and hence comes closest to what we are trying to achieve within this thesis project. The difference between vanilla LSTM models and a VAE based LSTM models is that for a standard RNN language model (LM) each word within a sentence is conditioned on all previous words and the current hidden state. This means, that the model only looks at the relationship between subsequent words and it does not contain or encode any global features of the text such as sentiment for instance. The presented VAE LSTM consists of an LSTM encoder, which task is is to extract global features by mapping the variable length input sentence onto a fixed length latent space and a second LSTM decoder, that converts the intermediate representation back into some sentence, ideally one that is close to the input sentence. The authors observe an issue of VAE LSTM, which we will also observe in our model architecture when working with LSTM encoder-decoder. The decoder is complex enough to create its own language model LM and thus ignores the latent representation $\mathbf{z}$. To mitigate this problem the authors suggest two types of fixes:

1. Annealing the KL divergence term of the VAE at the beginning of the training by increasing it slowly from 0 to 1. In effect, this means that their model starts as standard AE and by the end of the training becomes a vanilla VAE

2. Using word dropout to weaken the LSTM decoder network. This is done by

randomly replacing the predicted output (which the LSTM decoder feeds itself with from time step t to time step $t+1$) with the standard token UNK, which is used for unknown words. This makes the decoder LSTM dependent on using and understanding the latent information z extracted from the encoder LSTM in order to mitigate the corrupted decoder input.

The latent code in this model captures features such as number of tokens (in effect sentence length) and broad topic information. Still, the latent linear interpolation between two latent codes $\mathbf{z_1}$ and $\mathbf{z_2}$, which the authors refer to as *homotopy*, clearly shows that the learned representation is extremely discontinuous and sharp. Moreover, the generated sentences clearly result in grammatically wrong sentences which have nothing in common with correct and most of all natural language. This homotopy is known from the image domain of applied VAEs as *latent traversal*. In case of sentences and text domain this method gives an insight into what neighborhoods in code space look like, how the VAE organizes its compressed input and what it understands with regards to a linear manipulation between different sentence latent codes. Given two sentences and a range of $b \in [0, 1]$ intermediate representation are given by the function $\mathbf{z}(t) = \mathbf{z_1} * (1 - t) + \mathbf{z_2} * t$.

It remains a challenge to perform latent space manipulation with smooth latent code and preserve other aspects of the sentence such as content of the source sentence. This is because the goal of controlling one aspect in a sentence such as writing style can only be done, when the latent representation is able to independently separate the style from the content. It turns out, that already the discrete property of natural language does hinder researchers to apply state of the art models which are successful in disentanglement of visual concepts within the image domain. Also can we see, that reaching out for metrics which evaluation which employ human perception and judgement of the sentence quality is still on going and performed due the complexity of written natural language and its ambiguity and special case rules which one has to consider. Many VAEs try to fill up and condense the latent space with meaning by encoding a representation which contains grammatically wrong sentences just to ensure, that the latent space is smoothly populated. This gives raise to the question if any generative model for NLP of a continuous one which is artificially densely populated. We will examine this question with two architectures and experiments in the following chapters.

Chapter 4

New Disentanglement Dataset for NLP

For research on disentangled representation learning in computer vision and images, Google DeepMind has created and open source the so called *dSprites* [60] dataset in 2017. This dataset contains a set of two dimensional black and white images of objects with different generative factors such as shown in Table 4.1.

Generative Factor Number	**Generative Factor**	**Latent Values**
1	shape	heart, ellipse and square
2	position X	32 values in range $[0, 1]$
3	position Y	32 values in range $[0, 1]$
4	rotation	40 values in range $[0, 2\pi]$
5	scale	6 values in range $[0.5, 1]$

Table 4.1: Generative factors and latent values of the dSprites dataset

The dSprites dataset has a total of 737,280 2D-images generated via the Cartesian product of all five generative factors (latents) $v_1, v_2, ..., v_5$. All possible combinations of these latents are present exactly once.

The dSprites dataset serves as the first choice for unit tests of disentanglement of unsupervised generative models. Many newly published latent variable models within the last year [61, 62] started to use this dataset. This makes sense, as the generative factors are an instruction of how to generate any image by providing the five-tupel of generative

factors and the value each generative factor takes on. In this regard, learning the inverse mapping from sample to source generative factors is the key task we are trying to solve in disentangled representation learning. The knowledge of these generative factors is of course unknown for complex data distributions such as real world images or natural language. But in order to be able to have full control of the generative latent models, it is helpful to assume, that the generative factors are well known in advance. It will help us to get an idea of what independent generative factors a model has learned and if the latent representation comes close to the generative factors representation. This idea of unsupervised learning of factorized and independent source factors of variations has its early roots from ideas and work published Schmidhuber in 1992 [63].

Current generative models applied on VAEs most often work with datasets such as:

- Large Movie Review Dataset (IDMB) [64]
- Stanford Sentiment Treebank-2 (SST) [65]
- Twenty Newsgroups Data Set [66]

All of them are big datasets with meaningful features a latent model could learn such as sentiment, genre or review length. Still, they miss an important property which comes with the dSprites dataset, namely:

1. Knowledge of ground truth generative factors
2. Independence of ground truth generative factors

Due to these reasons and in order to have a dataset on which we could experiment deep learning disentanglement methods and employ current state of the art disentanglement metrics presented in Factor-VAE [7], DIP-VAE [62] and β-TCVAE [5]. All of these disentangling VAE variations are in need of the knowledge of ground truth generative factors knowledge in order to evaluate disentangling success quantitatively. Thus, we decided to construct our own dataset, which we call in analogy to the dSprites dataset: *dSentences*.

In the following we will present how we constructed the dataset, what technical tools we made use of. Next, we give an overview of the generative factors we decided to implement, keeping in mind to maintain independence between all of these factors and finally a short overview of how the sentences can be generated from a given sentence-latent representation of generative factors.

4.0.1 Dataset construction

For the construction of the dataset we had to first come up with an idea of what independent generative factors of simple but still natural language could look like and how we can include

the three main properties of natural language namely

- grammar
- sentence structure
- syntax

Next we needed to manually construct a list of *(verb, object)* tuples which together generate some syntax in each sentence. For example, the (verb, object) tuple (‘eat‘, ‘chair‘) does not generate any meaningful natural language syntax as eat and chair are not directly related to each other in common and simple sentences. On the other hand, if we replace the object ‘chair‘ in the verb-object tuple with ‘fruit‘, then we are able to construct a sentence with syntactical property such as ‘I will be eating the fruit.‘.

After we have manually created a list of 1100 verb-object tuples, we can start the automated part of the sentence construction pipeline. For that, we wrote a deterministic sentence generator script which follows grammatical rules and a correct sentence structure guideline. The main clauses are procedurally generated using nine different generative factors variables. Next, each sentence representation (a vector of generative factor values $\mathbf{v} = [v_1, v_2, ..., v_N]$), is parsed into the sentence generator script and the construction happens in three steps:

1. We extract the base verb form by looking at the value of the verb-object-tuple generative factor at position index 0 of $\mathbf{v}$. Next, we conjugate the verb into all necessary variations. After that, we decide whether the sentence is a statement or a question. Then we determine if the verb needs to be in progressive form or not. Finally, we determine the sentence tense from the corresponding generative factor and if the sentence in negated or not. All case evaluations will lead to separate strings of the sentence we will be constructing. We save these strings for later processing.
2. In the second step, we extract the plain object by looking at the value of the verb-object-tuple generative factor, again at position index 0 of $\mathbf{v}$. The object needs then to be adjusted to the plural version in case the generative factor value for plural verb of the object is set. At the end, we add the definite article ‘the‘ in front of this string, to make the sentence more ‘natural‘.
3. Finally, we combine the strings from the first step with the string from the second step in order to receive a complete and correct sentence.

We used two helpful python packages namely NLTK [67] and Pattern [68] for natural language processing tasks which does alleviate to work with irregularity and ambiguity in natural language such as:

- generating the plural version of an object (e.g. ‘car‘ $\leftrightarrow$ ‘cars‘)
- conjugating a verb into singular third person (e.g. ‘do‘ $\rightarrow$ ‘does‘ and ‘be‘ $\rightarrow$ ‘is‘)
- creating the progressive form of a verb with ‘ing‘ (e.g. ‘drive‘ $\leftrightarrow$ ‘driving‘)

4.0.2 Data properties and generative factors

We determined nine generative and independent factors of which we could sample in order to reconstruct a basic sentence. The Cartesian product of all possible combinations of our generative factor values results in a dataset size of $N = 576{,}000$. The dataset size is largely affected by the most asymmetric generative factor, which is the verb-object-tuple with 1100 different values of variation. Our generative factors can be categorized into three distinct sentence level attributes. The table 4.2 below gives an overview of the generative factors which contribute to generate a sentence deterministically and what property of language each generative factor attributes to.

4.0.3 Getting from generative factor values to sentences

Below in Table 4.3 we give an idea of how the sentences are represented as a ground generative factor vector $\mathbf{v}$ and what the sentences in the dataset look like once they are constructed from $\mathbf{v}$. Assuming the verb-object tuples used are located at the dataset indices $i, i+1$, where the index i represents the tuple (‘eat‘, ‘fruit‘) and the index $i+1$ represents the tuple (‘cut‘, ‘fruit‘) we would retrieve the sentences in the following way:

Generative Factor Number	**Generative Factor Name**	**Latent Values**	**Sentence Attribute**	**Influence on S,V,O**
1	verb-object-tuple	1100 variations of $(verb, object)$ e.g. (‘eat, fruit‘) or (‘drive‘, ‘car‘)	syntax	verb, object (V,O)
2	grammatical number object	2 categories with values from $\{singular, plural\}$ e.g. ‘car‘ $\leftrightarrow$ ‘cars‘	grammar	object (O)
3	sentence type	2 categories with values from $\{interrogative, declarative\}$ e.g. sentence is a question (‘ ? ‘) or a statement (‘ . ‘)	structure	-
4	gender	2 categories with values from $\{male, female\}$	grammar	-
5	grammatical number subject	2 categories with values from $\{singular, plural\}$ e.g. ‘I, you, he, she‘ for singular subject	grammar	subject (S)
6	grammatical number person	3 categories with values from $\{1st, 2nd, 3rd\}$ e.g. ‘he, she, they‘ for 3rd person	grammar	subject (S)
7	sentence negation	2 categories with values from $\{affirmative, negative\}$ e.g. ‘I will not eat the fruit.‘ $\leftrightarrow$ ‘I will eat the fruit.‘	syntax	-
8	tense	3 categories with values from $\{past, present, future\}$ e.g. ‘did eat‘, ‘eat‘ and ‘will eat‘	grammar	verb (V)
9	style	2 categories with values from $\{progressive, not-progressive\}$ e.g. ‘eat‘ $\leftrightarrow$ ‘eating‘	grammar	verb (V)

Table 4.2: Generative factors and latent values of the dSentences dataset

Ground truth generative factor values v	**Generated sentence**
[$(verb, object) = \text{i}$, $obj_{sing./pl.} = 1$, $sent_{type} = 1$, gender $= 0$, $subj_{sing./pl.} = 0$, $subj_{pers} = 2$, $sent_{neg} = 1$, $verb_{tense} = 1$, $verb_{style} = 1$]	'Is he not eating the fruits'
[$(verb, object) = \text{i}$, $obj_{sing./pl.} = 1 \rightarrow 0$, $sent_{type} = 1 \rightarrow 0$, gender $= 0$, $subj_{sing./pl.} = 0 \rightarrow 1$, $subj_{pers} = 2 \rightarrow 0$, $sent_{neg} = 1 \rightarrow 0$, $verb_{tense} = 1$, $verb_{style} = 1$]	Change generative factor values
[$(verb, object) = \text{i}$, $obj_{sing./pl.} = 0$, $sent_{type} = 1$, gender $= 0$, $subj_{sing./pl.} = 0$, $subj_{pers} = 0$, $sent_{neg} = 0$, $verb_{tense} = 1$, $verb_{style} = 1$]	'We are eating the fruit'
[$(verb, object) = \text{i} \rightarrow \text{i+1}$, $obj_{sing./pl.} = 0$, $sent_{type} = 1$, gender $= 0$, $subj_{sing./pl.} = 0$, $subj_{pers} = 0 \rightarrow 1$, $sent_{neg} = 0$, $verb_{tense} = 1 \rightarrow 0$, $verb_{style} = 1 \rightarrow 0$]	Change generative factor values
[$(verb, object) = \text{i}$, $obj_{sing./pl.} = 1$, $sent_{type} = 1$, gender $= 0$, $subj_{sing./pl.} = 0$, $subj_{pers} = 0$, $sent_{neg} = 0$, $verb_{tense} = 0$, $verb_{style} = 0$]	'You did cut the fruit'

Table 4.3: Sentence encoded as generative factor values and its reconstruction into a string

Chapter 5

Variational Autoencoders (VAE)

In this chapter we will first give an overview on generative models and next introduce the VAE as such as model. The VAE framework serves as backbone model for all our model implementations and variations with different encoder and decoder blocks but also with variations in latent spaces such as continuous or discrete ones. Moreover, we will show how disentanglement of VAEs can be evaluated. For that, we will explain two main strategies, one is of qualitative nature while the other is of quantitative meaning. Regarding the quantitative metrics we showcase two state of the art metrics which leverage different tools from statistics respectively information theory.

5.1 Overview generative models

Generative models are one one the most potential approaches towards the goal of understanding the underlying distribution of datasets from different domains such as image, text or audio.This class of models is able to create (generate) new data samples which are similar to the data samples the model has seen during training. The generative models are forced to discover and efficiently learn the underlying distributions and properties of the data in order to reconstruct or generate similar samples. We can think of generative models as a machine which is able to look at any object for example a car and by inspecting a huge amount of sample cars the model finally learns a construction manual plan of how to build new variations of cars with different colours, shapes, engine types, heights, number of doors and so on. If a model is truly able to generate new samples which follow the appearance of real world objects, we can truely say that it indeed has learned and understood a concept without teaching, known as supervised learning. The long term goal is to be able to automatically extract and learn these natural features occurring in the real world no matter what the data distribution might look like. There are two approaches to generative modeling which are shown in Figure 5.1 and described below

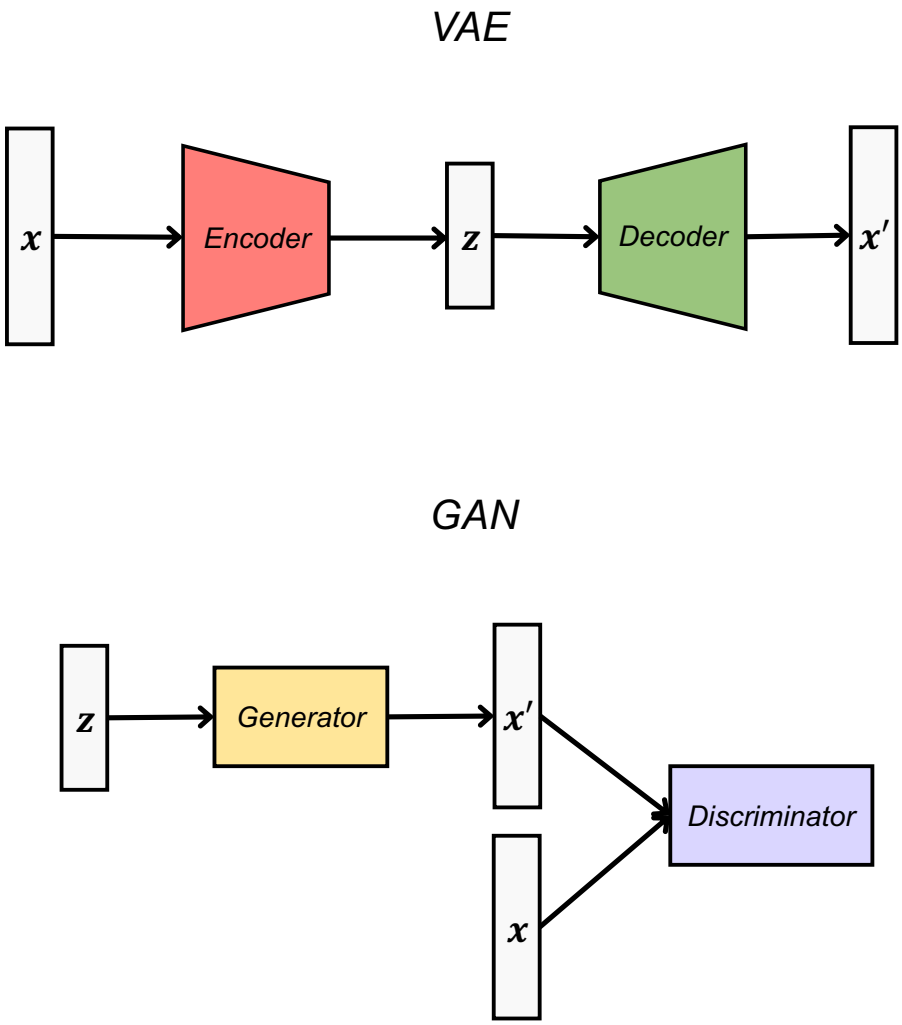


Figure 5.1: Architectural difference between GAN and VAE

- Generative Adverserial Networks (GAN):
  These models are trained in a competing game between two separate networks. One network is creating (generator) data samples, while the other network (discriminator) tries to distinguish data samples as either coming from the training data set or from the generator. The training is stopped as soon as the discriminator is randomly guessing between fake and real data samples, which means that its probability to correctly discriminate between images from the true distribution and those from the generator distribution is exactly $p = 0.5$. GANs are able to generate new data samples for the image domain with astonishing results; yet, it remains difficult to learn to generate discrete data such as text. Finally, when we want to learn a compressed representation of our data samples, in effect perform inference, GANs can not be used as they are a mapping from a latent code z with unit gaussian noise distribution $\mathcal{N}(0, 1)$ into the image domain x, but in effect we want the inverted functionality.

- Variational Autoencoders (VAE):
  VAEs can be used to approach and solve the two disadvantages of GANs, which is why we are using focusing on this class of generative models for our work with written language and NLP. VAEs give us the nice property to formalize the learning problem in a directed probabilistic graph model $p(x|z)$, where the encoder learns an approximation of the of the true data distribution $p(z|x)$ where we have to optimize our VAE objective in order to maximize the log likelihood of the data distribution. VAEs are an extension of vanilla autoencoders (AE) with constraints on the distribution of the latent variables z. VAEs similarly to GANs also consist of two neural networks. One of them is the inference network outputting the latent variable distribution of z and the second NN is the generation network, which maps back from latent variable space Z into the data domain X.

In order to understand how the architecture for our experiments in the following chapter works, we will start with the basic model contained in VAEs, the autoencoder AE. From there we will explain the differences between AE, VAE and β-VAE.

5.2 Autoencoder (AE)

Autoencoders first introduced in [69] are a concept of neural network models which try to learn a compressed representation $\mathbf{c} \in \mathbb{R}^m$ of the input data $\mathbf{x} \in \mathbb{R}^n$, which is of at least higher dimensionality. AEs learn these features in an unsupervised manner without any corresponding label because the goal of the autoencoder is to reconstruct the input at the output. Autoencoders consist of two neural network blocks:

- **Encoder:** The task of the encoder function is to map the input x from an n-dimensional space into a smaller m-dimensional code space, which is equivalent to the values within the hidden layer. The hidden layer if often referred to as bottleneck, as it is the layer with the lowest dimensional representation of the data from input layer up until the output layer. The input layer dimension is usually larger than the output layer dimension of the encoder.
- **Decoder:** The task of the decoder function is to re-map the compressed code representation from the m-dimensional code space back into the original n-dimensional input data space.

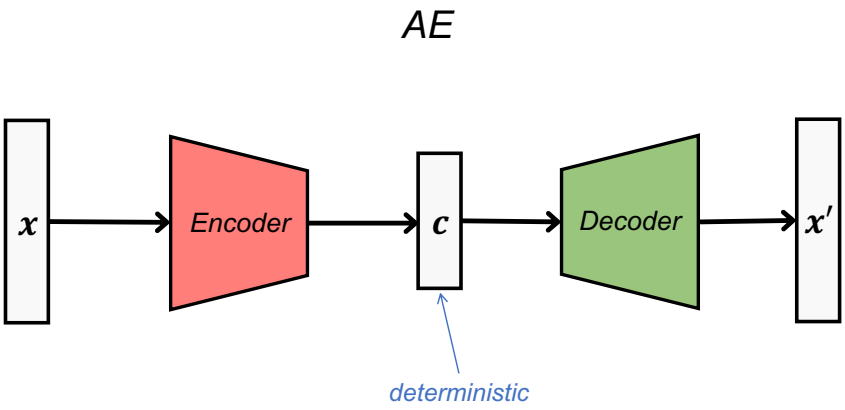


Figure 5.2: Simple autoencoder (AE) with non-stochastic latent code $\mathbf{c}$

The basic structure of an AE is shown in Figure 5.2. AEs differ from other previously explained neural network models because they do not predict a certain ground truth information but rather try to reconstruct the output as accurately as possible to the input $\mathbf{x}$. In effect, vanilla AEs come closest to models which learn the identity function, while at the same time extract a condensed, less sparse representation of the input data. Assuming that the encoder function is given by $\mathbf{c} = g_\phi(x)$ and the decoder function by $\mathbf{x}' = f_\theta(c)$, the objective function of the autoencoder is to minimize the sum of differences between every input and output, which represents the reconstruction error

$$\mathcal{L}_{AE}(\phi, \theta, \mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2 \tag{5.1}$$

The parameters of neural network parameters of the encoder ϕ and decoder θ are learned concurrently during training to reduce the reconstruction error. The ultimate goal of the enoder is $\mathbf{x} \approx f_\theta(g_\phi(\mathbf{x}))$, which can be achieved by the identity function.

Autoencoders can be applied for problems, where we are interested in *dimensionality reduction* into a lower dimensional representation while at the same time learning a latent code mapping. Variations of the the autoencoder family are:

- **Denoising Autoencoders** [70] which prevent AEs from overfitting when there are more network parameters than the total amount training samples. Moreover, denoising AEs improve upon robustness of the latent representation in case of corruption or partial data loss.
- **Sparse Autoencoders** [71] which introduces a ‘sparse‘ constraint upon the activation of single units within the latent code $\mathbf{c}$ to prevent overfitting and improve robustness of the internal representation. The goal is to keep the number of simultaneously active hidden units c_i at a minimum, ideally at most one unit is activate

at any given time. The sparsity constraint is added as penalty term into the vanilla AE objective

- **Variational Autoencdoers** [1] (which will be explained in the next subsection)

5.3 Variational Autoencoder (VAE)

The term autoencoder in variational autoencoder is quite misleading [72], as it is more related to *Variational Bayesian* methods and *probabilistic graphical model* which describes the dependence structure between different random variables RV. The VAE's goal is to find an explicit distribution which is able explain most of the data samples of the input instead of learning a simple mapping onto a fixed vector (here fixed means deterministic as in the AE case). The VAE is a probabilistic approach to describe data samples $\mathbf{x}$ in the latent representation space $\mathbf{Z}$. Therefore, every latent variable in the latent code $\mathbf{z}$ is described by a probability distribution. The model architecture is shown in Figure 5.3 below.

