Understanding Variational Autoencoders and Disentanglement Metrics

Kruthika Hassan

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Master of Science

University of Washington

2022

Reading Committee:

J. Nathan Kutz, Chair

Bamdad Hosseini

Program Authorized to Offer Degree:
Applied Mathematics

University of Washington

Abstract

Understanding Variational Autoencoders and Disentanglement Metrics

Kruthika Hassan

Chair of the Supervisory Committee:
Professor J. Nathan Kutz
Applied Mathematics

In this thesis, we conduct a thorough study of *Variational Autoencoders*. We explain the limitations of "supervised learning" and emphasize the need for "generative models" to solve complex problems. Variational Autoencoder(VAE) is an effective tool for generative modeling. We understand the rich mathematical basis of VAEs, the evidence lower bound(ELBO) and how regularization of original VAEs brings the trade-off between reconstruction fidelity and quality of disentanglement within the learnt representations. We validate the theory by conducting experiments on *Frey's Faces* dataset.

In addition to this, we also learn what constitutes a "good representation" and how *disentanglement metrics* helps in comparing representations obtained from two different models. We briefly describe two different types of disentanglement metrics $\beta - VAE$ metric and *Mutual Information Gap*(MIG).

TABLE OF CONTENTS

Page

Chapter 1: Preliminaries 1
1.1 Statistical Inference 1
1.2 Latent Variable Models 2
1.3 Multi Layer Perceptron 3
1.4 Information Theory 7

Chapter 2: Introduction 9
2.1 Limitations of Supervised Learning 10
2.2 The Problem of Generative Modeling 10
2.3 Generative Models 11

Chapter 3: Variational Autoencoders 13
3.1 Variational Inference 14
3.2 Mathematical Basis of VAEs 15
3.3 β-VAE Framework 22

Chapter 4: Dataset and Architecture 23
4.1 Dataset 23
4.2 Architecture 24

Chapter 5: Experiments 30
5.1 Experimental Observations 30

Chapter 6: Disentanglement Metrics and Conclusions 39
6.1 Disentanglement Metrics 39
6.2 Conclusions 41

Bibliography 43

Appendix A: 46
A.1 KL Divergence between two multi-variate Gaussians 46
A.2 Proof that Monte Carlo estimate of expectation of $Q(z|X)$ w.r.t variational parameters ϕ is differentiable 47

Chapter 1

PRELIMINARIES

1.1 Statistical Inference

There are two main philosophical approaches to statistics - *frequentist or classical* approach and *Bayesian* approach. The *frequentist* approach assumes that the numerical parameters defining a data population is of unknown fixed value. Random samples are drawn from this population and the sample statistic is calculated. The statistic has a probability distribution defined over all possible random samples. The average value of the statistic and the parameter of the population are asymptotically equal to each other. The *Bayesian* method assumes the parameter describing the population is not of fixed value, rather a random variable. There are two aspects to *Bayesian* modeling - ***prior*** and ***posterior***. *Prior* characterizes the beliefs of the observer, even before data is seen. For example, observer 'A' might believe the parameter follows a Gaussian distribution, while observer 'B' believes it to be more of a Uniform distribution, even before seeing the data. The *posterior* refers to updated beliefs.

Inferring about the data distribution using the method of Bayes by computing the posterior is called *Bayesian Inference*. To elaborate on this, let dataset $\mathcal{D} = \{x_1, x_2, x_3, \ldots, x_n\}$, and parameters be θ. Then we can write Bayes rule:

$$\mathcal{P}(\theta|\mathcal{D}) = \frac{\mathcal{P}(\mathcal{D}|\theta)\mathcal{P}(\theta)}{\mathcal{P}(\mathcal{D})}$$

where

$$\mathcal{P}(\mathcal{D}|\theta) \approx \text{Likelihood function of } \theta$$
$$\mathcal{P}(\theta) \approx \text{Prior probability of } \theta$$
$$\mathcal{P}(\theta|\mathcal{D}) \approx \text{Posterior distribution over } \theta$$
$$\mathcal{P}(\mathcal{D}) = \int \mathcal{P}(\mathcal{D}, \theta) d\theta$$

Bayesian Inference is the key aspect to *latent variable modeling*; described in the next section.