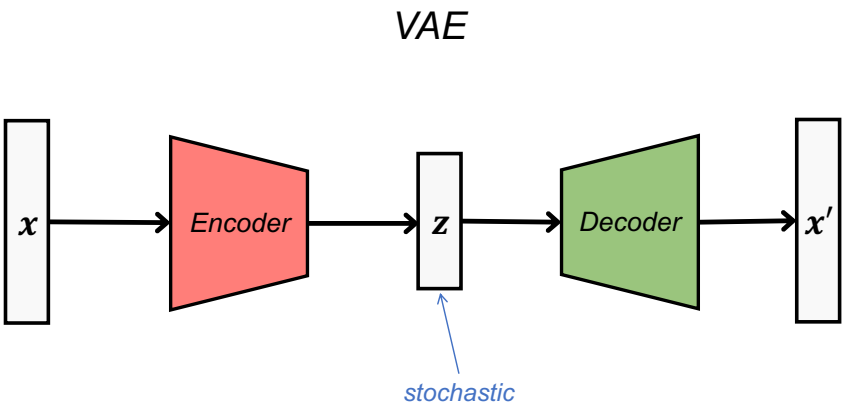


Figure 5.3: Architecture design of VAE

which can be further detailed and explained by opening up the decoder to have a look at the stochastic sampling process as shown below in Figure 5.4

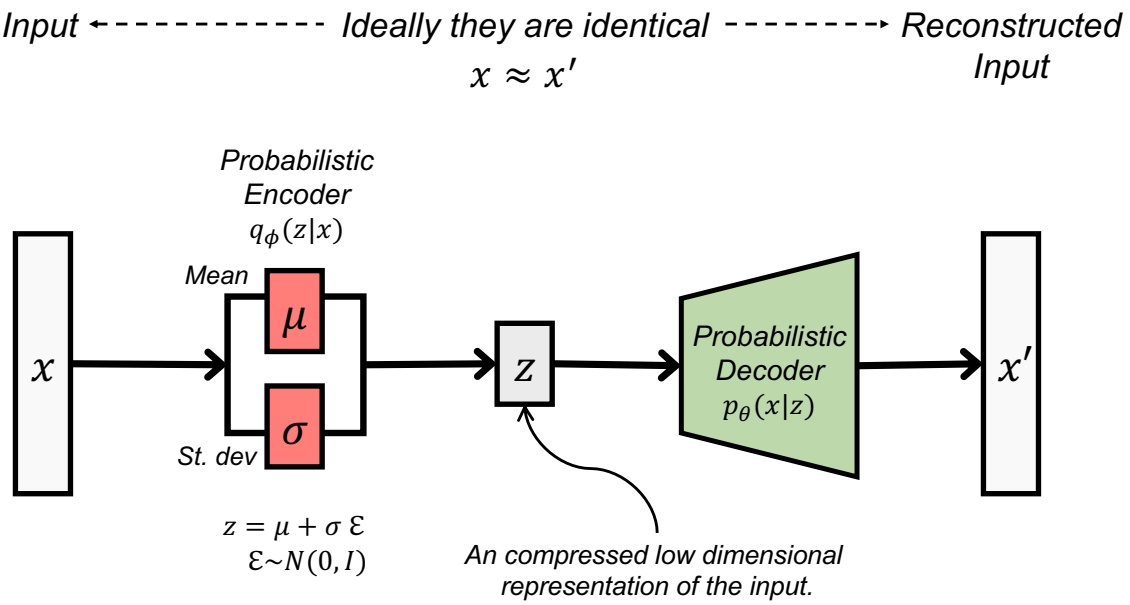


Figure 5.4: Architecture design of VAE - with detailed encoder process

In order to understand the principle of VAEs we need to differentiate between three different probability respectively conditional probability distributions which are

- *Prior* $p(\mathbf{z})$
- *Likelihood* $p_\theta(\mathbf{x}|\mathbf{z})$, which is defined by the decoder neural network
- *Posterior* $q_\phi(\mathbf{z}|\mathbf{x})$, which is defined by the encoder neural network

Knowing these terms, we can sketch our probabilistic graphical models as shown in Figure 5.5:

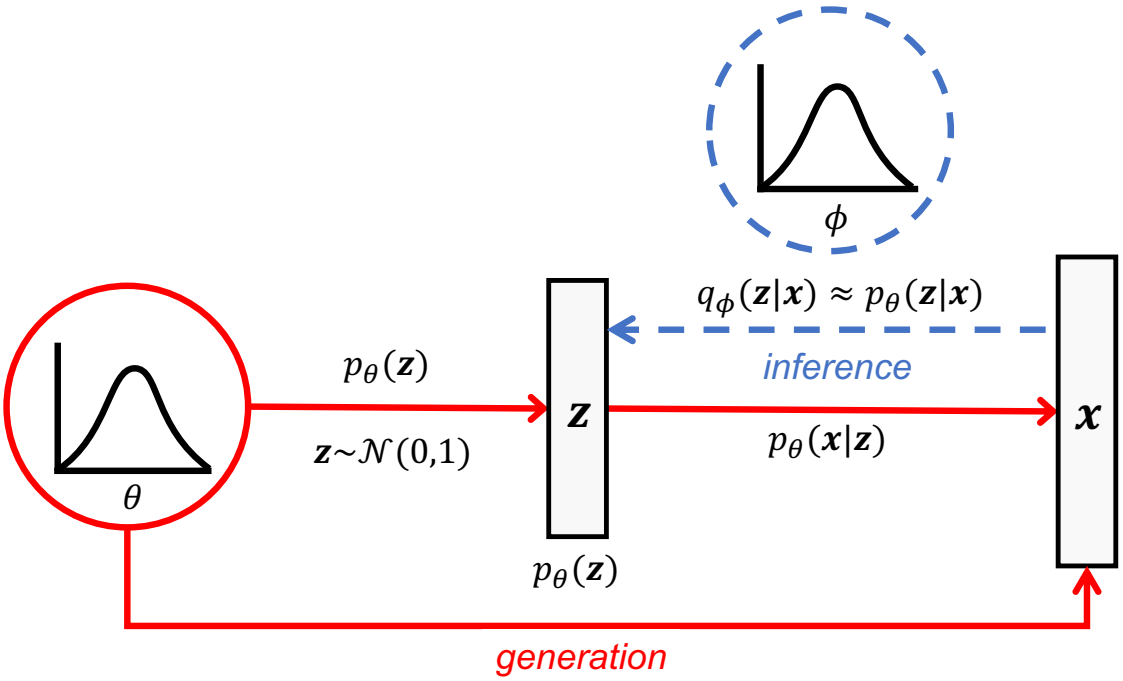


Figure 5.5: Graphical probabilistic model of the VAE process (inference + generation)

Where again ϕ is the collection of parameters belonging to the encoder function and θ for the parameters of the decoder function $\mathbf{z}$ denotes the latent space representation as output of the encoder. We can generate new data samples by doing the following two steps

1. Sampling a latent variables vector $\mathbf{z}^{(\mathbf{i})}$ from the posterior $p_\phi(\mathbf{z}|\mathbf{x})$ which is an approximation to the prior distribution $p(\mathbf{z})$
2. Using the decoder as generative network and reconstructing the sampled latent vector with the conditional likelihood function $p_\theta(\mathbf{x}|\mathbf{z} = \mathbf{z}^{(i)})$

The lower dimensional space $\mathbf{z}$ is stochastic with the encoder posterior distribution function $p_\phi(\mathbf{z}|\mathbf{x})$ being a Gaussian. Due to the sampling step 1, the latent representation $\mathbf{z}$ in the VAE is noisy and not deterministic. Therefore, we are actually enforcing the latent space representation to be continuous and smooth. Latent space vectors $\mathbf{z}$ which are close to each other, in effect only differ due to the added sampling noise, will thus be similar in their reconstructed appearance.

Within the VAE, information is lost because of two reasons

1. We squeeze our input $\mathbf{x}$ from a large dimensional space $\mathbb{R}^n$ through the encoder network into a lower dimensional space $\mathbb{R}^d$, which represents a bottleneck

2. In addition to the bottleneck, we additionally sample from the bottleneck which makes our reconstruction even more inaccurate to the original encoder input

The optimal parameter set θ' is the one, which maximizes the reconstruction likelihood for each data point $\mathbf{x^{(i)}}$. Therefore, our goal is

$$\theta' = \arg\max_{\theta} \prod_{i=1}^{N} p_\theta(\mathbf{x^{(i)}}) \tag{5.2}$$

which is equivalent to maximizing the sum of log probabilities. This is due to the fact that the logarithm *log* is a monotonically increasing function. Moreover, does it make computation numerically stable and allows subsequent simplifications

$$\theta' = \arg\max_{\theta} \sum_{i=1}^{N} log(p_\theta(\mathbf{x^{(i)}}) \tag{5.3}$$

If we want to compute the true posterior distribution of the latent space $p_\theta(\mathbf{z}|\mathbf{x})$ we would have to determine $p_\theta(\mathbf{x})$ given the *Bayes' Theorem*

$$p_\theta(\mathbf{z}|\mathbf{x}) = \frac{p_\theta(\mathbf{x}|\mathbf{z})\, p_\theta(\mathbf{z})}{p_\theta(\mathbf{x})} \tag{5.4}$$

Where $p_\theta(\mathbf{x})$ is the evidence which can be calculated as marginalization of all the possible latent space vector values $\mathbf{z}$ as follows

$$p_\theta(\mathbf{x^{(i)}}) = \int p_\theta(\mathbf{x^{(i)}}|\mathbf{z})\, p_\theta(\mathbf{z}) d\mathbf{z} \tag{5.5}$$

It is evident, that this approach is inconvenient because calculating the posterior $p_\theta(\mathbf{x^{(i)}})$ for each data sample $\mathbf{x^{(i)}}$ quickly becomes quite expensive and intractable. Therefore, VAEs use variational approximate inference of the intractable distribution of the true posterior which is represented by the encoder distribution function by $q_\phi(\mathbf{z}|\mathbf{x})$. Because $q_\phi(\mathbf{z}|\mathbf{x})$ is only an estimation of the true intractable posterior, we have to keep track of the difference of those two probabilities. This can be done by measuring the difference between those probability distributions with the help of the KL divergence which quantifies the distance. For two probability distributions the KL divergence is given by

$$D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})) = \int q_\phi(\mathbf{z}|\mathbf{x})\, log\left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right) d\mathbf{z} \tag{5.6}$$

and by Jensen's inequality, the KL divergence is always non-negative $D_{KL}(p||q) \geq 0$. So we are able to determine how different the approximated posterior $q_\phi(\mathbf{z}|\mathbf{x})$ is from the true posterior $p_\theta(\mathbf{z}|\mathbf{x})$.

Now we will derive the objective function for the VAE by analyzing and decomposing the KL-divergence term above.

$$
\begin{aligned}
D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})) &= \int q_\phi(\mathbf{z}|\mathbf{x}) \, log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} \right) d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \, log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x}) \, p_\theta(\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} \right) d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \, \left(log \, p_\theta(\mathbf{x}) + log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} \right) \right) d\mathbf{z} \\
&= log \, p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \, log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} \right) d\mathbf{z} \\
&= log \, p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \, log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) \, p_\phi(\mathbf{z})} \right) d\mathbf{z} \\
&= log \, p_\theta(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\, log \left(\frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} \right) - log \, p_\theta(\mathbf{x}|\mathbf{z}) \,] \\
&= log \, p_\theta(\mathbf{x}) + D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}) - \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\, log \, p_\theta(\mathbf{x}|\mathbf{z}) \,]
\end{aligned}
\tag{5.7}
$$

Line three to four uses the fact, that any probability distribution has the property $\int q(\mathbf{z}|\mathbf{x}) d\mathbf{z} = 1$. From line five to six, we used the definition of KL divergence provided above. Now, when we reorder the equation above, we get the following identity:

$$
\begin{aligned}
log \, p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})) = \\
\mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\, log \, p_\theta(\mathbf{x}|\mathbf{z}) \,] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z})) = \\
-ELBO
\end{aligned}
\tag{5.8}
$$

The left-hand side is our objective for the VAE which we want to maximize because the first term represents the reconstruction likelihood (we want to increase this term as much as possible) and the second term ensures that our learned approximate posterior distribution is similar to the true prior distribution of the data samples (we want to keep this term as small as possible). This works because the KL divergence term has the effect of a regularizer. Given the fact, that many optimization algorithms such as the stochastic gradient descent (SGD) work by minimzing the objective function, we have to flip the signs of this term. The final objective function for the VAE is thus given by

$$
\begin{aligned}
\mathcal{L}_{VAE}(\phi, \theta, \mathbf{x}, \mathbf{z}) &= -ELBO \\
&= \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\, log \, p_\theta(\mathbf{x}|\mathbf{z}) \,] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}))
\end{aligned}
\tag{5.9}
$$

ELBO is an expression defined for Variational Bayesian methods. This loss expression is known as the *variational lower bound* respectively *evidence lower bound*. ELBO is the negative of the objective function for VAEs. The term is using 'lower' because the KL divergence is non-negative. Therefore, the VAE objective will always be smaller than the 'true' log likelihood $log\, p_\theta(\mathbf{x})$ and therefore underestimating it as follows

$$log\, p_\theta(\mathbf{x}) - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z}|\mathbf{x})) = \mathcal{L}_{VAE} = -ELBO \leq log\, p_\theta(\mathbf{x}) \tag{5.10}$$

Further, we now see that we are optimizing by minimizing the negative log-likelihood (NLL), which is the first term in the equation above. Thus we retrieve the best set of parameters by minimizing the loss function $\mathcal{L}_{VAE}$ respectively maximizing the evidence lower bound (ELBO) using gradient descent methods to solve

$$\theta', \phi' = \arg\min_{\theta,\phi} \mathcal{L}_{VAE} \tag{5.11}$$

We can see, that by minimizing the VAE loss, we are simultaneously maximizing the ELBO which is proportionally behaving to the likelihood and thus we increase the reconstruction likelihood. To sum up how the VAE works here a sequence of steps from input to output:

1. Pass input $\mathbf{x}$ into the encoder network
2. A latent representation vector $\mathbf{z}$ can be sampled from the posterior distribution $q_\phi(\mathbf{z}|\mathbf{x})$
3. Pass the latent vector $\mathbf{z}$ into the decoder to retrieve a close reconstruction $\mathbf{x}'$ of the initial input $\mathbf{x}$ in step 1.

5.3.1 Reparameterization trick - normally distributed prior

The only missing part in fully explaining the VAE is the process within the bottleneck which generates latent variable samples $\mathbf{z}$ which are non-deterministic by sampling from $\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})$. This method is called *reparameterization trick*. This is needed because back-propagation (BP) in neural networks for determining the encoder and decoder parameters θ, ϕ only works for deterministic nodes and operations. This problem is shown in the Figure 5.6 where we clearly have to pass through a deterministic sampling node (yellow colored).

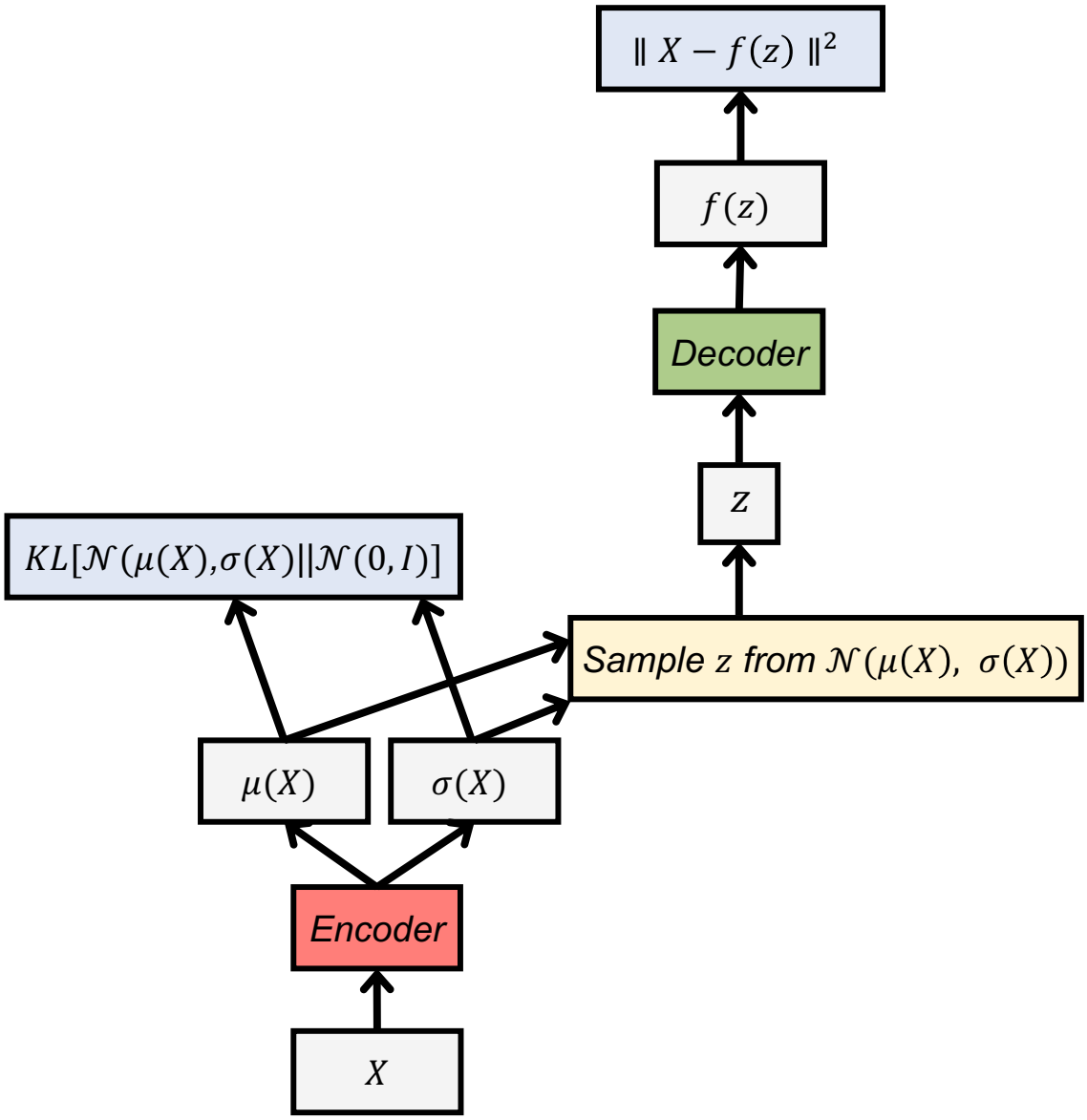


Figure 5.6: Backpropagation and the need for a reparameterization trick

This excludes the option of a simple random sampling in the bottleneck because sampling generates stochastic values, whereas backpropagation expects deterministic values. The reparamterization trick is our remedy for solving this issue. It assumes, that we can sample a random latent variable vector $\mathbf{z}$ by employing a deterministic construction function which is determined by the data input $\mathbf{x}$ and some small value of randomness ϵ sampled

independently and injected into this construction function as follows

$$\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma}^2 * \mathbf{I}) \tag{5.12}$$

In that way, the stochasticity from the sampling process is independent of the parameters of the encoder and decoder network. The values will be sampled in the following way

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \tag{5.13}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are deterministic outputs generated by two different FC layers. Here, $\odot$ refers to the *Hadamard product* which denotes the element-wise product. A standard choice for the approximated posterior is a multivariate Gaussian distribution with a diagonal covariance matrix. Therefore, in order to get such a multivariate Gaussian distribution, our sampled noise must also follow a Gaussian distribution. A common way to achieve this is by simply sampling $\boldsymbol{\epsilon}$ from the normal distribution; thus we have

$$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{5.14}$$

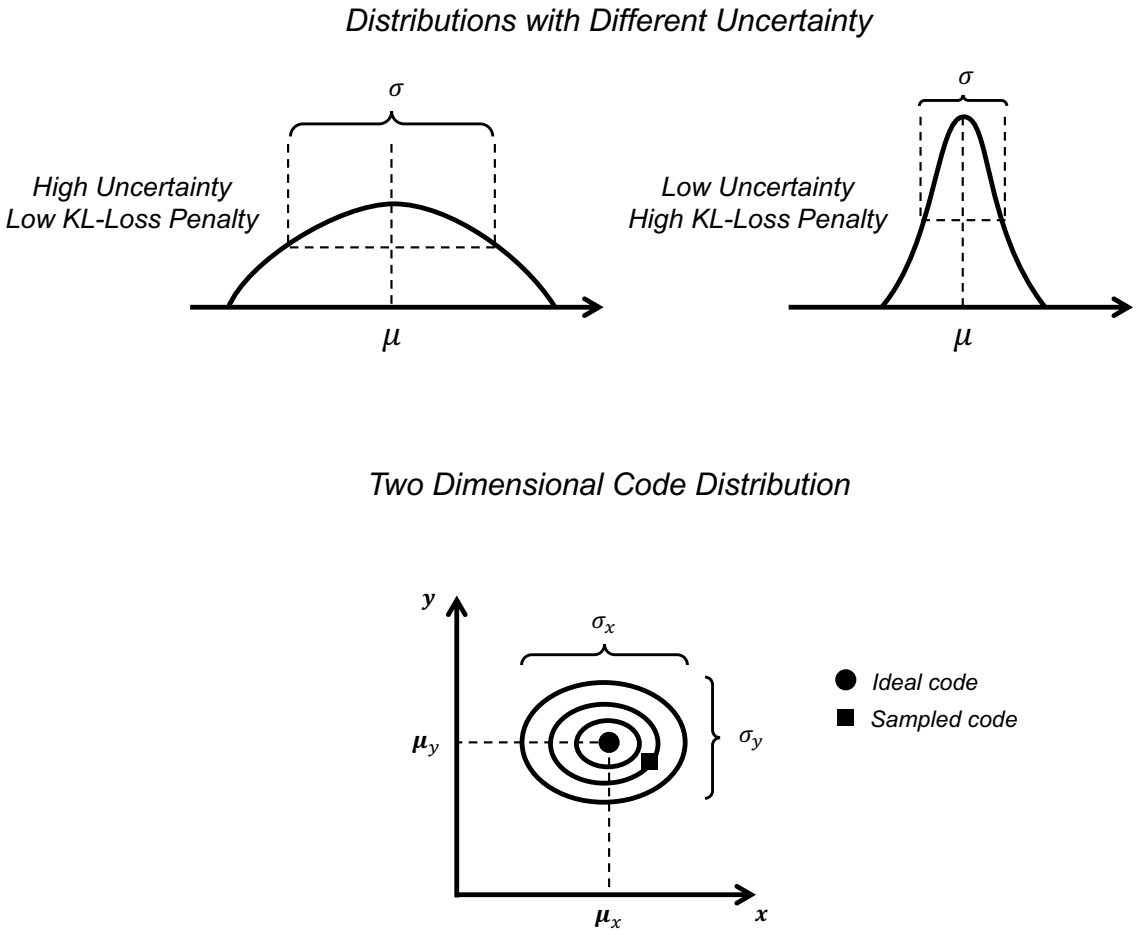


Figure 5.7: The interpretation behind the variance & mean for a latent code variable z

Therefore, the circumvented random sampling process using the reparameterization trick looks as shown in the Figure 5.8 below

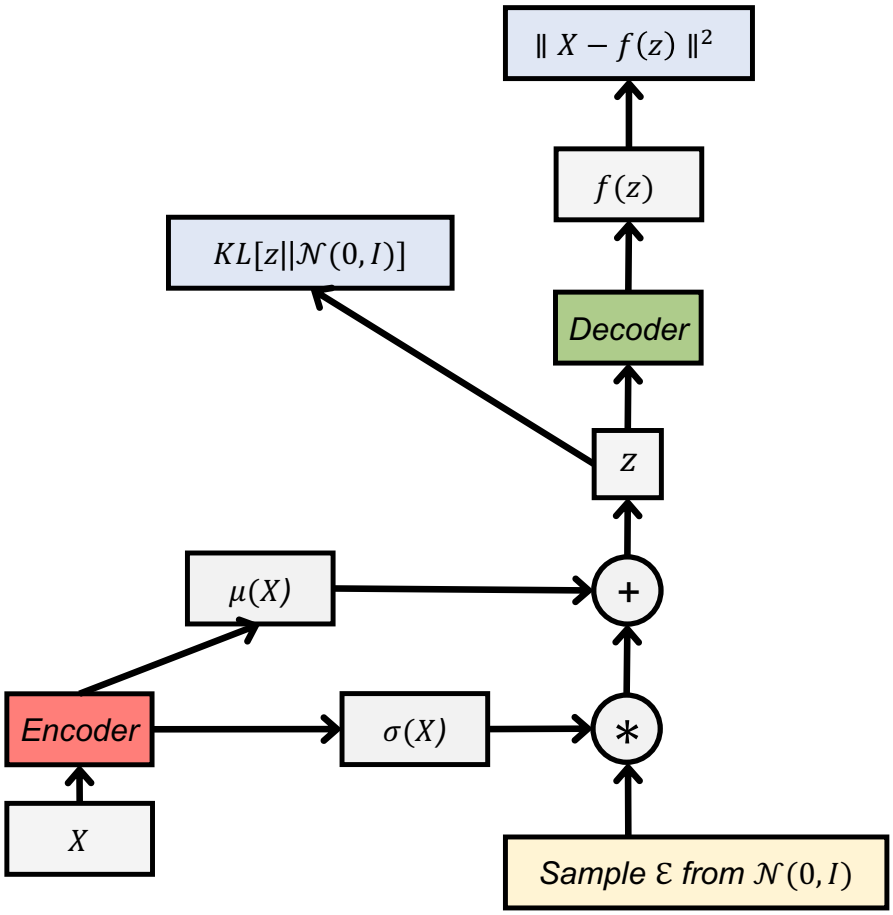


Figure 5.8: Reparameterization without any stochastic node on the way from output to input feasible to backpropagation (BP)

Changing the sampling process in the bottleneck and getting from '∼' which is equivalent to drawing from a random distribution to '=' is the most important step in the VAE and allow us to leverage the tools of deep learning. This reparameterization function only depends on the deterministic parameters from the inference network. We can therefore calculate the gradients of the decoder output $f(\mathbf{z})$ with respect to the parameters of the latent variable distribution $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ and backpropagate this information to the encoder. Finally, this allows us to train the VAE in an end-to-end fashion while using a graphical model and variational bayesian methods. The parameter σ can be understood as a measure of uncertainty. This is shown in Figure 5.7. The autoencoder is a distribution, in which the sentences of our dataset are encoded without any uncertainty as shown in the Figure 5.9.

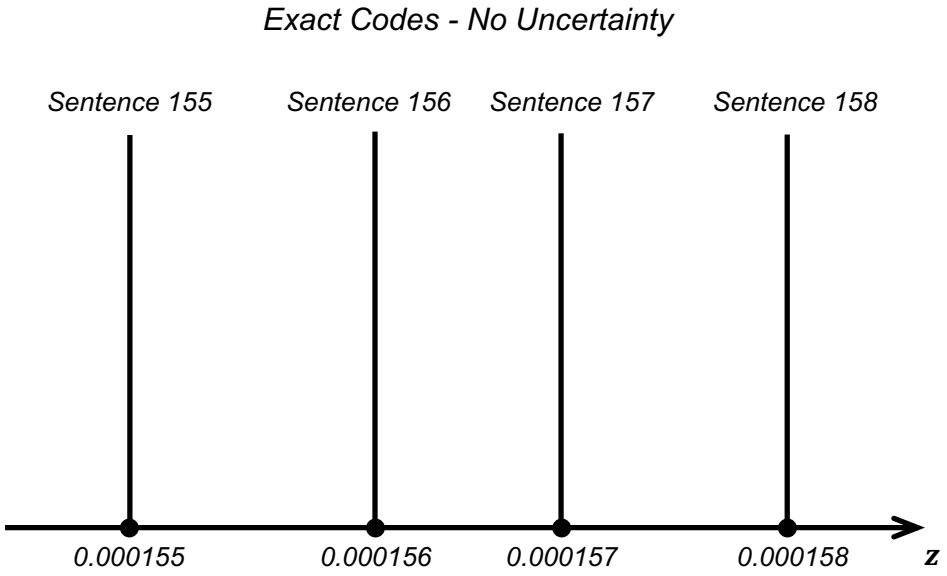


Figure 5.9: What the distribution of an autoencoder (AE) looks like

5.4 Regularized Variational Autoencoder (β-VAE)

Now, that we have in detail explained how the VAE works, understanding what β-VAE is doing is quite straightforward because the β-VAE model is a modification of the VAE framework presented in previous section. The emphasis of introducing β-VAE lies in the discovery of independent generative factors in separate latent dimensions z_i of within the whole latent vector $\mathbf{z}$, thus disentangled representation learning. The key essence is the introduction of a *Lagrangian multiplier* β which is considered as an additional hyperparameter to the original VAE objective which is affecting the KL divergence between our approximated posterior $q_\phi(\mathbf{z}|\mathbf{x})$ and the imposed prior distribution $p_\theta(\mathbf{z})$. From now on, in order to prevent confusion between the likelihood and the prior, we will denote the prior as: $p(\mathbf{z})$. β-VAE focuses on the latent space $\mathbf{Z}$ and understanding its disentanglement. One can think of the latent space $\mathbb{R}^d$ as a space, in which every dimension d contains certain information which might be helpful in reconstructing the input $\mathbf{x}$, for example when talking about NLP, one dimension could save information about the sentiment of the text document, whereas the second dimension would describe the topic of the text document and so on. With that regard, the latent space contains the reconstruction plan the decoder should make use of in order to optimize the data likelihood. Often this latent space is entangled, which means, that attributes such as style, topic, tense and so on are contained in several latent variables z_i and therefore single latent variables will have a high correlation or contain redundancy such as information or

attributes already sufficiently described by another latent variable z_i. The main benefit of a β-VAE is that is is capable of learning a smooth latent space representations $\mathbf{z}$ while at the same time forcing the model to encode different factors of variation into different latent variable units z_i due to the constraint it gets in the KL divergence term over β. Whereas for standard AEs (where $\beta = 0$), we only need to learn a 'hard' in discontinuous representation of the input which only is capable of exactly reproduce the input without any safety margin in case our latent space information is corrupted by a small amount of noise or shifted on purpose.

A disentangled representation means, that each latent dimension in the latent representation vector $\mathbf{z}$ is only sensitive to a single ground truth generative factor and therefore invariant to variations of other generative factors. Therefore, we can clearly inspect and denote the attribute each latent dimension has learned which helps us in clearly interpreting the learned latent representation. β-VAE is able to learn such attributes in a very disentangled manner such that it separates information about the x and y coordinates into two mutually exclusive latent variable dimensions. Because VAE and β-VAE are directly related to each other, we also have to maximize the log likelihood of the generated data samples while keeping the KL divergence loss between prior and posterior fairly small. As the only difference is the introduction of the new KL divergence hyperparameter β, the objective function looks as follows

$$\begin{aligned} \mathcal{L}_{\beta-VAE}(\phi, \theta, \beta, \mathbf{x}, \mathbf{z}) &= -ELBO \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\, log\, p_\theta(\mathbf{x}|\mathbf{z})\,] - \beta D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \,||\, p_\theta(\mathbf{z})) \end{aligned} \tag{5.15}$$

Having introduced the β-VAE, the relationship between all presented variations of bottleneck models can be described as follows:

- $\beta = 0$: vanilla AE with standard maximum likelihood objective function
- $\beta = 1$: vanilla VAE with the standard Bayes solution
- $\beta \notin \{0, 1\}$: regularized VAE, β-VAE

A variation of the β-VAE is the capacity controlled β-VAE by introducing a target capacity value C which can be increased during the training process and allow a higher KL divergence loss as training progresses. The capacity C corresponds to the amount of information encoded in the latent and it is measured in the *nats* which is a unit of information originating from ideas in information theory. The capacity C is increased from zero to a value which is provides enough capacity in order to generate output with lower reconstruction error.

The objective function for the capacity controlled γ-VAE is described as follows

$$\begin{aligned}\mathcal{L}_{\gamma-VAE}(\phi,\theta,\gamma,C,\mathbf{x},\mathbf{z}) &= -ELBO \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})}[\,log\, p_\theta(\mathbf{x}|\mathbf{z})\,] - \gamma\,|D_{KL}(q_\phi(\mathbf{z}|\mathbf{x})\,||\,p_\theta(\mathbf{z})) - C|\end{aligned} \tag{5.16}$$

5.4.1 Effect of β

β in the objective function acts as a regularizer on the KL divergence between the approximated posterior and the imposed prior.

We want to explain this given the following three visualizations.

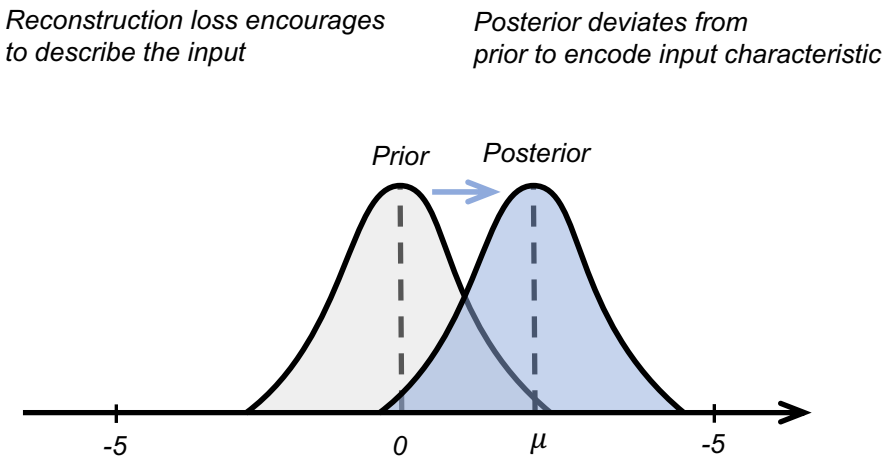


Figure 5.10: Active posterior

In the case of 5.10, the posterior is moving away from the posterior and therefore tries to find a space in the latent space, which might help in reducing the reconstruction error. Whereas in 5.11 the opposite effect is happening. As the KL penalty is larger the the benefit the posterior distribution is contributing to the increase the likelihood, the posterior is being drawn towards the distribution of the prior in order to reduce the penalty introduced by the KL divergence term.

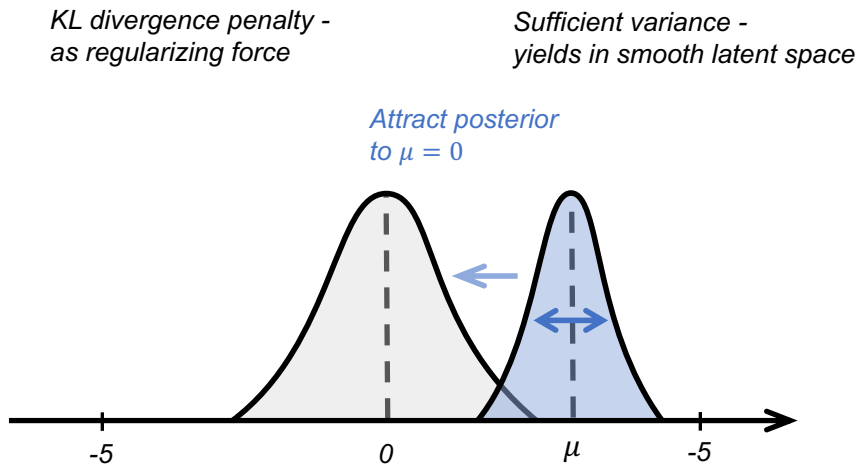


Figure 5.11: Collapsing posterior

Finally, in 5.12, the case without a KL divergence term, the posterior decides to only encode a very tiny space in the latent space, which causes the latent space to be very unsmooth and filled with a lot of 'air' instead of sample representations

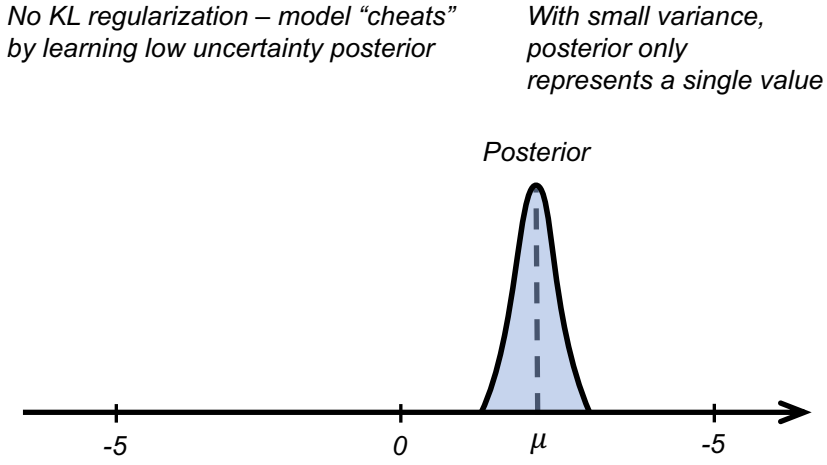


Figure 5.12: Posterior distribution in case: no KL divergence applied

It can be viewed as a coefficient which balances out the magnitude of the gradients [4] during backpropagation (BP) from the two terms in the loss function, which are the log-likelihood expectation (by sampling from the posterior distribution) and the KL-term in order to match prior and posterior. Thus, When $\beta > 1$, the VAE model enforces a

stronger constraint on the latent representation bottleneck to keep the distributions similar. Moreover, the effect of β also depends on the dimensionality d of our latent representation $\mathbf{z}$. Larger d also results in the need for larger β values. β limits the 'representation capacity' $\mathbf{z}$ of the bottleneck without changing the capacity in terms of the number of existing latent variable units z_i and thus the VAE architecture remains physically same. When β is to low (AE case) or too high the model's learned latent representation becomes entangled and uninterpretable. In one case, when β is very low, there is just no incentive to use the capacity effectively because enough capacity is given, In the other case of high β, there is just too little capacity available to learn anything and the model decides to create an entangled latent representation $\mathbf{z}$. An appropriate sized and tuned β value encourages the model to more efficiently encode and structure necessary information by making better use of the available capacity in the bottleneck. This encourages disentanglement, where latent variables are factorized and therefore uncorrelated and independent from each other.

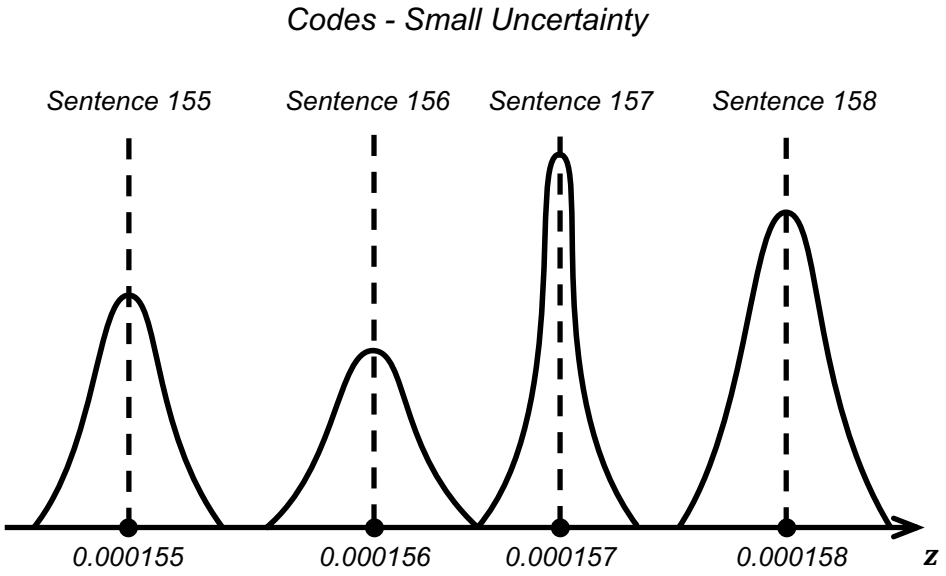


Figure 5.13: Latent code of β-VAE with not too large regularizer, small uncertainty

This is effect is shown in Figure 5.13. The only drawback, that comes with too high β values is that we run into a trade-off between reconstruction quality (in effect the NLL term in the ELBO) and successfully disentangling the latent variables (in effect the KL divergence term in the ELBO). We will later show, that this is not necessarily always the case and might well depend on the type and features of the latent space. The higher the bottleneck capacity C, the less blurry the output of an image is. Still, it remains open, what a 'blurry' reconstruction might mean in the case of other data domains such as text and audio. When the hyperparamter gets even higher, which leads to larger variances and thus higher code uncertainty, the structure of the latent variables must be well organized,

such that neighbouring samples are similar to each other. This results in a smooth latent space, which is shown in the Figure 5.14 below. In this case, we really want that sentences which are similar or closely related to each other also be located in that property within the latent space.

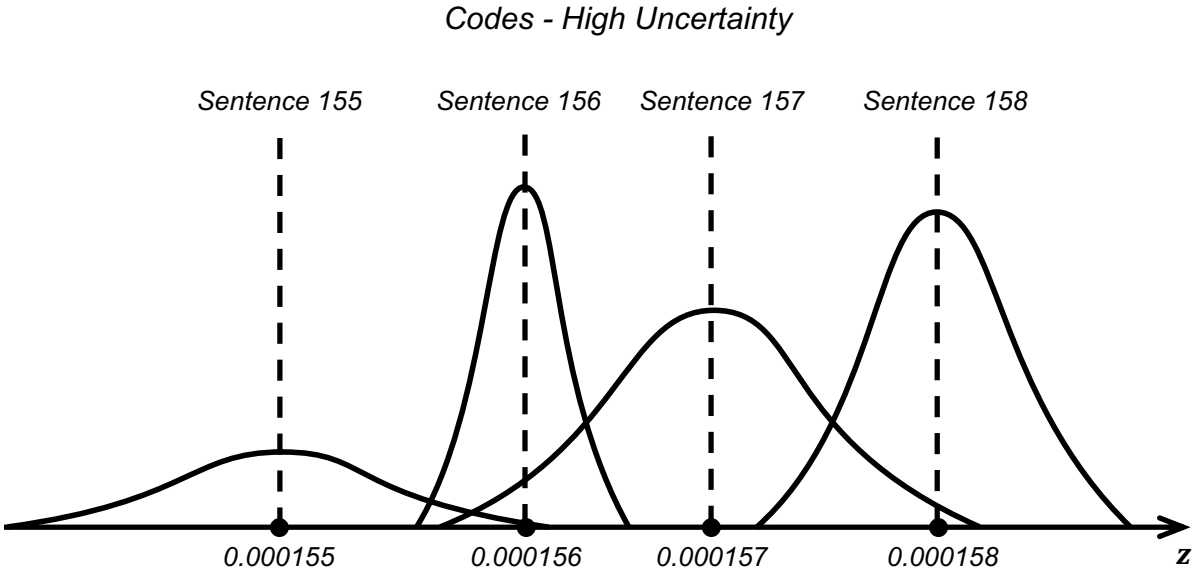


Figure 5.14: Latent code of β-VAE with not large regularizer, high uncertainty

Finally, good reconstruction results are not an indicator for a highly disentangled latent representation. Perfect reconstruction can be achieved with an AE, as its objective is to learn the identity function.

5.5 Discrete β-VAE with flexible latent sizes

For some data distributions, categorical variables, in effect a discrete latent representation, would be better suited. This might be for the identity of a digit in the MNIST dataset but also for text in natural language which consists of a sequence of discrete tokens.

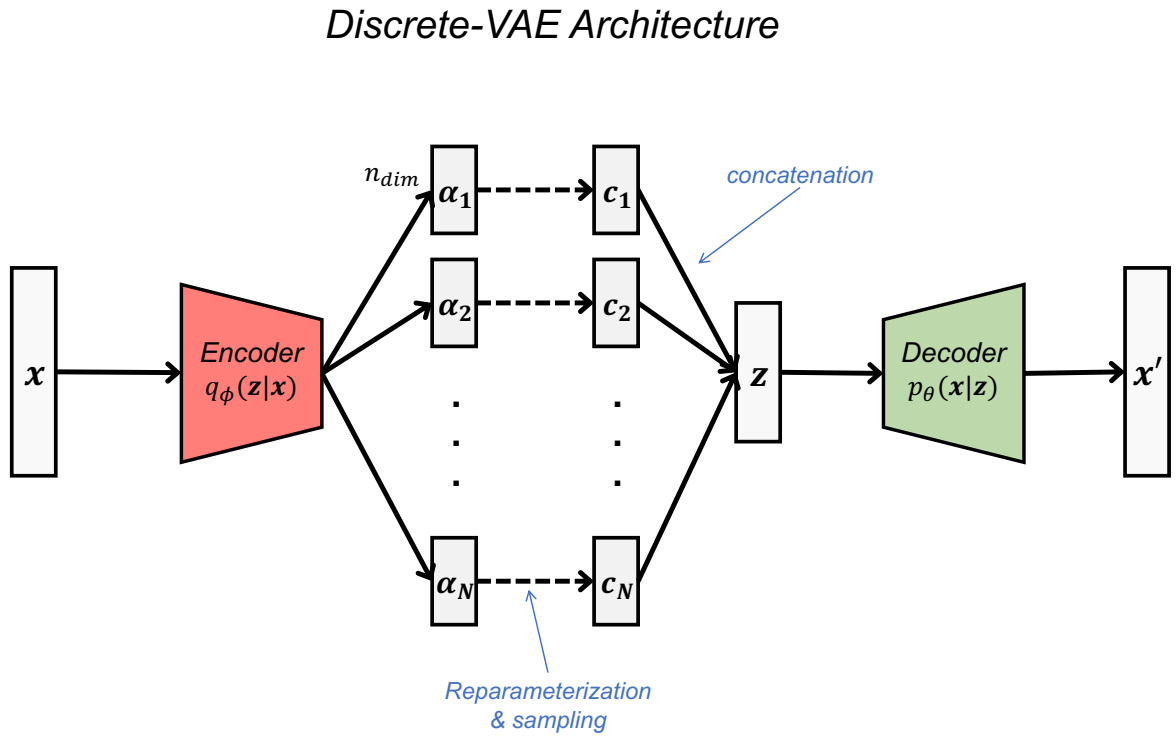


Figure 5.15: Our implemented architecture of the fully discrete β-VAE

Figure 5.15 gives an shows how we have constructed the fully discrete VAE. If we want to apply the VAE with a discrete latent space in a similar way as the continuous latent space VAE models, we clearly need to solve the problem of being able to backpropagate through the discrete samples in the bottleneck $\mathbf{c}$. We will explain how this is done using the Gumbel-Softmax trick for sampling and reparameterization of the latent representation $\mathbf{c}$. In the discrete-β-VAE categorical variables with n categories are represented as one-hot vectors in an n-dimensional space. In this section we switch the notation of our bottleneck from $\mathbf{z}$ to $\mathbf{c}$ because the presented VAE architecture consist of a fully discrete latent vector with flexible dimension size in each latent variable c_i. Looking at what source in VAEs enforces the probability distribution of latent variables, it becomes quite clear, that in order to train a VAE with discrete latent space we have to enforce that with a discrete prior distribution. For that, we are choosing the prior to be a *categorical distribution* denoted with $p_\theta(\mathbf{c})$. The β-VAE objective thus becomes

$$\begin{aligned}\mathcal{L}_{discrete-\beta-VAE}(\phi,\theta,\beta,\mathbf{x},\mathbf{c}) &= -ELBO \\ &= \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{c}|\mathbf{x})}[\,log\, p_\theta(\mathbf{x}|\mathbf{c})\,] - \beta D_{KL}(q_\phi(\mathbf{c}|\mathbf{x})\,||\,p_\theta(\mathbf{c}))\end{aligned} \tag{5.17}$$

The KL divergence term between the categorical latent variable distribution $q_\phi(\mathbf{c}|\mathbf{x})$ of the

inference network and a uniform categorical prior $p_\theta(\mathbf{c})$ is bounded. Therefore, we have the nice property of a limited and not diverging KL term for the discrete β-VAE setup as opposed to the KL-term for the vanilla VAE setup which has no limit and can cause issues in getting a VAE to initially train. We denote the distribution of the discrete posterior as $P := q_\phi(\mathbf{c}|\mathbf{x})$ and similarly denoting the discrete categorical prior distribution $Q := p_\theta(\mathbf{c})$. We can use the analytical expression of the KL divergence and evaluate the KL divergence for the discrete latent space $\boldsymbol{C}$ as follows

$$\begin{aligned} D_{KL}(P \,||\, Q) &= \sum_{i=1}^{n} p_i \, log \, \frac{p_i}{q_i} \\ &= \sum_{i=1}^{n} p_i \, log \, \frac{p_i}{1/n} \\ &= -H(P) + log(n) \leq log(n) \end{aligned} \tag{5.18}$$

Where n denotes the number of categories a single latent variable vector code can take on. We can think of it as the dimension a *one-hot* vector would need to encode n different categories or classes. Line to follows from the fact, that any uniform categorical distribution Q with n different elements has a limited entropy of $H(Q) = log(n)$. This is because if Q has n different classes, then the entropy is calculated as follows

$$\begin{aligned} H(Q) &= -\sum_{i=1}^{n} Q_i \, log \, Q_i \\ &= -\sum_{i=1}^{n} \frac{1}{n} \, log \, \frac{1}{n} \\ &= -n \, \frac{1}{n} \, (log(1) - log(n)) \\ &= log(n) \end{aligned} \tag{5.19}$$

Line two follows due to the fact, that for any uniform categorical distribution with n different classes the probability of any class is given by $Q_i = 1/n$. Given the case, that we have several discrete latent variable vectors $\mathbf{c^{(i)}}$, which can be of different dimensions $d^{(i)}$ (in effect take on a different amount of categorical values), then we can retrieve the final latent representation by concatenating all single categorical latent vectors into a single latent vector $\mathbf{c}$.

5.5.1 Reparameterization trick - categorically distributed prior

The novelty in the discrete-β-VAE is the fact, that our bottleneck is discrete. This means, as we have another prior distribution we definitely must sample from another noise source than a Gaussian normal distribution as we have done in for the vanilla VAE models. Moreover, we need to come up with a different way of sampling one-hot vectors for the latent space representation $\mathbf{c}$. We cannot parameterize the posterior with a categorical distribution because we need to consider that we want to train the VAE which employs backpropagation which generally only works for continuous valued nodes in the computation graph. The remedy in this case is to make use of the Gumbel-Softmax trick during reparameterization. It solves the difficulty in sampling from the categorical probability distribution $q_\phi(\mathbf{c}|\mathbf{x})$. We can think of $\mathbf{c}$ as a set categorical variables c_i where each category has a specific class probability α_i. The Gumbel-Softmax distribution is reparameterizable and therefore it allows to circumvent the stochastic node which we would have in a normal and direct sampling process. Therefore, we can again apply backpropagation solely through deterministic computation nodes. This reparameterization trick can smoothely deform the posterior during training into a categorical distribution. Drawing samples from the categorical posterior distribution $q_\phi(\mathbf{c}|\mathbf{x})$ with class probabilities α_i can be achieved as follows

$$\mathbf{c} = OneHot\left(\arg\max_i \left[g_i + \alpha_i\right]\right) \tag{5.20}$$

Where $\mathbf{g}$ is a vector of i.i.d sampled Gumbel noise values g_i from the Gumbel distribution $G(0, 1)$ which is given by

$$g = -\,log(-log(u)) \tag{5.21}$$

where u is sampled from

$$u \sim Uniform(0, 1) \tag{5.22}$$

Now, ones we have our class probabilities α_i and a set of corresponding Gumbel noise values g_i, we can use the softmax function as an alternative to $arg\,max$ in order to retrieve a continuous and differentiable estimation for the categorically distributed latent vector sample $\mathbf{c}$. Therefore, we can instead generate such 'soft' one-hot samples using the following equation

$$c_i = \frac{exp((log(\alpha_i) + g_i)/\tau)}{\sum_{j=1}^{n} exp((log(\alpha_j) + g_j)/\tau)} \tag{5.23}$$

Where n denotes the dimension of the categorical sample $\mathbf{c}$, which is a one-hot vector and thus can represent exactly n different categories.

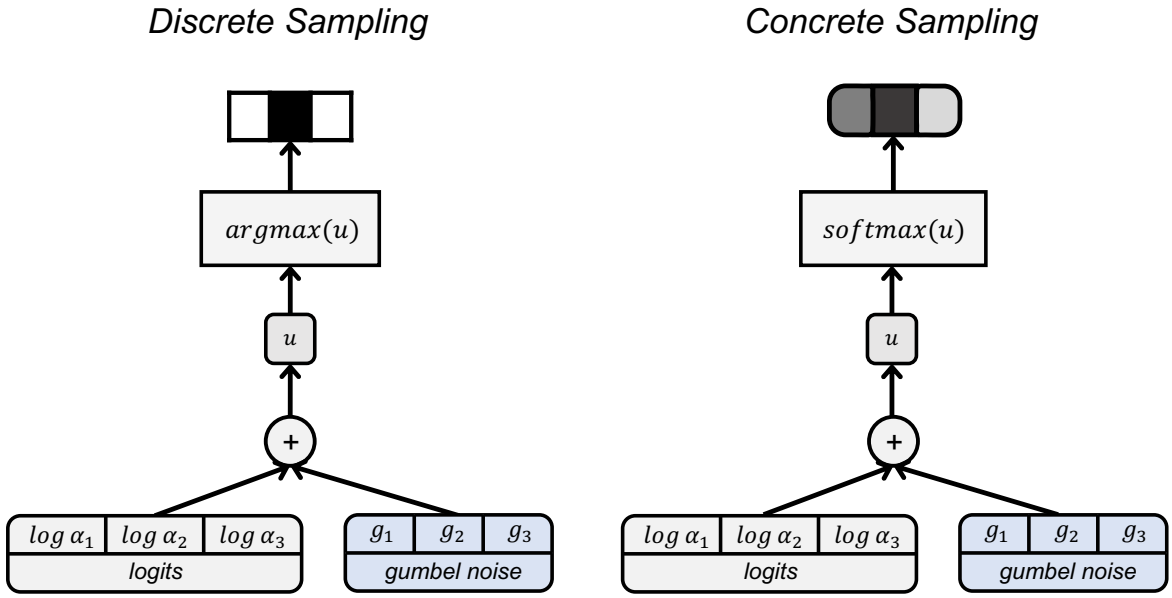


Figure 5.16: Discrete sampling for testting mode and conrecte sampling for training mode. Purple colour represents stochastic nodes and square boxes denotes discrete valued where as round denotes continuous valued

This process is shown in 5.16. The distribution from which we can sample these categorical latent vectors is termed as *Gumbel-Softmax* distribution. Moreover, τ denotes a temperature value, an additional parameter which can be used to control how close these 'soft' one-hot sampled vectors get to the actual categorical distribution, which would then be a 'hard' one-hot sampled vector. When $\tau \to 0$, the outputs of this Gumbel-Softmax distribution become real one-hot vectors and therefore there is no difference between applying the argmax operator directly. The temperature value during training phase starts from a larger value, in order to allow the flow of gradients and thus information for updating weights and biases past the sampling process back to the encoder network. Ones the training progresses, we decrease the temperature parameter τ to make the 'soft' one-hot samples from Gumbel-Softmax distribution become more and more distinct. Still, the temperature value is never annealed down to zero, as this would result in not being able to apply backpropagation and on the other hand the temperature annealing must be limited as too small values do lead to exploding gradient values.

This categorical but continuous distribution is called as Gumbel-Softmax distribution.

The prior in the discrete VAE model $p_\theta(\mathbf{c})$ is equal to the product of uniform Gumbel-Softmax distributions and the categorical posterior distribution $q_\phi(\mathbf{c}|\mathbf{x})$ is calculated as the product of independent Gumbel-Softmax distributions such that $q_\phi(\mathbf{c}|\mathbf{x}) = \prod_i q_\phi(\mathbf{c_i}|\mathbf{x})$. Therefore, each dimension in the posterior $q_\phi(\mathbf{c_i}|\mathbf{x})$ is a Gumbel-Softmax distribution itself with $q_\phi(\mathbf{c_i}|\mathbf{x}) = g(\boldsymbol{\alpha}^{(\boldsymbol{i})})$

To sum it up, the reparameterization trick in case of categorically distributed prior works as follows

1. Sample Gumbel noise $\mathbf{g}$ from the Gumbel distribution $G(0, 1)$
2. Apply the softmax function onto the logits vector in order to obtain a probability vector $\boldsymbol{\alpha}$ of class probabilities α_i
3. Finally add the terms from step 1. and step 2. and apply the softmax function to retrieve a 'soft' one-hot latent representation vector $\mathbf{c_i}$

To make the comparison clear we want to show the process of sampling using the VAE architecture as presented in 5.17.

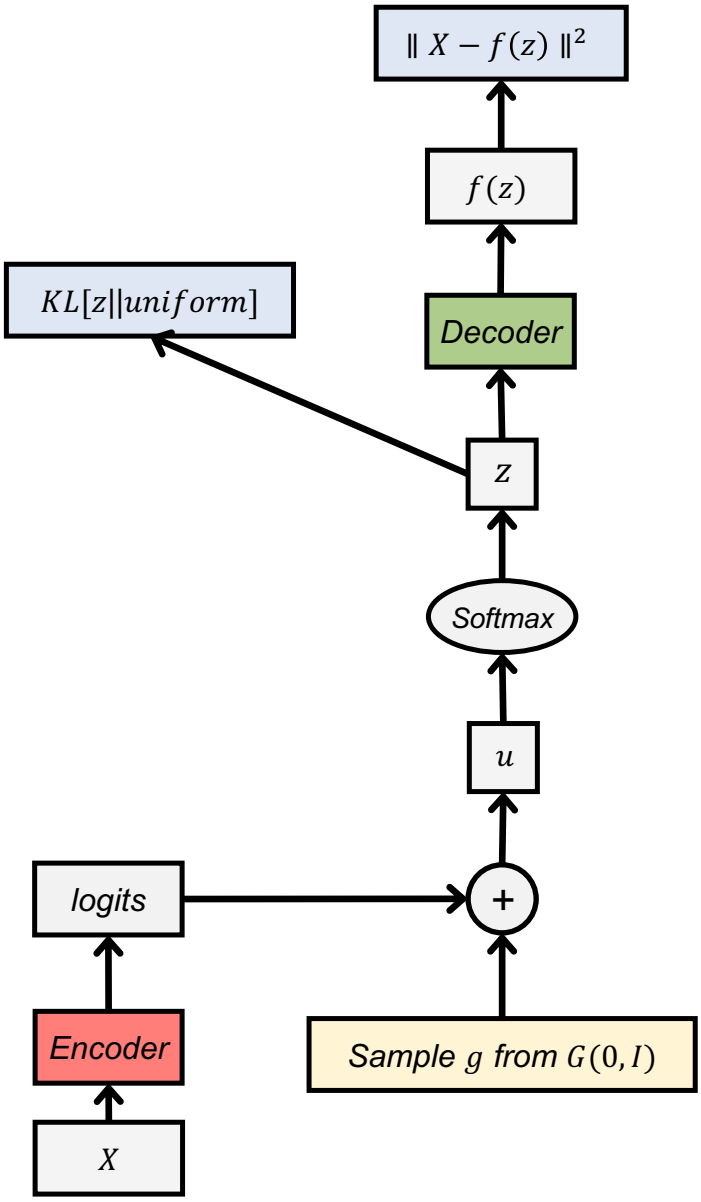


Figure 5.17: Gumbel Softmax trick within the VAE

For the vanilla VAE with continuous latent space we sample latent variables z_i as follows

$$z_i \sim \mathcal{N}(\mu_i, \sigma_i^2) \tag{5.24}$$

whereas for the discrete VAE with a categorical latent space we sample latent variables c_i

as follows

$$c_i \sim g(\boldsymbol{\alpha}^{(i)}) \tag{5.25}$$

5.6 Evaluation of disentanglement

In this section we will explain and present two ways which help to get an idea of what the latent representation has 'learned', how smooth it is and if the model at hand was able to perform disentanglement by factorizing its latent representation. Basically, the presented tools and methods are ways to inspect the latent space in order to make conclusion about the model's performance as well as comparing performance in reconstruction and disentanglement of various models.

5.6.1 Latent traversal - qualitative inspection

After we have successfully trained any VAE model. We can parse our data samples ones more e.g. test samples and retrieve their latent representations. We can now evaluate the latent space features by performing translation operations in the latent space such as interpolation or traversals by changing single dimensions while keeping all other values of the latent representation $\mathbf{z}$ fixed. The concept of latent traversal in context with natural written language is best explained with the following Figure 5.18.

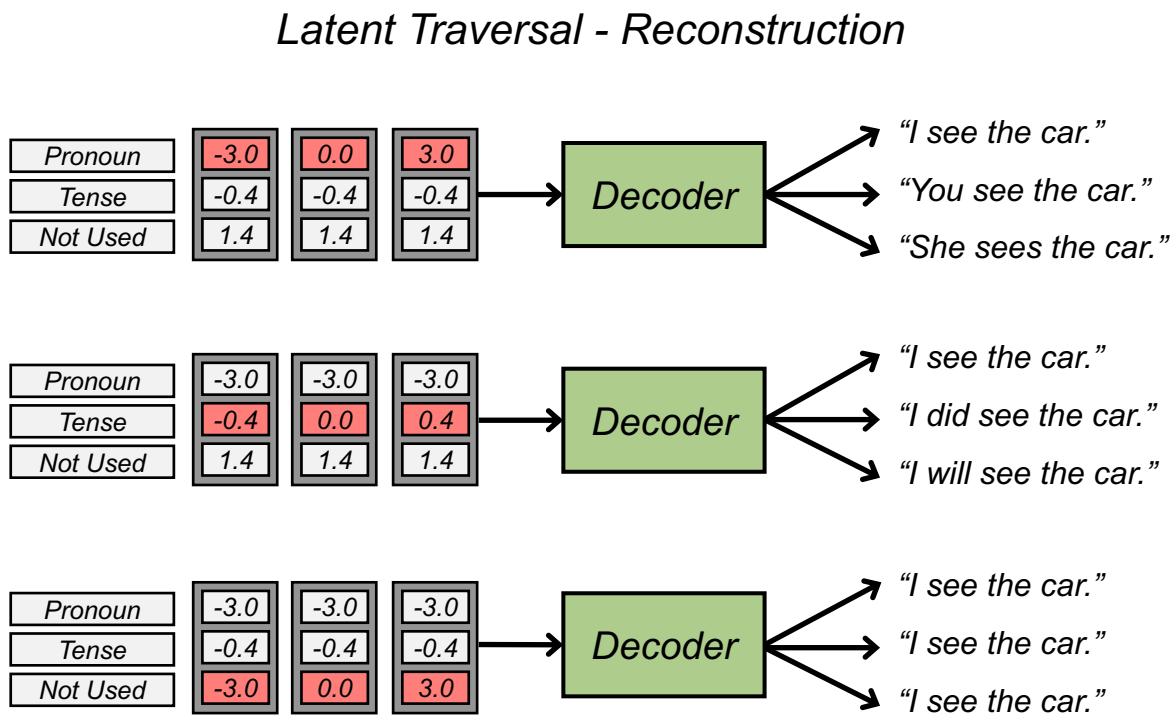


Figure 5.18: Latent Traversal of sentences with VAE

whereas the regular method without any manipulation of the encoded latent representation of any data sample input is referred to as the normal reconstruction operation and is shown in Figure 5.19 below.

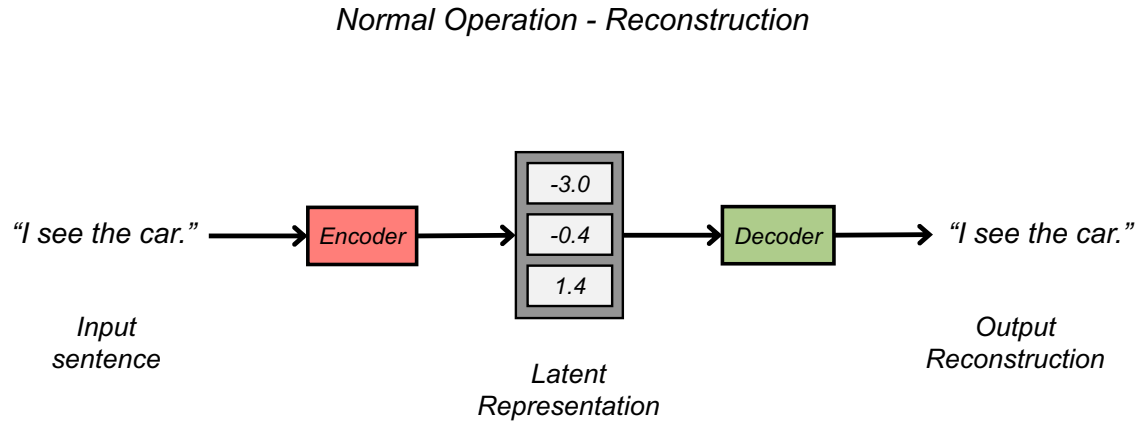


Figure 5.19: Regular reconstruction operation of sentences with VAE

We can do many other interesting things such as arithmetics. This means that instead of interpolating, we can add or subtract several latent space representation vectors $\mathbf{z}$. As we are interested in understanding and inspecting the property of disentanglement what we basically do is the following

1. Retrieve the latent representation $\mathbf{z}$ which is a tuple of vectors ($\boldsymbol{\mu}$, $log(\boldsymbol{\sigma}^2)$) of a data sample $\mathbf{x_i}$
2. Convert the log variance vector into variance vector $\boldsymbol{\sigma}^2$ and inspect the values of each dimension
3. One the one hand, a large variance which is still close to 1 and thus similar to the prior, means that the dimension has not learned anything that could contribute in reducing the reconstruction error. On the other hand, a small variance value which is significantly smaller than 1 and thus diverged from the prior, means that the dimension has learned something that could contribute in reducing the reconstruction error
4. Having determined variance dimensions σ_i which are significantly small, we can now start and take the corresponding mean of that tuple-pair μ_i in order to get a fix point from which we can start evaluating what the dimension has actually learned
5. We know, that the prior of the inspected dimension has a variance of 1. Therefore, by traversing along that dimension for values in the range of $[-3, +3]$, we can ensure, that we cover most of the space under the Gaussian distribution. This is because the variance σ_i of the i-th entry in the variance vector is significantly smaller than 1
6. We perform this for several positions within $[-3, +3]$, reconstruct our new latent variable z_i and decode the resulting latent variable vector $\mathbf{z}$ where all remain same but the values of the i-th dimension have changed
7. Doing this for every dimension which is significant and comparing the reconstructions, we can get an idea of what attributes the latent representation has learned and how smooth it is by comparing the reconstruction of subsequent latent variable values within the manipulated dimension i

The steps described above represent the case of a continuous latent space with a unit Gaussian and zero mean distribution. For the case of discrete latent space in a discrete VAE the procedure of latent traversal looks as follows

1. Retrieve the latent representation $\mathbf{c}$ which is a sequence of concatenated one-hot vectors $\mathbf{c_i}$
2. Now it is time to inspect the dimensionwise KL term to determine what dimensions are useful to traverse in
3. One the one hand, a latent variable dimension $\mathbf{c_i}$ will have a large KL loss if and only if it deviates from the uniform categorical prior, this indicates, that the i-th

concatenated vector in the sequence of one-hot vectors has learned something that could contribute in reducing the reconstruction error. On the other hand, if a latent variable dimension $\mathbf{c_i}$ will have a small KL loss, it means, that is does not diverge by a lot from the uniform categorical prior distribution and thus has a large entropy and no information helpful to reconstruct the input is contained in this dimension

4. Because he latent space is constructed out of one-hot vectors, if we want to traverse to another point in the discrete latent space for the latent variable we are currently inspecting, then have can simply start decoding all d variations of one-hot vectors the dimension i respectively the latent code $\mathbf{c_i}$ can take on
5. Doing this for every dimension which contributes to the total KL loss and comparing the reconstructions, we can get an idea of what attributes the latent representation has learned and how smooth it is by comparing the reconstruction of subsequent latent variable values within the manipulated dimension i

5.6.2 Disentanglement metric - quantitative inspection

In the following two subsections we will explain the main algorithms behind two different approaches on quantitative evaluation of disentanglement. One algorithm uses a statistics measure by leveraging the ideas of variance analysis on random variables RV, while the other algorithm uses an information theory technique by leveraging the meaning of mutual information MI between two random variables (RV). The goal of disentanglement metrics is to make evaluation of different generative VAE models easier, to be more scientific by using quantitative evaluation tools and finally by the fact, that qualitative evaluation does not scale and is very biased as it needs interpretation of what a latent variable dimension might have learned or not. Moreover, latent traversals are only practical for the image domain. When it comes to written language in NLP it is quite hard to get an idea of what the meaning of a latent variable is and it gets even worse in the case of audio data, where we would have to listen to each latent traversed and reconstructed audio sample.

Variance based metric - statistical approach

We have implemented and experimented with the disentanglement metric which is similar the ideas in [7]. Below we present the main steps of the algorithm

Mutual information based metric - information-theoratical approach

We have implemented and experimented with the disentanglement metric which is similar to the ideas in [5] but in addition to that we have extend the disentanglement metric such we can use it for both latent spaces, continuous ones as well as discrete ones. This can

Algorithm 1: Variance based disentanglement metric

input : M (number of votes), L (number of samples generating a vote), ENC (VAE-encoder for inference), D (dataset) , GF (ground truth generative factors of variation)

output: S (Disentanglement Score)

initialization;
votes = empty 1D array;
V_{mat} = 2D zero array (# ENC.output.dim x # GF.rows);
for $i \leftarrow 1$ **to** M **do**
    `// Construct metric input` $L_{samples}$
    $gf_{idx} \leftarrow$ sample 1 generative factor-index from {1,2, ... ,# GF.rows};
    $gf_{idx,value} \leftarrow$ sample 1 form value from {1,2, ... ,# of variations in gf_{idx}};
    $L_{samples} \leftarrow$ given $gf_{idx,value}$ sample L data points from D;
    `// Fetch latent representation z`
    $z \leftarrow$ ENC(x);
    `// Calculate empirical variances`
    $s \leftarrow$ std(z, axis = 0);
    $z_{normal} \leftarrow z/s$;
    $var_{empirical} \leftarrow$ var(z_{normal}, axis = 0);
    `// Determine which generative latent variable varies the least`
    $z_{idx} \leftarrow$ argmin($var_{empirical}$);
    `// Now we generate the vote as pair of latent variable index and generative factor index`
    vote $\leftarrow$ (z_{idx}, gf_{idx});
    votes $\leftarrow$ vote;
`// Construct vote matrix with majority vote classifier`
V = len(votes);
for $i \leftarrow 1$ **to** V **do**
    `// Access values of vote tuple i`
    vote $\leftarrow$ votes(i);
    $V_{mat}(vote(0), vote(1)) \leftarrow V_{mat}(vote(0), vote(1)) + 1$;
    `// Determine max in each row`
    $max_{values} \leftarrow$ max(V_{mat}, axis = 1);
    $sum_{correct} \leftarrow$ sum(max_{values});
    $S \leftarrow sum_{correct}/M$;

be achieved by performing an argmax operation and mapping the discrete latent representation of concatenated one-hot vectors into a latent representation which preserves all the information as argmax values of each one-hot vector. In that way, using the argmax operation helps us in reducing the sparsity of the latent representation vectors $\mathbf{c}$ and allow us to use the disentanglement metric. Below we present the main steps of the algorithm

Both presented models are not a totally generalized way to inspect disentanglement, as one is not really axis-aligned and both of these metrics have the drawback, that we have to know the ground truth generative factors. This is for many datasets simply no helpful assumption and thus work needs to be done towards disentanglement metrics which work without the knowledge of these ground truth factors of variation and which most of all are applicable across different data domains and latent spaces (continuous, discrete).

Algorithm 2: Mutual information based disentanglement metric

input : N (number of data samples), ENC (VAE-encoder for inference), D (dataset) , GF (ground truth generative factors of variation)
output: S (Disentanglement Score)

initialization;
N-indices ← indices(N);
S-scores = empty 1D array;
`// Fetch N samples as ground truth generative factors representation`
gf ← sample generative factor vectors from GF with N-indices;
`// Fetch model input` x `z`
x ← sample x vectors from GF with N-indices;
`// Fetch latent representation z`
z ← ENC(x);
n_{GF} ← # GF.columns;
n_Z ← # z.columns;
for $gf_{idx} \leftarrow 1$ **to** n_{GF} **do**
    `// Fetch column in generative factors array`
    gf_{col} ← GF(,gf_{idx});
    MI-scores = empty 1D array;
    for $z_{idx} \leftarrow 1$ **to** n_Z **do**
        `// Fetch column in latent representation array`
        z_{col} ← GF(,z_{idx});
        `// Calculate mutual information between two RVs`
        MI ← I(z_{col},gf_{col});
        MI-scores ← MI;
    sortedMI-scores ← sortMax(MI-scores);
    `// Fetch 1st and 2nd largest MI scores`
    firstMI ← sortedMI-scores(0);
    secondMI ← sortedMI-scores(1);
    MI-score ← abs(firstMI - secondMI);
    S-scores ← MI-score;
S ← mean(S-scores);

Chapter 6

Architecture implementations and experiments

In this chapter, we will perform the experiments using the β-VAE model with a continuous latent space and a discrete latent space. Furthermore, we examine the differences when using different embedding sizes and embedding methods. Finally, we evaluate the disentanglement and in parallel to that evaluate a new metric *generative factors accuracies* which in parallel to metrics such as the reconstruction loss give a better idea of what the model is currently able to reconstruct and for which attributes it encounter difficulties in learning them.

6.1 Experiments with continuous latent space β-VAE

In this section we will analyze the continuous VAE in regard to disentanglement, reconstruction and latent loss. Moreover, will we analyze the correlation between the latent space and the ground truth generative factors of variation by using a 'mutual information correlation matrix'.

6.1.1 Reconstruction Loss

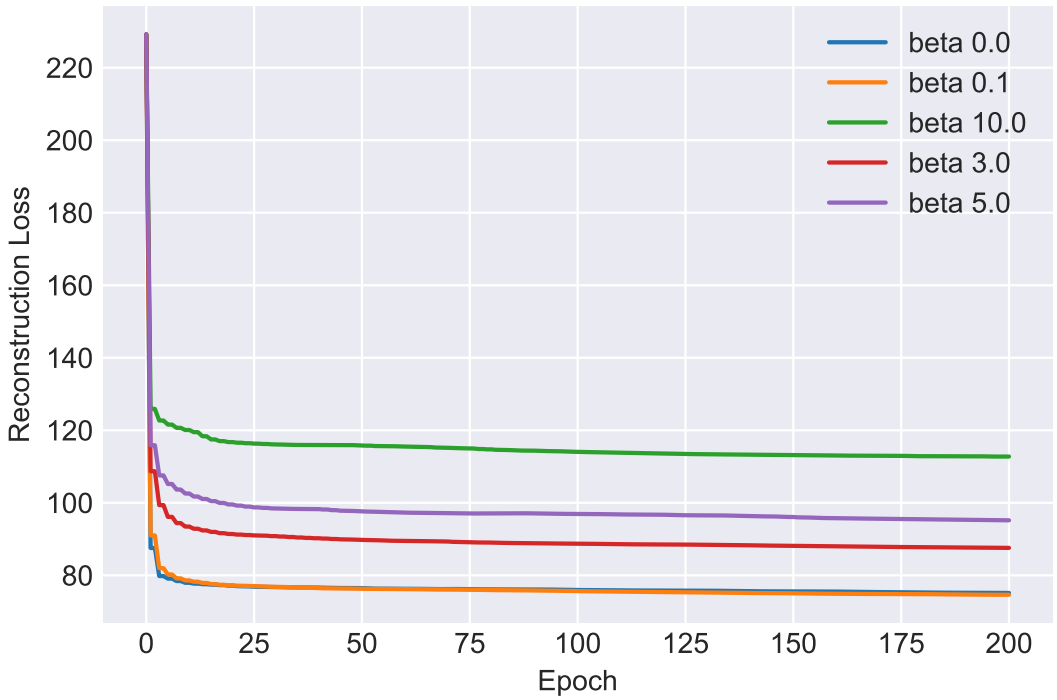


Figure 6.1: Reconstruction loss of VAE for various β values

In Figure 6.1 we can see, that the hyperparameter β worsens the reconstruction loss, the larger β becomes. It clearly acts as a regularizer which we expected due to the well-known trade-off between disentanglement and reconstruction performance of VAEs.

6.1.2 Latent Loss

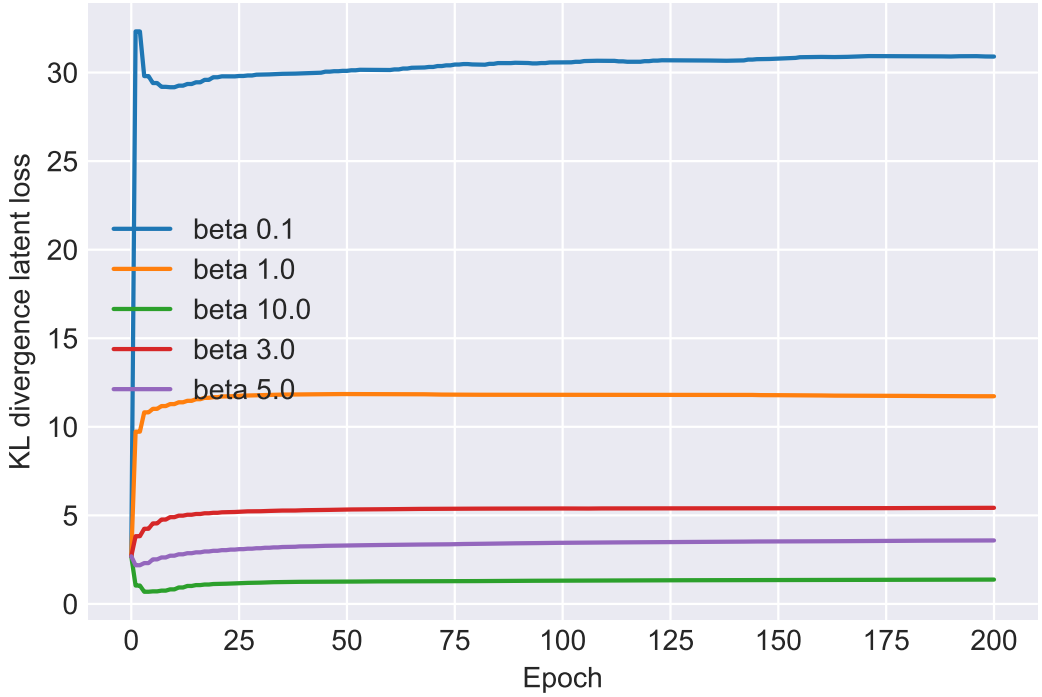


Figure 6.2: KL divergence loss between posterior and prior for various β values

In Figure 6.2 we can see, that the when the hyperparameter β increases the KL divergence is getting smaller. This is because the β increases the KL divergence penalty and thus the model tries to circumvent that by moving again closer to the prior distribution. It clearly shows that β causes a stronger bottleneck because the KL divergence is also representing the amount of information, measured in *nats*, our bottleneck is able to encode and transfer to the decoder.

6.1.3 MIG disentanglement metric

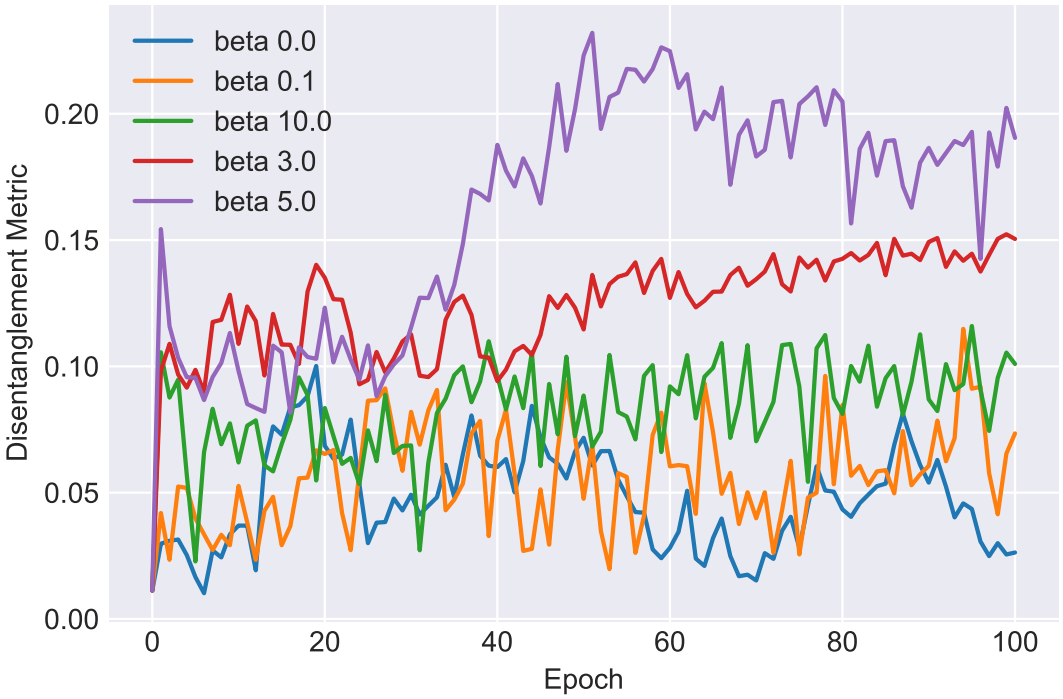


Figure 6.3: Disentanglement performance scores for various β values

By analyzing Figure 6.3 together with Figure 6.1, we can observe, that a lower reconstruction error does not necessarily mean, that the latent space is more entangled and therefore better structure. Further, we get to see that the highest disentanglement score is achieved by the VAE with $\beta = 5$. Once, the hyperparamter gets to large, we drastically drop in both, disentanglement score as well as reconstruction performance, which is getting worse.

6.1.4 Evaluation of learned generative factors for differing β

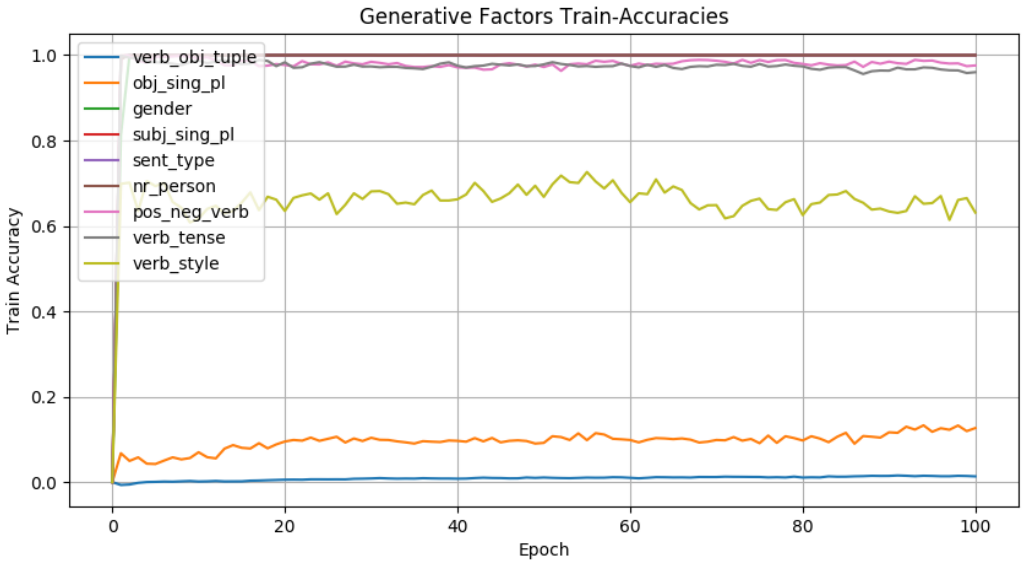


Figure 6.4: Generative factors accuracies for β equal to 0.1

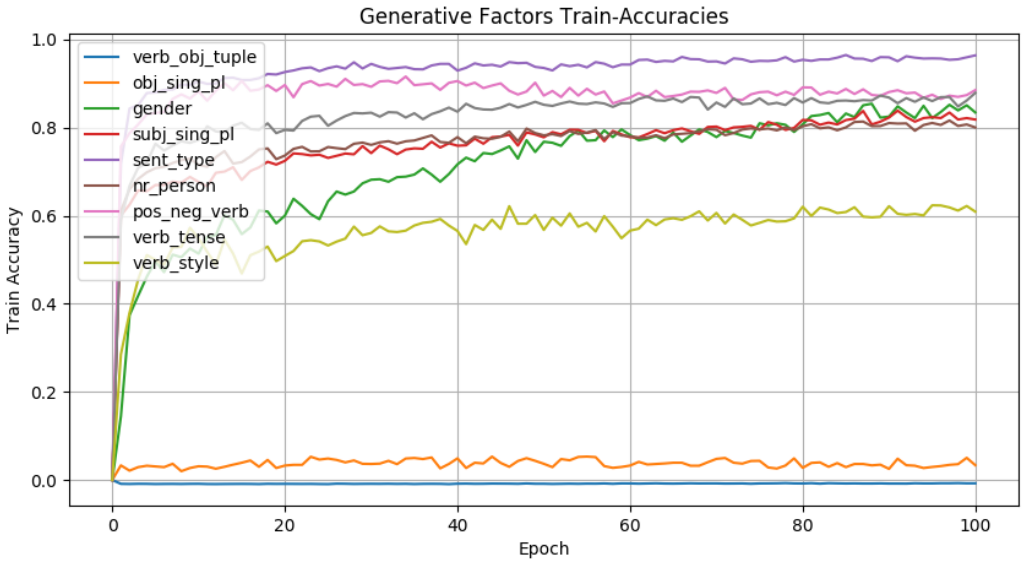


Figure 6.5: Generative factors accuracies for β equal to 3.0

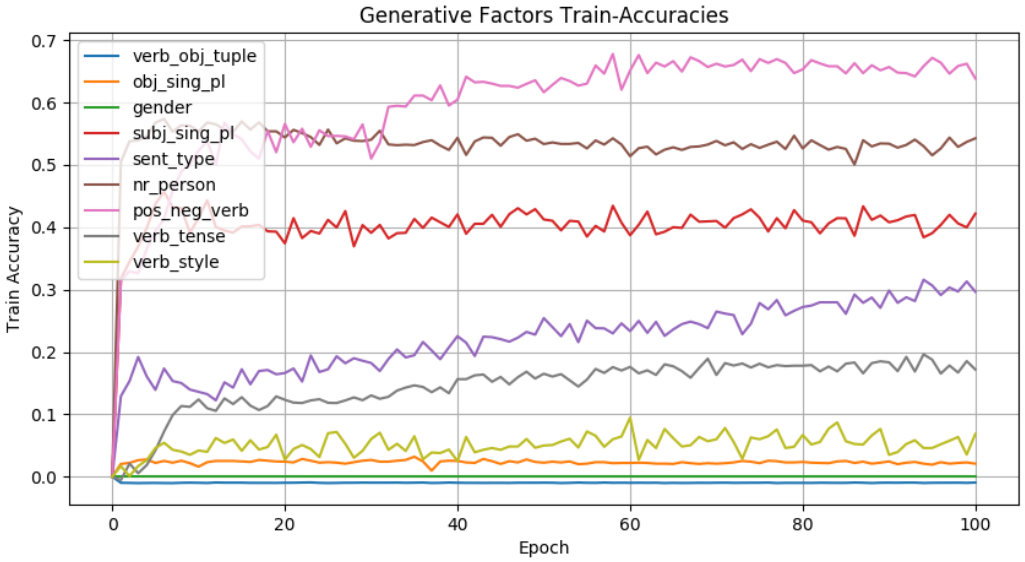


Figure 6.6: Generative factors accuracies for β equal to 10.0

Looking at all three Figures 6.4, 6.5, 6.6 together we can see a clear message. There is a sweet spot for the hyperparameter β for which the VAE learns the generative factors in the best way. Above, this is the case in Figure 6.5 with β equal to 3.0. Moreover, we can see that regardless of the value of β, there are some generative factors which can not be learned. In this case these are the verb-object-tuple as well as the singular and plural version of the object. So clearly, both factors are related to each other as they need to incorporate the right object. As this generative factor is the most asymmetric one, the network has difficulties in learning it and probably prefers to learn other generative factors which only two to three different form values instead of the hardest one. This makes sense as it is a meaningful strategy, when capacity is a limited resource.

6.1.5 Analyzing the relationship between latent variables and generative factors

Finally, we will inspect the correlation between single latent variable dimensions and generative factor dimensions by plotting the mutual information correlation matrix at the beginning of the training and for different β values after the training phase is finished.

Below, we can observe the relationship between generative factors and latent variables before starting to learn anything about the data distribution.

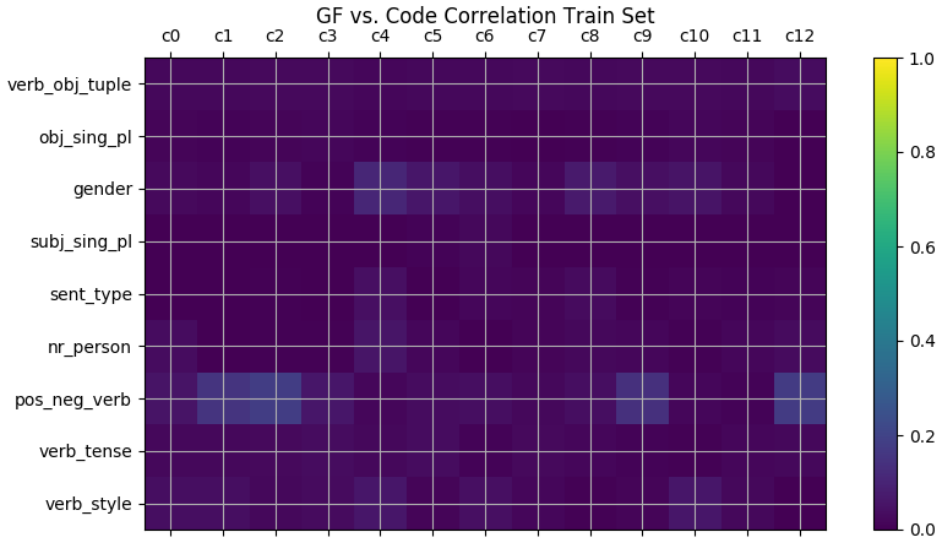


Figure 6.7: Pre training MI correlation matrix between all generative factors and latent variable dimensions, same for any β

As we ca see in Figure 6.7 there is no existing trend or correlation between those two arrays; this is what we would have expected and it is same for any β value because this measurement was done prior to the training phase.

Now, let us have a look at what happens after finishing the learning process and inspecting different hyperparameter values for β.

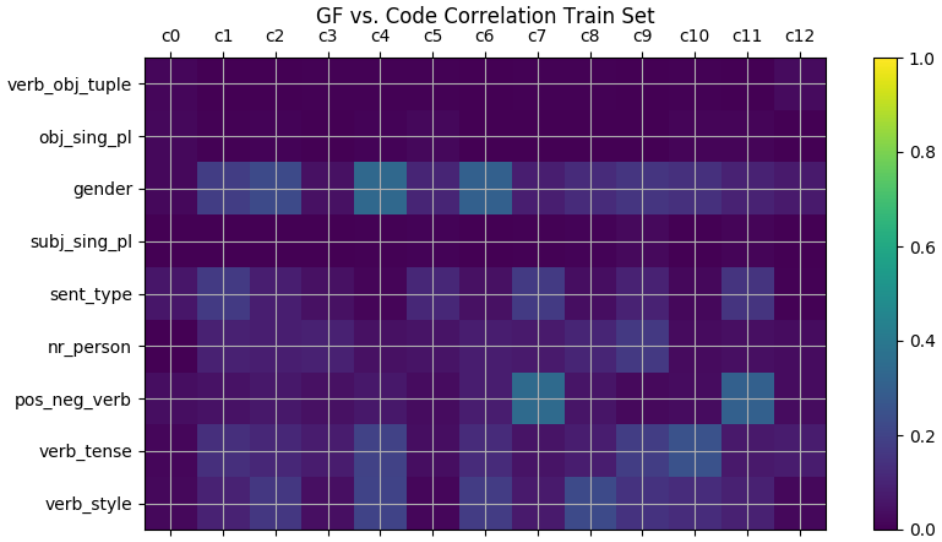


Figure 6.8: Post training MI matrix between all generative factors & latent variables, β equals 0.0

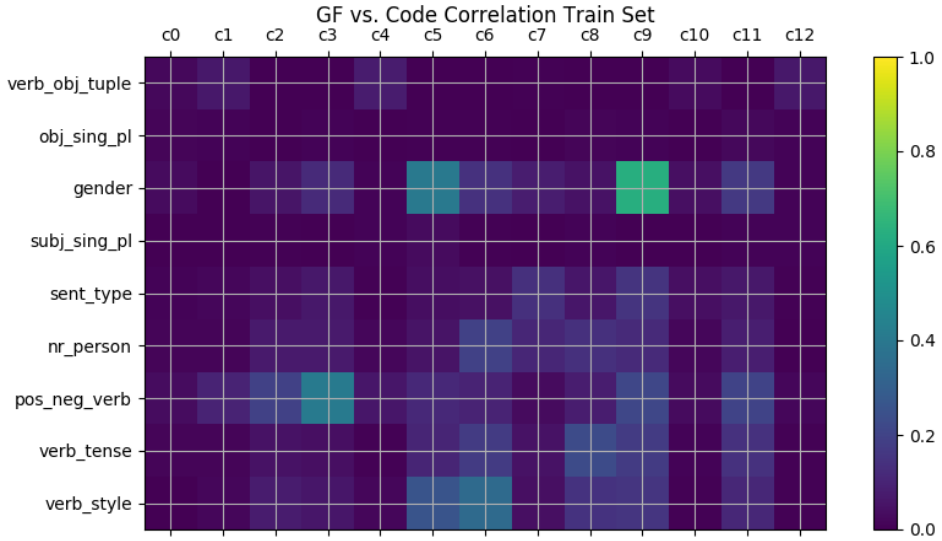


Figure 6.9: Post training MI matrix between all generative factors & latent variables, β equals 1.0

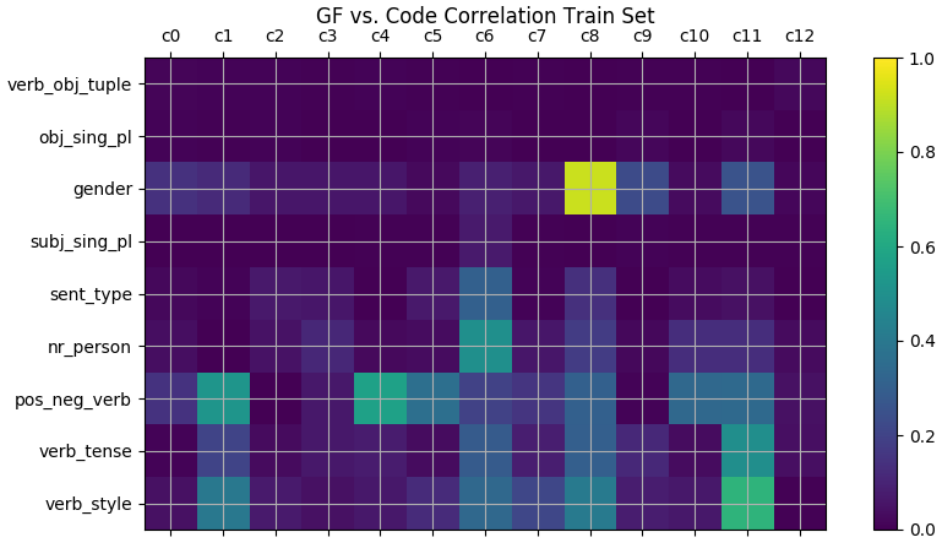


Figure 6.10: Post training MI matrix between all generative factors & latent variables, β equals 5.0

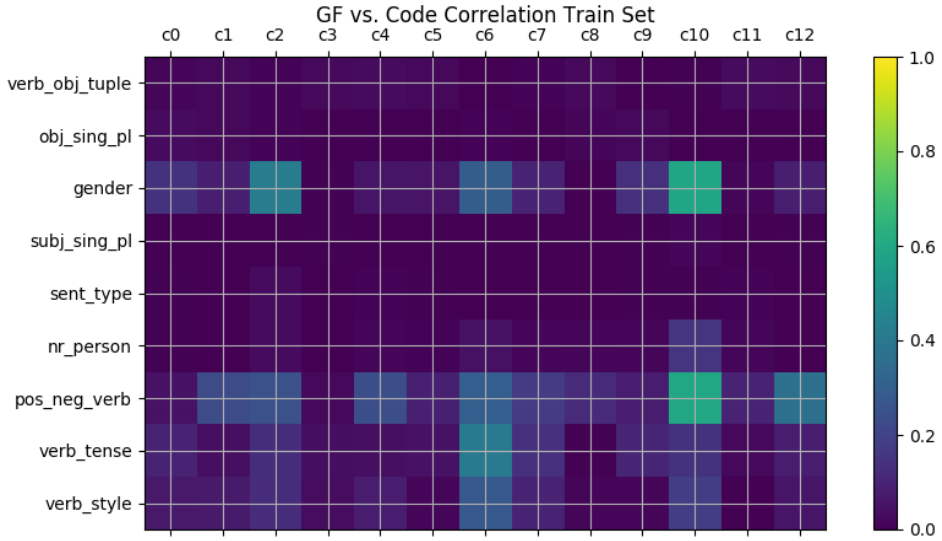


Figure 6.11: Pre training MI matrix between all generative factors & latent variables, β equals 10.0

Looking at the Figures 6.8, 6.9, 6.10, 6.11 we can observe two phenomena. FIrst, we see, that the AE, which corresponds to the Figure 6.8 clearly shows the fact, that it is possible to learn a high quality reconstruction but at the same time having learned a representation latent space with no interpretable character. Moreover, the fact that the reconstruction is very good but there are no hot spots in the MI matrix means, that the information is spread all over the latent space and is equally distributed. Thus latent space has no incentive to develop a smooth placement between similar sentences. Second, we can see, that the higher the β values gets, the more factorized does our latent space become as we start to see stronger relationships between the latent space and several generative factors. This is especially the case in Figure 6.10 where we see a strong relationship between the gender generative factor and the latent code dimension c_8. Moreover, we could map the verb style to be contained within the latent variables c_1 and c_1 and the number of persons within the latent variable c_6. Thus, we can see, that the hyperparameter β indeed has some effect on disentangling the latent space compared to the AE ($\beta = 0$) or the VAE ($\beta = 1$) models.

6.2 Experiments with discrete latent space β-VAE

In this section we will analyze the discrete version of the β-VAE in regard to disentanglement, reconstruction and latent loss. Moreover, will we analyze the correlation between the latent space and the ground truth generative factors of variation by using a ‘mutual information correlation matrix‘.

6.2.1 Reconstruction Loss

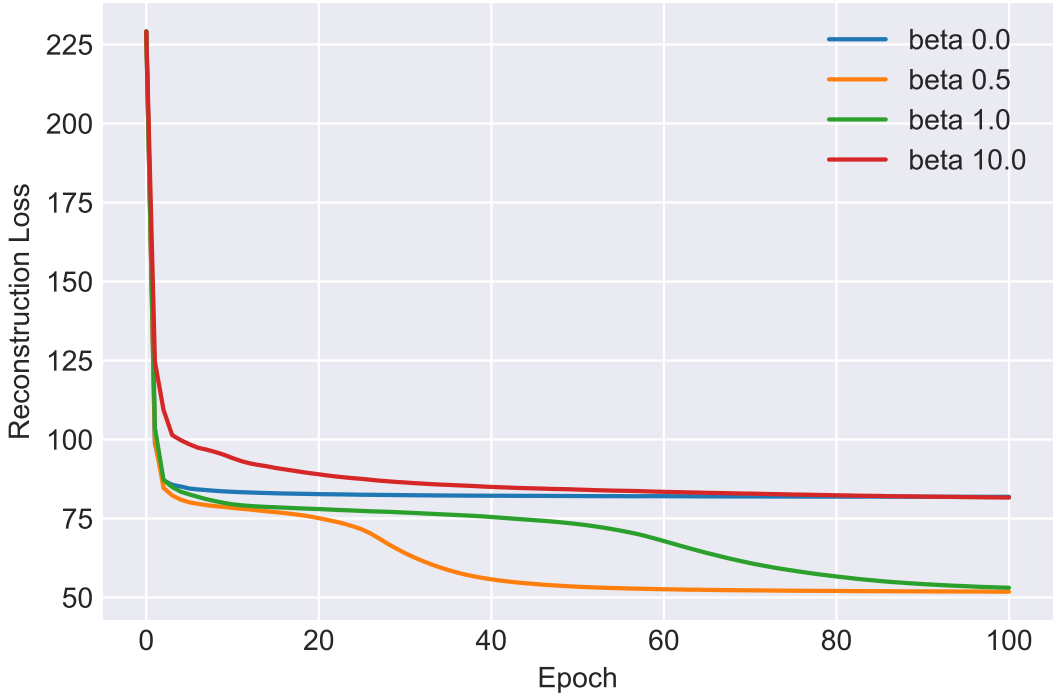


Figure 6.12: Reconstruction loss of discrete VAE for various β values

In Figure 6.12 we can see the probably most unexpected outcome of our research experiments while working with the VAE for NLP in a discrete latent space. Usually, the rule is, that a higherβ always leads to a worse reconstruction performance and therefore, the AE always outperforms the VAE when it comes to maximizing the log-likelihood. Surprisingly, we have discovered quite the opposite when it comes to the VAE living in a capacity limited and discrete latent space. Here, for β values around 0.5 to 1.0 the discrete VAE is performing by far better than the AE as can be seen on the scale, the difference with about 25 is remarkable. Of course, there is still a trade-off between β values and reconstruction error as can bee seen by looking at the red curve for β equal to 10. The reconstruction loss bounces back and becomes similar or worse compared to the AE model.

6.2.2 Latent Loss

Now as we have made this interesting discovery, let us inspect what the latent loss for the discrete VAE looks like for various values of β.

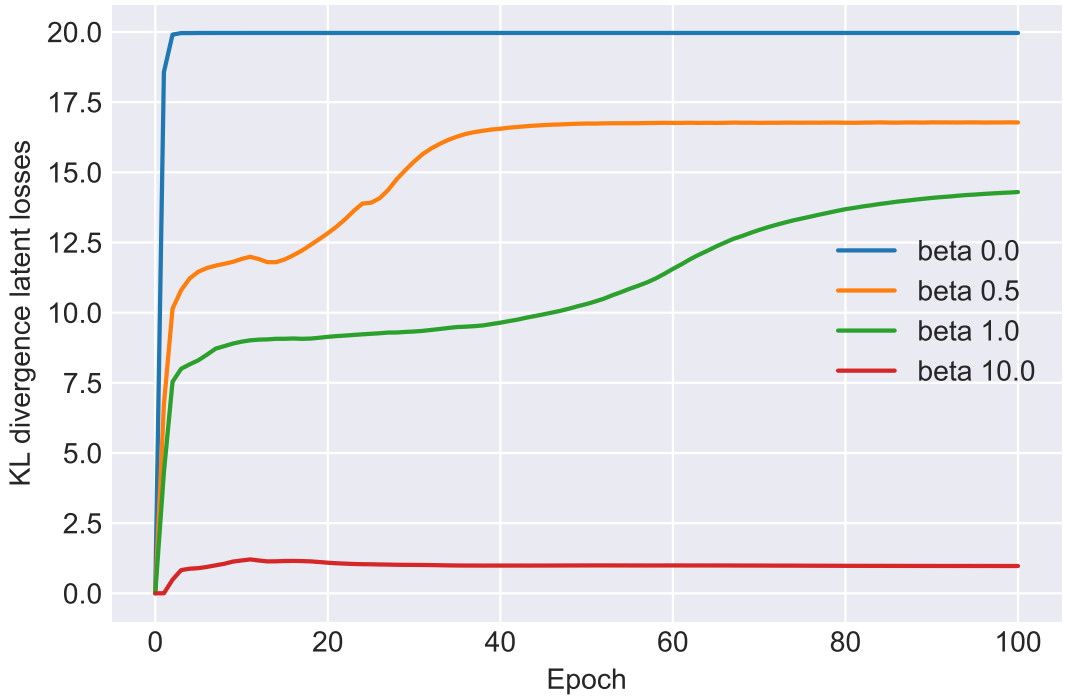


Figure 6.13: KL divergence loss between posterior and categorical prior for various β values

Above in Figure 6.13 we can see, what we are already used to from the continuous β-VAE model. The larger the hyperparameter becomes, the smaller will our KL divergence loss be. This is because a large hyper parameter as coefficient in front of the posterior KL divergence term between our categorical distribution sampled from the Gumbel-Softmax and the categorical uniform prior distribution forces our model to better remain similar to the prior as the penalty for being 'different' is large. Finally, when looking the reconstruction Figure 6.12 above, we can observe that the orange and green curve end up with almost similar reconstruction performance. But still, from 6.13 we have to notice first, that the model with larger β has a stronger bottleneck limitation and thus less effective capacity. As we can see by looking at the final KL divergence value at the end of epoch 100. Nonetheless, the $\beta = 1.0$ VAE model is able to compete with the orange $\beta = 0.5$ model. From that, we have to conclude, that the latent space is better disentangled, interpretable

and compressed. Similar results behaviour is well studied for data in the image domain but it looks as if this behaviour can be transferred to discrete VAEs which operate on written language and using a categorical latent space representation.

6.2.3 MIG disentanglement metric

In order to close this section, let us finally have a look at the disentanglement metric for each of those regularized discrete VAE models in order to get an idea of what we the disentanglement really looks like.

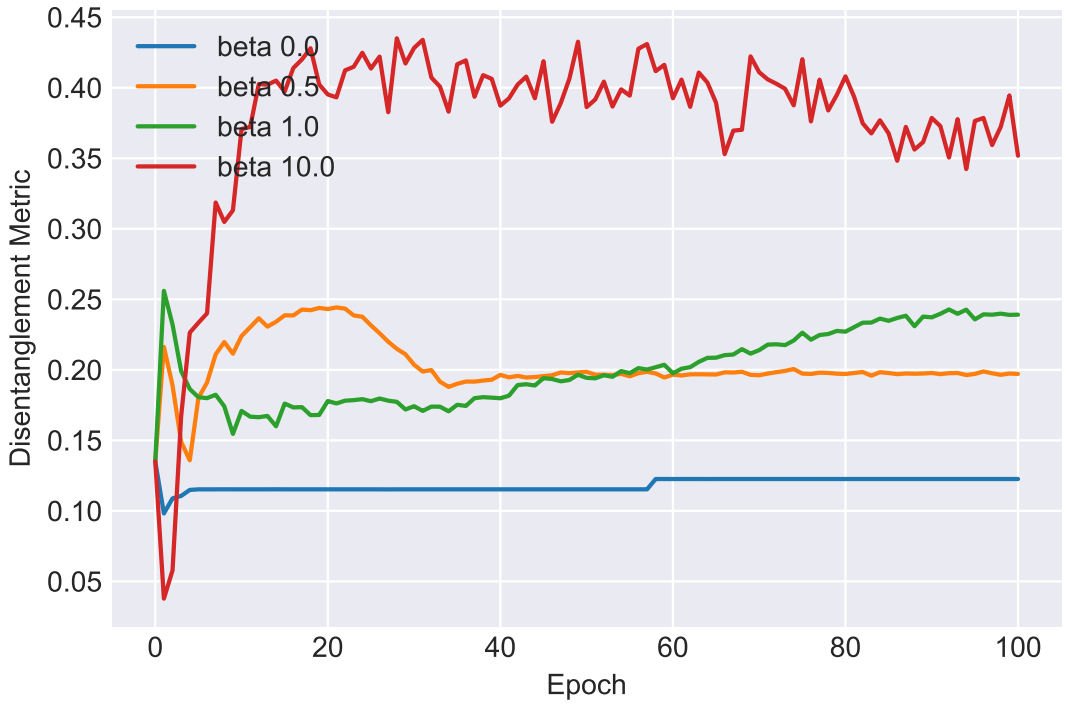


Figure 6.14: Disentanglement performance scores for various β values

By analyzing Figure 6.14 together with Figure 6.12, we can observe, that a high disentanglement metric score does not necessarily mean, that the latent space is better suited for reconstructing the input, it simply means, that the code is more factorized but this factorization comes with the price of reducing the reconstruction quality. Further, we see that the highest disentanglement score is achieved by the VAE with $\beta = 10$. Thus, when comparing to the behaviour in the continuous latent space VAE, our discrete VAE needs

larger β values in order to be encouraged to start disentangling the latent representation $\mathbf{c}$ and generate a smoother latent representation. It is remarkable how much larger the disentanglement score is compared to all other models.

6.2.4 Evaluation of learned generative factors for differing β

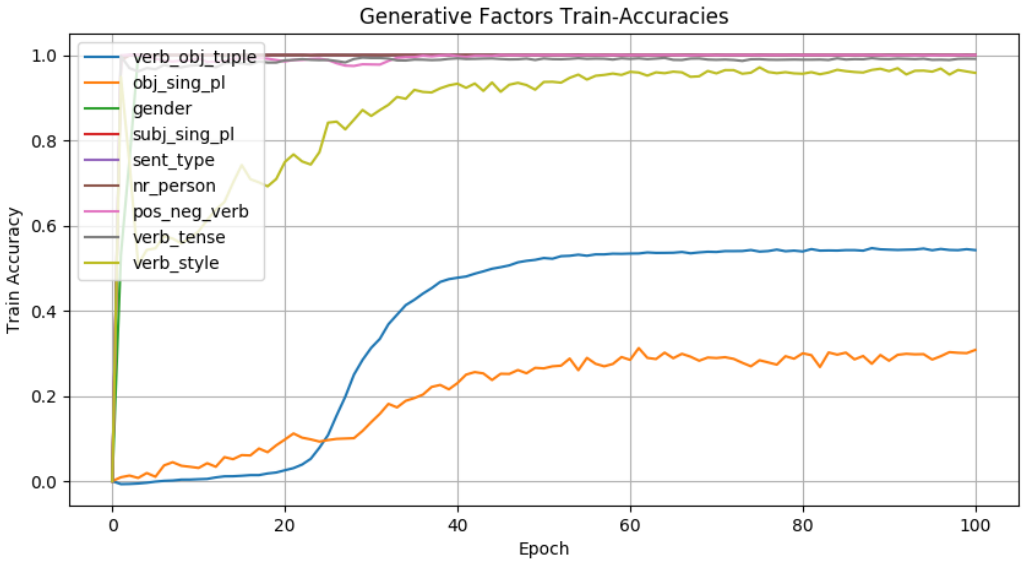


Figure 6.15: Generative factors accuracies for β equal to 0.5

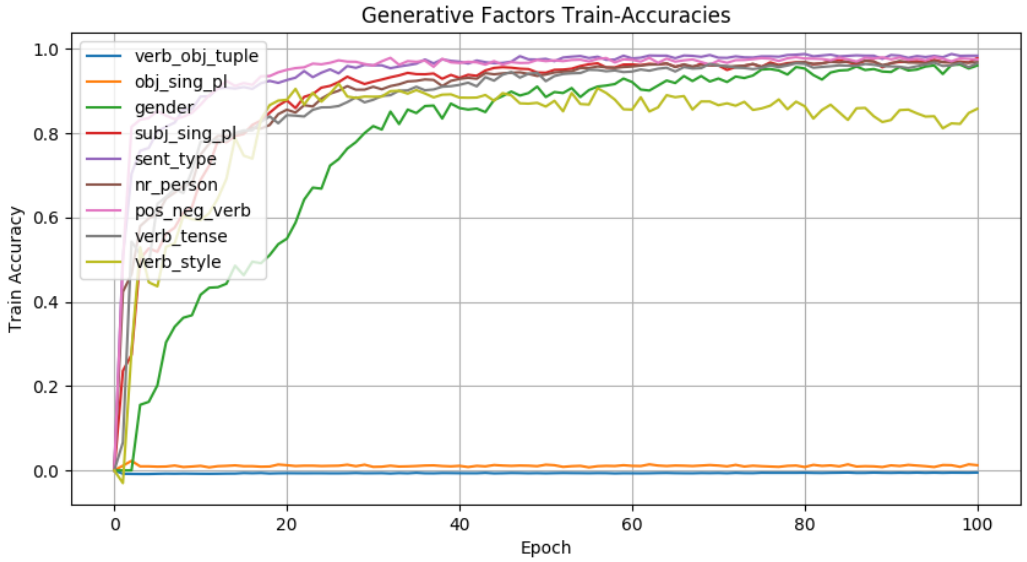


Figure 6.16: Generative factors accuracies for β equal to 5.0

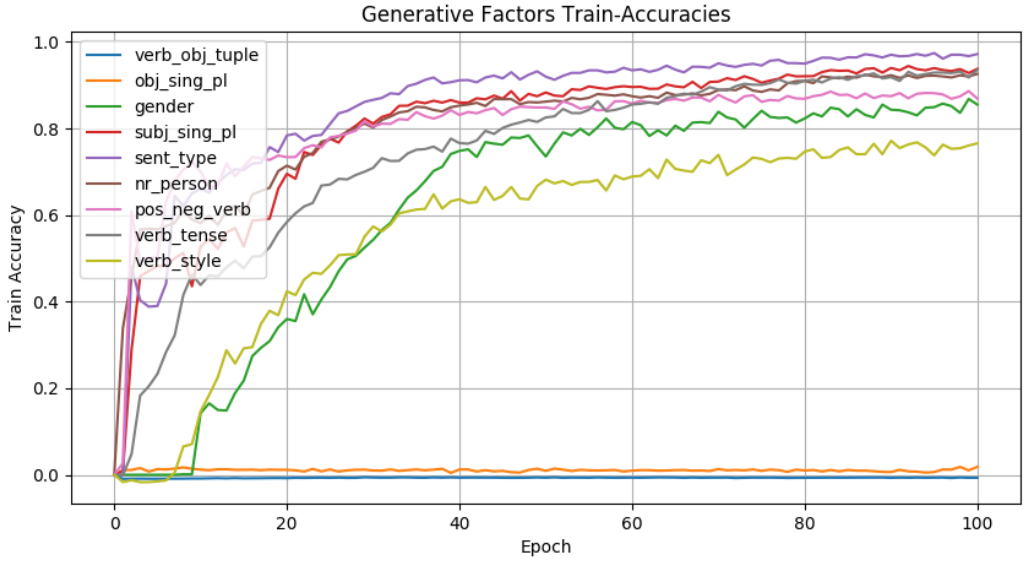


Figure 6.17: Generative factors accuracies for β equal to 10.0

Looking at all three Figures 6.15, 6.16, 6.17 together we can see something very unintuitive. Looking at Figure 6.15, we can see, that it is the only model which is capable of learning

the verb-object-tuples as well as the singular and plural version of the objects. But more interestingly, it decides upon learning these generative factors only after a very long period. The model is already in epoch 20, when it suddenly becomes aware of this generative factor and improves upon that performance. It achieves that, without even paying with performance degradation in regard to all other generative factors. We are unsure about how this is achieved but the only explanation is, that the other latent variables are gradually restructured, which emits new capacity, which can be used in order to encoder another generative factor of variation. The Figures with increased β namely 6.16 and 6.17 are not able to learn these two hard to learn generative factors and they unluckily start to learn all nine factors of variation at the same time. This is often an indicator of a very entangled latent representation of the VAE.

6.2.5 Analyzing the relationship between latent variables and generative factors

Finally, we will inspect the correlation between the discrete, categorical code and the generative factor dimensions by plotting the mutual information correlation matrix at the beginning of the training and for different β values after the training phase is finished.

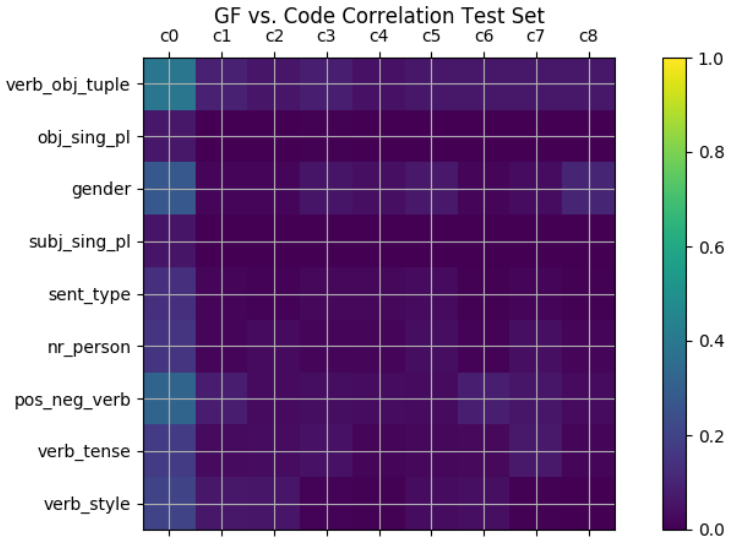


Figure 6.18: Pre training MI correlation matrix between all generative factors and discrete latent dimensions, same for any β

Above in Figure 6.18 we can observe the relationship between generative factors and latent variables before starting to learn anything about the data distribution. We can see, that

is not much different to the distribution we saw using the continuous latent space with a Gaussian Normal distribution as prior for all but one dimension. The first one; which in this model architecture is consisting of roughly 1100 dimensions such that the model would have the possibility to encoder the verb-object-tuple into a single latent dimension, in effect 1100 different categories or one-hot vectors. It turns out, that there is some mutual information between the largest dimensional latent code and all other generative factors of variation. What we just described can be explained the following Figure 6.19 while it might look not correct, connecting several generative factors with a single latent variable dimension very likely reduces the disentanglement score significantly.

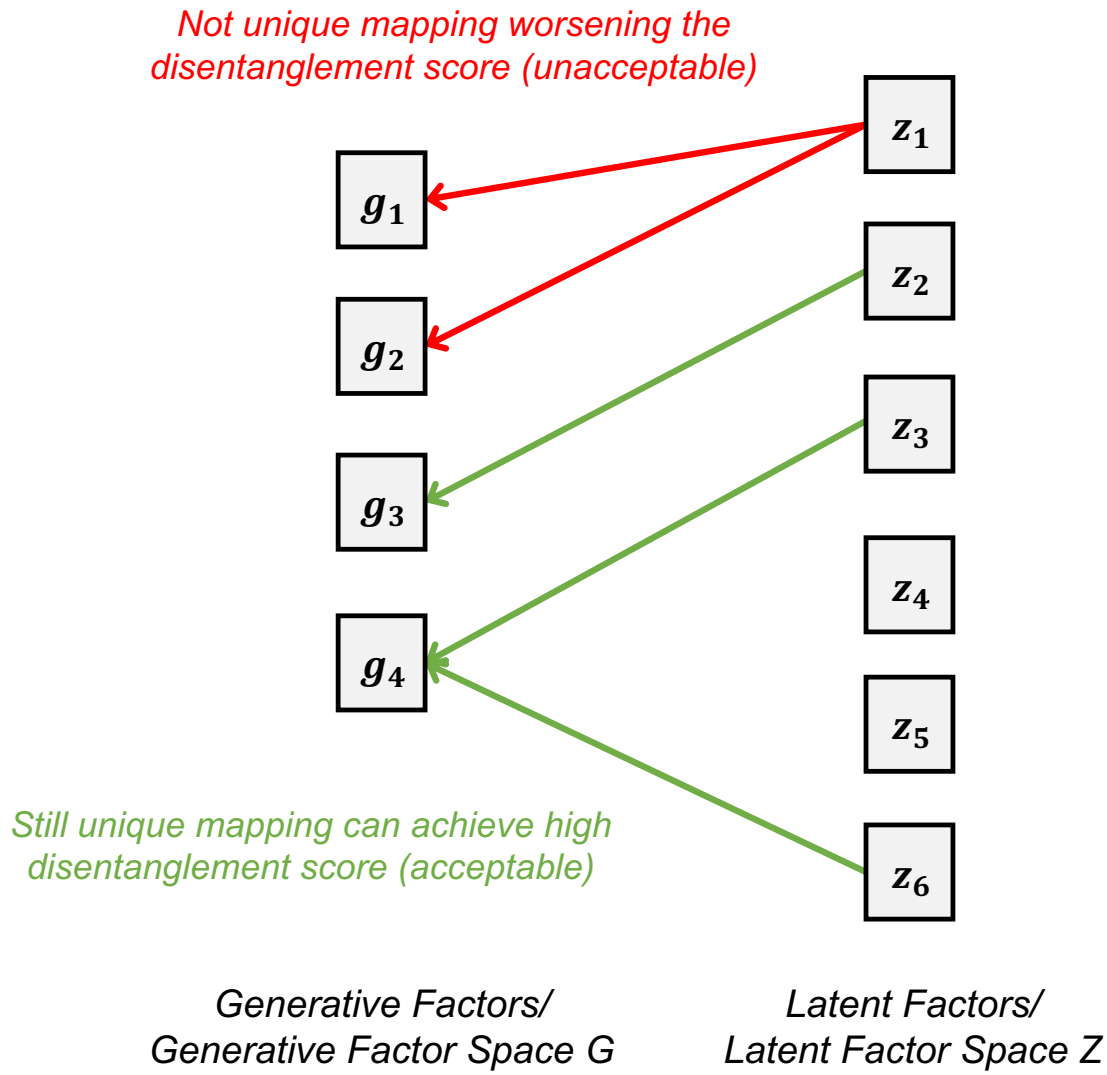


Figure 6.19: Two different types of cases which affect disentanglement

As we ca see in Figure 6.7 there is no existing trend or correlation between those two arrays; this is what we would have expected and it is same for any β value because this measurement was done prior to the training phase.

Now, let us have a look at what happens after finishing the learning process and inspecting different hyperparameter values for β.

Figure 6.20: Post training MI matrix between all generative factors & latent variables, β equals 0.0001

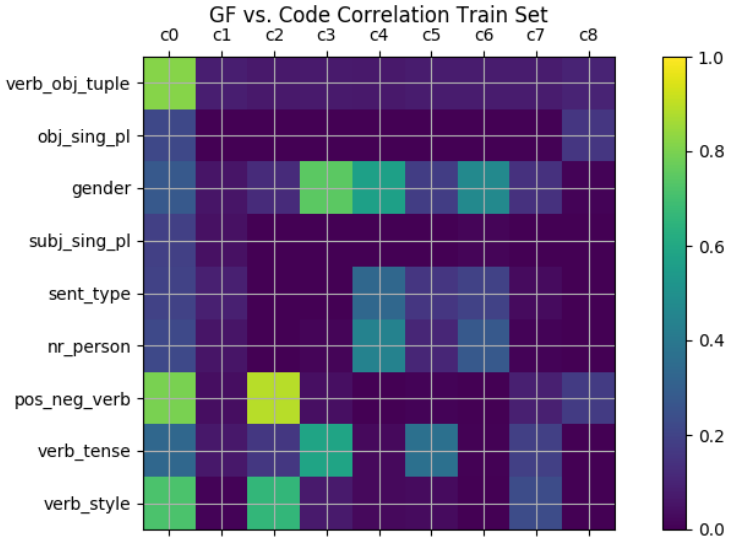


Figure 6.21: Post training MI matrix between all generative factors & latent variables, β equals 0.5

Figure 6.22: Post training MI matrix between all generative factors & latent variables, β equals 5.0

Looking at the Figures 6.20, 6.21 and 6.22 we can observe two properties. FIrst, the already observed bias when checking the pre training MI matrx is only solved for certain β values, other wise this bias remains. The model which best this meaningless strong relationship is the VAE with β equal to 0.5. We can clearly see, that the first code dimension with 1100 units is highly correlated with the verb-object tuple. This is very interesting, because what the model in Figure 6.20 learns to encode into this dimension does not make quite sense. The gender is only 2 categorical yet, the model decides to put the gender information, probably robust but not intelligently structured, into this dimension. Second, the higher β becomes, the less correlating latent dimension do we get to see. This means, that the penalty through the KL divergence term is so strong, that the model decides, cf. 6.22, to encode the most important generative features, which by looking at this mutual information correlation matrix are all generative factors such that they fit into the large code dimension of 1100 units. This makes sense, because the model is getting penalized rather by the amount of differnt code dimensions it starts to use.

6.3 Calculating the error feedback with one-hot and word2vec embeddings

Depending on the input representation type; which can be one-hot or pre-trained word embeddings, we have to calculate the error in two different ways.

6.3.1 Loss calculation - one-hot

For the case of text representation as one-hot, we do specify the loss as shown in Figure 6.23 below:

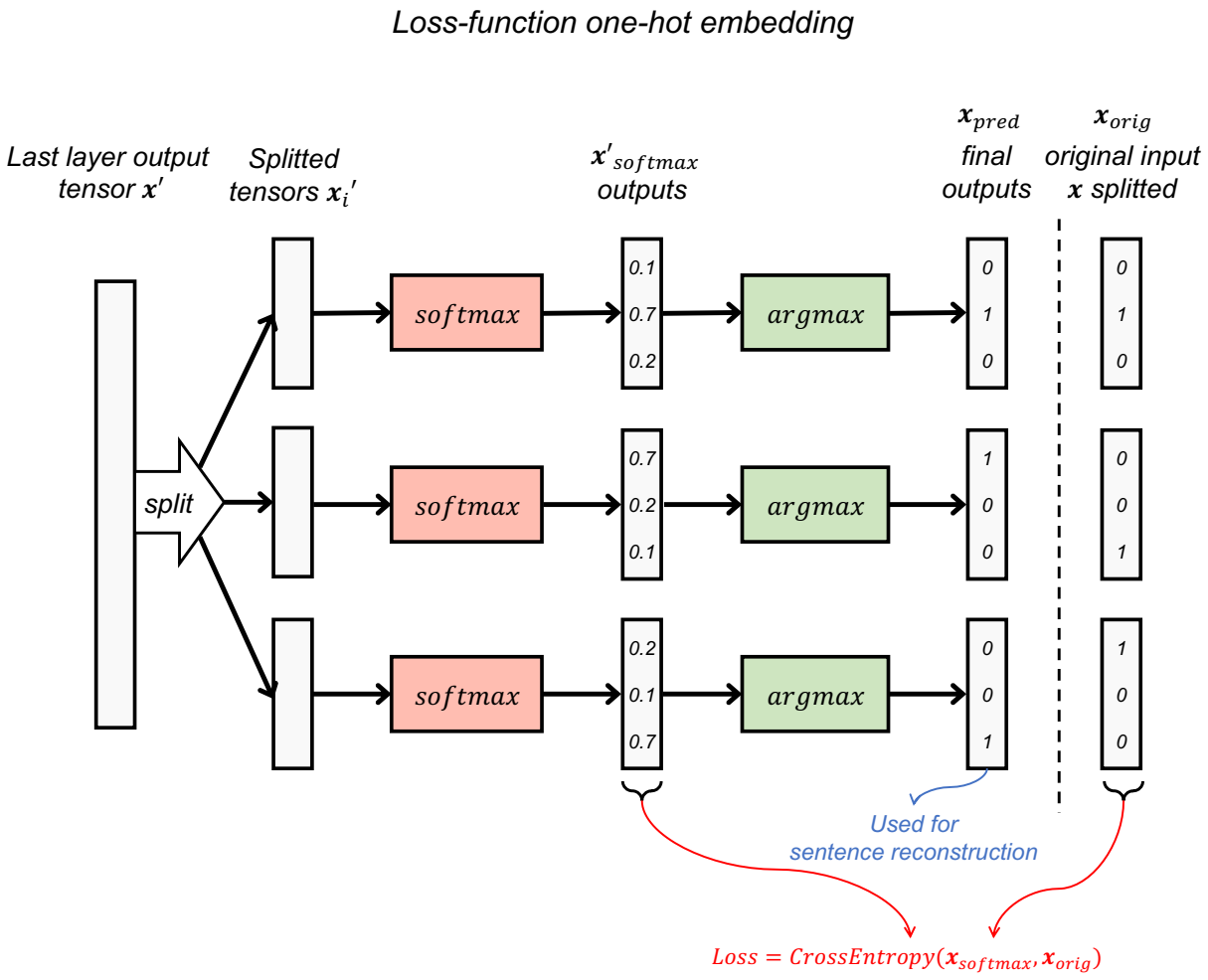


Figure 6.23: Determining the backpropagation error for one-hot representations

6.3.2 Loss calculation - *word2vec*

For the case of text representation as continuous and dense word embeddings, we do specify the loss as shown in Figure 6.24 below:

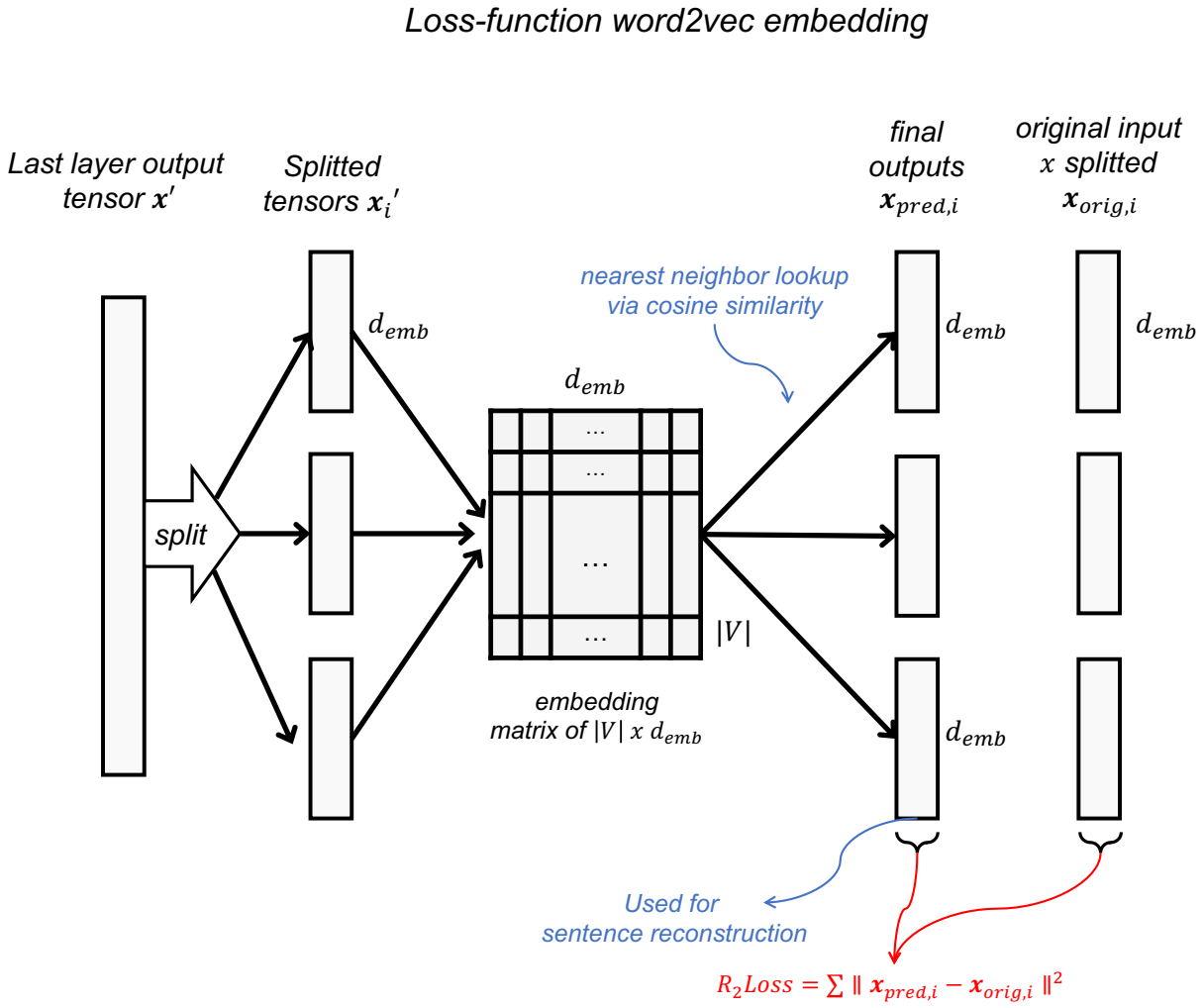


Figure 6.24: Determining the backpropagation error for *word2vec* representations

6.4 Sanity check of the generative factors accuracies

Below we show, that the *generative factors accuracies* can be applied to our architecture and dataset. It helps in understanding during which training epoch what generative factors the model has learned so far to reconstruct correctly. We are comparing a model in which

we randomly 'scramble' the latent representation $\mathbf{z}$ of the data input $\mathbf{x}$, thus making it impossible to learn any generative factor, with another model in which we left out this scrambling function. The scrambling function was performed by subsequently performing the following functions upon the correct latent representation $\mathbf{z}$. Moreover, we will make a comparison between a VAE architecture which uses only MLPs against a VAE architecture which uses only CNNs. We have experimented with RNNs within the VAE but they tend to ignore the latent space and collapse to a simple RNN language model (LM) therefore we mainly focused on comparing MLP as baseline model against CNN as alternative variation. This posterior collapse and the difference to a distribution with large KL loss, which means having learned information and characteristic of the data distribution, is shown below in Figure 6.25

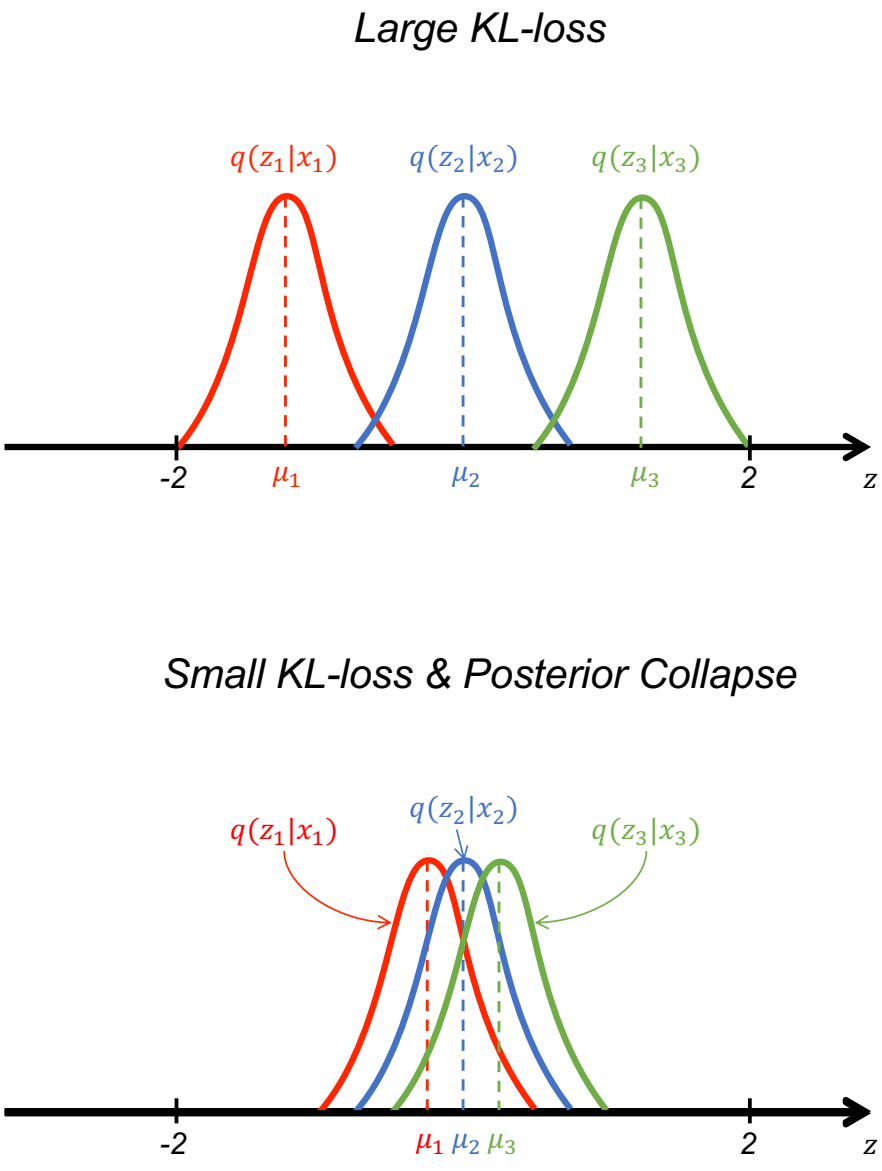


Figure 6.25: Posterior collapse due to decoder ignorance

1. Rotation of rows in **z**
2. Transposition of **z**
3. Rotation of columns in **z**
4. Reverse transposition of **z**

The result for the scrambled and not scrambled latent space on the performance of the average word level accuracy can be seen in Figure 6.26.

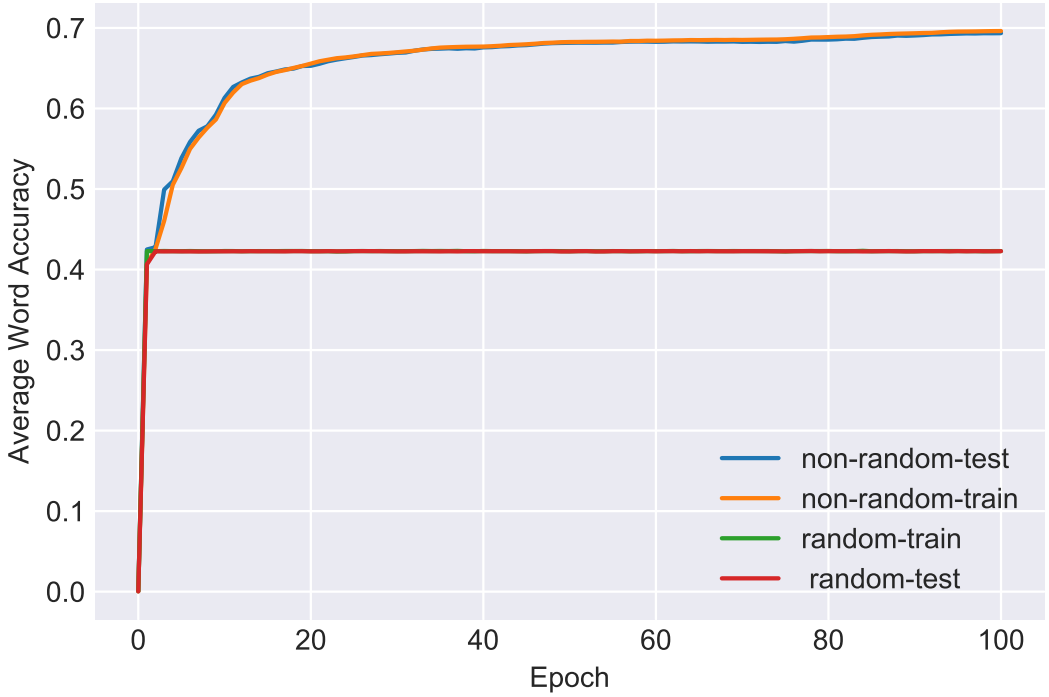


Figure 6.26: Word level accuracy for a random and not-random latent space representation

In Figure 6.26 we can see, that despite the randomness, the accuracy is remaining pretty high. Therefore, when we have a loot at the actual learned generative factors with out new metric, we can see the following as shown in 6.27.

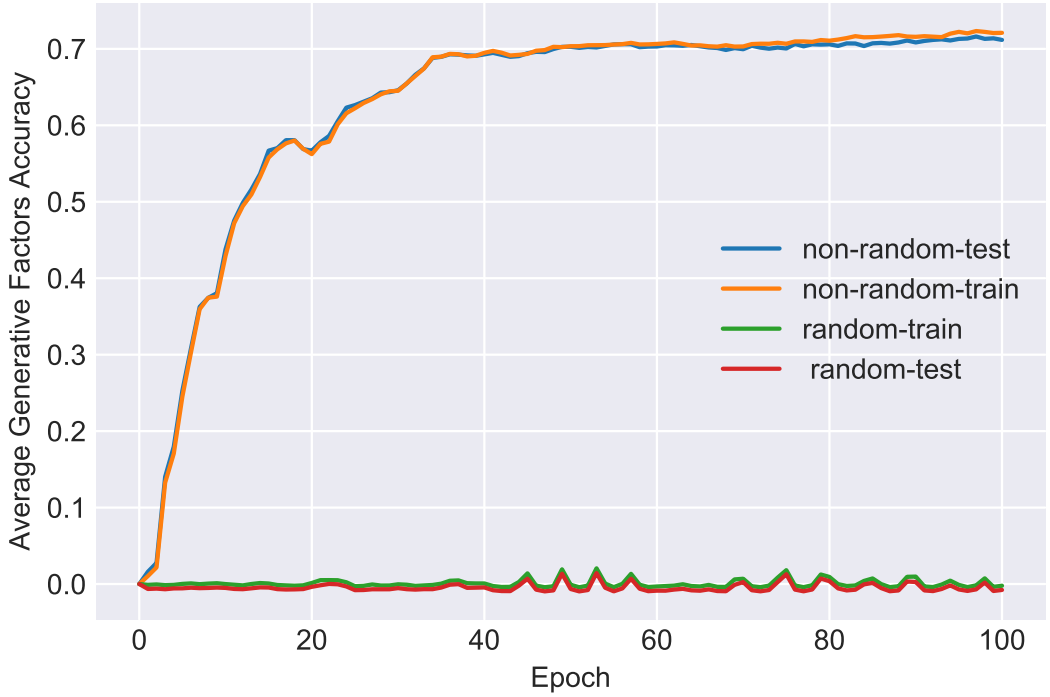


Figure 6.27: Generative factors accuracies for a random and not-random latent space representation

Now, from Figure 6.27 we can see, that the is a huge gap between the word level accuracy and the generative factor accuracies. Which is the reason, we are evaluating our model rather on this performance instead of the classical ‘overlap‘ metric between two sequences of tokens which do contain redundancy term such as the word ‘the‘, which appears in every sentence.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we have started the first steps in understand the β-VAE framework for natural language, by taking the same avenue as it has been done in the image domain. We first created a toy dataset, similar to the dSprites dataset by DeepMind Research, which we termed in analogy as dSentences and next we evaluated different model variations, embedding dimensions and latent spaces. We gained a lot of interesting insights into how to work with discrete sentences, how to apply them on different neural network NN setups within the framework of variational inference as a graphical model. One of the more interesting discoveries during the last six months, hat we learned a lot about how not to approach a problem and construct a dataset. We have gathered many ideas upon what could be improved with regards to the dataset. One of the key findings in our work, is the fact, that state of the art is terming the expression 'trade-off between reconstruction and disentanglement'. This is not quite true, as we prove the opposite and show that when using a fully discrete β-VAE, the goal of maximizing both desiderata can be well achieved. We have shown this in the previous section and this gives question to whether we have overseen something in the VAE framework or if language itself has such different properties, that this statement can not be generalized over all data domains such as images, text and audio.

7.2 Future Work

7.2.1 Dataset

During our work with the proposed dSentences dataset, we have come across many drawback our sentences have. First and foremost, our dataset assumes, that it can be possible to

learn the gender just by observing male versions of the personal pronouns ‘I,you,we,they‘. This is only true for models which can rely on some context such that they can make use of a the posterior distribution. For example, when speaking about the word ‘winds‘ without any context, it is not clear what exactly is meant. The context is missing, as it could mean the conjugated verb ‘to wind‘ but it could also refer to the object ‘wind‘ which has the plural version ‘winds‘. Therefore, learning language can not be boiled down to observing sequences of discrete elements and assuming that there must be a estimable distribution which can describe each and every language.

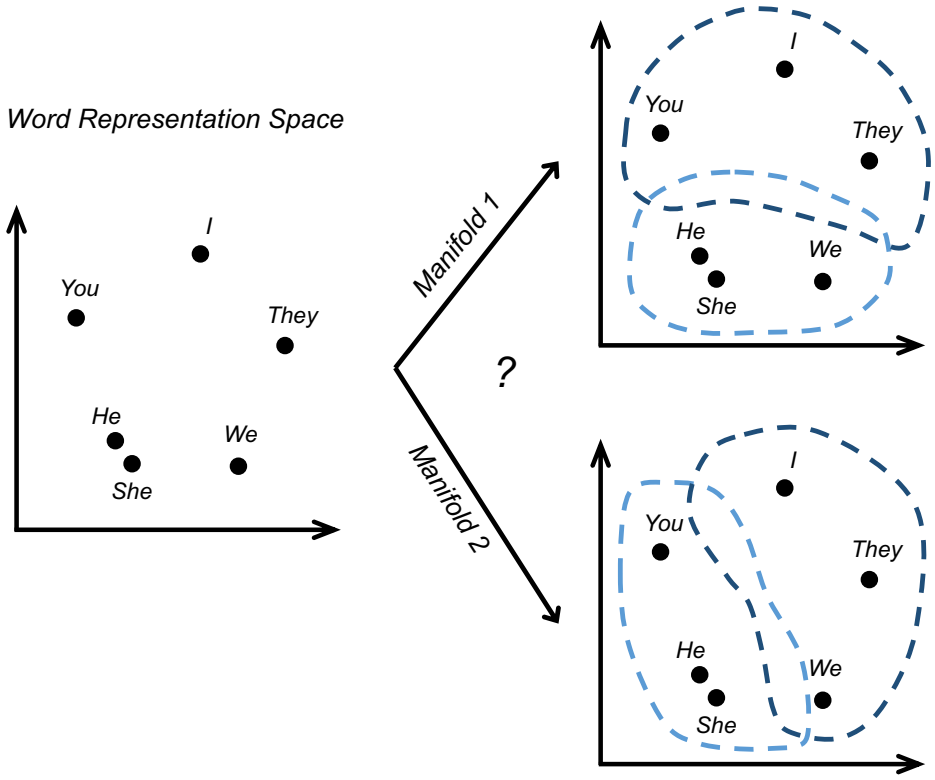


Figure 7.1: Difficult word representation space for manifold learning

7.2.2 Input representation

Having worked with one-hot embeddings and *word2vec* we have learned, that there are certain generative factors, that cannot not be learnt due to the fact, how language as written text is being represented. There are two distinct examples

1. learning the difference between a singular and plural object version

2. distinguishing between the second person singular and plural
3. learning the concept of gender just from text

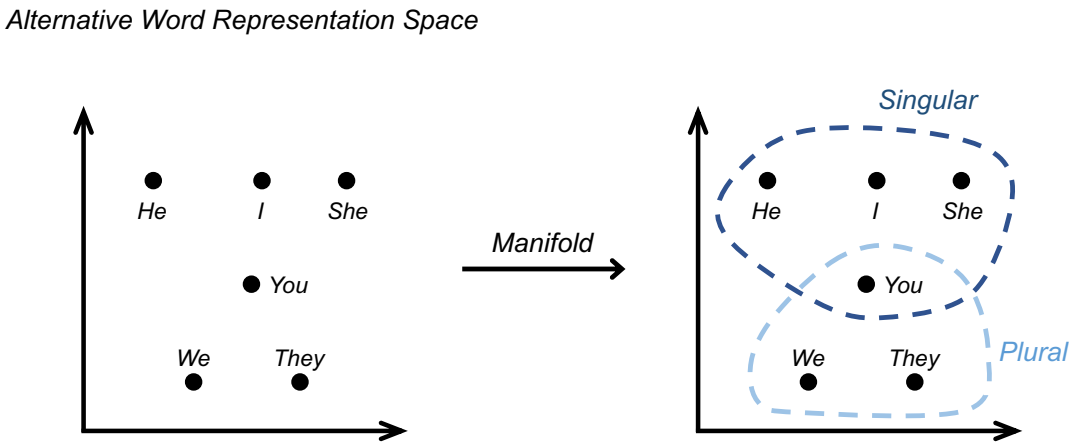


Figure 7.2: Alternative word representation space for manifold learning

All the points are difficult to achieve when using one-hot embedding, as any given word is independent from all other words. Thus there is no connecting relationship the VAE model could learn from. Therfore, we are convinced, that representing and trying to learn a disentangled representation for text with one-hot embeddings is a dead end. The problem of *word2vec* is the fact, that it already introducing an assumption of how the language might be distributed, therefore, when trying to disentangle text, from an standpoint, where our space of possible solutions is already limited is quite constraining. It might be well the case, that by using some predefined embedding in order to apply continuous deep learning techniques on natural language, we might already have introduced some obstacle, which lets us not learn a disentangled representation as shown in Figure7.2. In our opinion, the way to continue research in representation learning is to come up with a totally new approach of representing text. This new approach should while generating such a representation already consider and keep in mind, that a disentangled latent space is a desired property. An idea is shown in Figure as shown in Figure7.2 In order to move forward in terms of natural language understanding.

Bibliography

[1] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[2] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Infovae: Information maximizing variational autoencoders. *CoRR*, abs/1706.02262, 2017.

[3] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *CoRR*, abs/1606.03657, 2016.

[4] Irina Higgins, Loïc Matthey, Xavier Glorot, Arka Pal, Benigno Uria, Charles Blundell, Shakir Mohamed, and Alexander Lerchner. Early visual concept learning with unsupervised deep learning. *CoRR*, abs/1606.05579, 2016.

[5] Tian Qi Chen, Xuechen Li, Roger B. Grosse, and David K. Duvenaud. Isolating sources of disentanglement in variational autoencoders. *CoRR*, abs/1802.04942, 2018.

[6] Christopher P. Burgess, Irina Higgins, Arka Pal, Loïc Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae. *CoRR*, abs/1804.03599, 2018.

[7] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. *CoRR*, abs/1802.05983, 2018.

[8] Emilien Dupont. Joint-vae: Learning disentangled joint continuous and discrete representations. *CoRR*, abs/1804.00104, 2018.

[9] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6402–6413. Curran Associates, Inc., 2017.

[10] Maziar Moradi Fard, Thibaut Thonet, and Eric Gaussier. Deep k-means: Jointly clustering with k-means and learning representations, 2018.

[11] Aditya Devarakonda, Maxim Naumov, and Michael Garland. Adabatch: Adaptive batch sizes for training deep neural networks. *CoRR*, abs/1712.02029, 2017.

[12] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[13] Yann LeCun and Yoshua Bengio. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.

[14] Yoon Kim. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014.

[15] Dinghan Shen, Yizhe Zhang, Ricardo Henao, Qinliang Su, and Lawrence Carin. Deconvolutional latent-variable model for text sequence matching. *CoRR*, abs/1709.07109, 2017.

[16] Tao Lei, Yu Zhang, and Yoav Artzi. Training RNNs as fast as CNNs, 2018.

[17] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

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

[19] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[20] Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Józefowicz, and Samy Bengio. Generating sentences from a continuous space. *CoRR*, abs/1511.06349, 2015.

[21] Kelvin Guu, Tatsunori B. Hashimoto, Yonatan Oren, and Percy Liang. Generating sentences by editing prototypes. *CoRR*, abs/1709.08878, 2017.

[22] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondřej Bojar, Alexandra Constantin, and Evan Herbst. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, ACL '07, pages 177–180, Stroudsburg, PA, USA, 2007. Association for Computational Linguistics.

[23] P. Brown, J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, R. Mercer, and P. Roossin. A statistical approach to language translation. In *Proceedings of the 12th Conference on Computational Linguistics - Volume 1*, COLING '88, pages 71–76, Stroudsburg, PA, USA, 1988. Association for Computational Linguistics.

[24] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan

Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[25] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759, 2016.

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546, 2013.

[27] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[28] Radim Řehůřek and Petr Sojka. Software framework for topic modelling with large corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta, May 2010. ELRA.

[29] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[30] Courtney Napoles, Matthew Gormley, and Benjamin Van Durme. Annotated gigaword. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction*, AKBC-WEKEX '12, pages 95–100, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[31] Mark Davies. The wikipedia corpus: 4.6 million articles, 1.9 billion words, 2015.

[32] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 2672–2680, Cambridge, MA, USA, 2014. MIT Press.

[33] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR*, abs/1206.5538, 2012.

[34] L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[35] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1188–II–1196. JMLR.org, 2014.

[36] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

[37] Tejas D Kulkarni, William F. Whitney, Pushmeet Kohli, and Josh Tenenbaum. Deep convolutional inverse graphics network. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2539–2547. Curran Associates, Inc., 2015.

[38] Andrew D. Bagdanov, Alberto Del Bimbo, and Iacopo Masi. The florence 2d/3d hybrid face dataset. In *Proceedings of the 2011 Joint ACM Workshop on Human Gesture and Behavior Understanding*, J-HGBU ?11, page 79?80, New York, NY, USA, 2011. ACM.

[39] Scott E Reed, Yi Zhang, Yuting Zhang, and Honglak Lee. Deep visual analogy-making. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 1252–1260. Curran Associates, Inc., 2015.

[40] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[41] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[42] Matthew D Hoffman and Matthew J Johnson. Elbo surgery: yet another way to carve up the variational evidence lower bound. In *Workshop in Advances in Approximate Bayesian Inference, NIPS*, 2016.

[43] Alexander A. Alemi, Ben Poole, Ian Fischer, Joshua V. Dillon, Rif A. Saurous, and Kevin Murphy. Fixing a broken ELBO. In *ICML*, volume 80 of *JMLR Workshop and Conference Proceedings*, pages 159–168. JMLR.org, 2018.

[44] Satosi Watanabe. Information theoretical analysis of multivariate correlation. *IBM J. Res. Dev.*, 4(1):66–82, January 1960.

[45] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *CoRR*, abs/1611.01144, 2016.

[46] Chris J. Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *CoRR*, abs/1611.00712, 2016.

[47] Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, and Taylor Berg-Kirkpatrick. Improved variational autoencoders for text modeling using dilated convolutions. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 3881–3890. PMLR, 2017.

[48] Jason Tyler Rolfe. Discrete variational autoencoders. *CoRR*, abs/1609.02200, 2016.

[49] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. *CoRR*, abs/1711.00937, 2017.

[50] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017.

[51] Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. Style transfer from non-parallel text by cross-alignment. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6830–6841. Curran Associates, Inc., 2017.

[52] Mandy Guo, Qinlan Shen, Yinfei Yang, Heming Ge, Daniel Cer, Gustavo Hernández Ábrego, Keith Stevens, Noah Constant, Yun-Hsuan Sung, Brian Strope, and Ray Kurzweil. Effective parallel corpus mining using bilingual sentence embeddings. *CoRR*, abs/1807.11906, 2018.

[53] Igor Melnyk, Cícero Nogueira dos Santos, Kahini Wadhawan, Inkit Padhi, and Abhishek Kumar. Improved neural text attribute transfer with non-parallel data. *CoRR*, abs/1711.09395, 2017.

[54] Jonas Mueller, David K. Gifford, and Tommi S. Jaakkola. Sequence to better sequence: Continuous revision of combinatorial structures. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pages 2536–2544. PMLR, 2017.

[55] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. Toward controlled generation of text. In *International Conference on Machine Learning*, pages 1587–1596, 2017.

[56] Maxine Eskénazi, Kyusong Lee, and Tiancheng Zhao. Unsupervised discrete sentence representation learning for interpretable neural dialog generation. In *ACL (1)*, pages 1098–1107. Association for Computational Linguistics, 2018.

[57] Xi Chen, Diederik P. Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational lossy autoencoder. *CoRR*, abs/1611.02731, 2016.

[58] Kelvin Guu, Tatsunori B. Hashimoto, Yonatan Oren, and Percy Liang. Generating sentences by editing prototypes. *CoRR*, abs/1709.08878, 2017.

[59] Alec Radford, Rafal Józefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *CoRR*, abs/1704.01444, 2017.

[60] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

[61] Cian Eastwood and Christopher K. I. Williams. A framework for the quantitative evaluation of disentangled representations. In *International Conference on Learning Representations*, 2018.

[62] Abhishek Kumar, Prasanna Sattigeri, and Avinash Balakrishnan. Variational inference of disentangled latent concepts from unlabeled observations. *CoRR*, abs/1711.00848, 2017.

[63] J. Schmidhuber. Learning factorial codes by predictability minimization. Technical Report CU-CS-565-91, Dept. of Comp. Sci., University of Colorado at Boulder, December 1991.

[64] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[65] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642. Association for Computational Linguistics, 2013.

[66] Thorsten Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. In *Proceedings of the Fourteenth International Conference on Machine Learning*, ICML '97, pages 143–151, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[67] Edward Loper and Steven Bird. Nltk: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics - Volume 1*, ETMTNLP '02, pages 63–70, Stroudsburg, PA, USA, 2002. Association for Computational Linguistics.

[68] Tom De Smedt and Walter Daelemans. Pattern for python. *J. Mach. Learn. Res.*, 13(1):2063–2067, June 2012.

[69] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[70] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.*, 11:3371–3408, December 2010.

[71] Marc'aurelio Ranzato, Y lan Boureau, and Yann L. Cun. Sparse feature learning for deep belief networks. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1185–1192. Curran Associates, Inc., 2008.

[72] Carl Doersch. Tutorial on variational autoencoders, 2016. cite arxiv:1606.05908.