1.2 Latent Variable Models

The data we come across in the real world is mostly complex. Consider a simple image of size 28×28. The number of pixels (smallest image unit) is 784. These pixels can be assumed to be points in a real space of dimension 784. That is if each pixel $x \in \mathcal{X}$, then $\mathcal{X} = \Re^{784}$. Now, as the size of image increases, the dimension of $\mathcal{X}$ also increases. Here, the assumption of latent variable models become useful.

Latent variable modeling assumes that the observed high-dimensional data is generated from an underlying low dimensional process. That is, assuming there exists latent/hidden variables z which govern the observed variables x.

Let $\mathcal{X} \subseteq \Re^D$ be the D-dimensional data or observed space. Consider an unknown distribution $p(x)$ in the data space, for $x \in \mathcal{X}$, of which we only see a sample $\{x_n\}_{n=1}^N \subset \mathcal{X}$. In latent variable modeling we assume $p(x)$ is actually due to a small number $L < D$ of latent variables z acting in combination. This L-dimensional space is referred to as the latent space $\mathcal{Z} \subseteq \Re^L$.

A point z in latent space $\mathcal{Z}$ is generated according to distribution $p(z)$ and it is mapped onto data space $\mathcal{X}$ by a smooth, nonsingular mapping $f : \mathcal{Z} \to \mathcal{X}$. The joint probability distribution is $p(z, x)$ where $z \approx p(z)$, $x = f(z)$. Marginalizing this distribution over the latent space gives the distribution in data space.

$$p(x) = \int_{\mathcal{Z}} p(z, x) \; dz = \int_{\mathcal{Z}} p(x|z) \; p(z) \; dz \tag{1.1}$$

The mapping f is absorbed into the likelihood $p(x|z)$, making the dependence of x on z explicit.

If the latent variables are to be efficient in representing faithfully the observed variables, we should expect that, given a value for the latent variables, the values of any group of observed variables are independent of the values of any other group of observed variables.

Otherwise, the chosen latent variables would not completely explain the correlations between the observed variables and further latent variables would be necessary. Thus, for all d, e, $\in \{1, \ldots, D\}$,

$$p(x_d|x_e, z) = p(x_d|z) \implies p(x_d, x_e|z) = p(x_d|x_e, z)p(x_e|z) = p(x_d|z)p(x_e|z)$$

which gives the distribution of observed variables conditioned on the latent variables.

$$p(x|z) \equiv \prod_{d=1}^{D} p_d(x_d|z) \tag{1.2}$$

That is, for some $L \leq D$, the observed variables are conditionally independent given the latent variables. This is usually called the axiom of local independence, and it is a means to fully explain the joint distribution of observed variables in terms of the latent ones. The aim is to find the smallest number of latent variables $L \leq D$ for which it holds. However, in practice one needs to select several values for L and select the best one.

1.3 Multi Layer Perceptron

The multi layer perceptron (MLP) is made up of small units called neurons; loosely modeled on neuron of the brain. Consisting of one or more hidden layers sandwiched between input and output nodes, the neurons exhibit a dense connection between the nodes at different layers, forming a highly connected network. Each node is modeled as a differentiable nonlinear activation function and has associated learnable weights (synaptic weights).

A popular and efficient method to train the MLP is backpropagation. Backpropagation works in two phases - forward and backward. In the forward phase, the synaptic weights are fixed and the the input signal is propagated through the network, layer by layer, until it reaches the output. In the backward phase, an error signal is produced by comparing the network output with the desired response. This error signal is propagated in the backward direction and adjustments are made to the synaptic weights of the network. This is the learning process of the network. Figure 5.1 shows architectural graph of MLP with two hidden layers [Haykin et al., 2009].

The hidden neurons act as feature detectors. Layer by layer they learn the structure in training data and discover its salient features. The input is mapped into a high dimensional space such that the patterns become separable. The MLP can be trained in batches, where the entire training dataset of N samples is presented at a time or in the case of online learning the network learns when training data is presented one sample at a time.

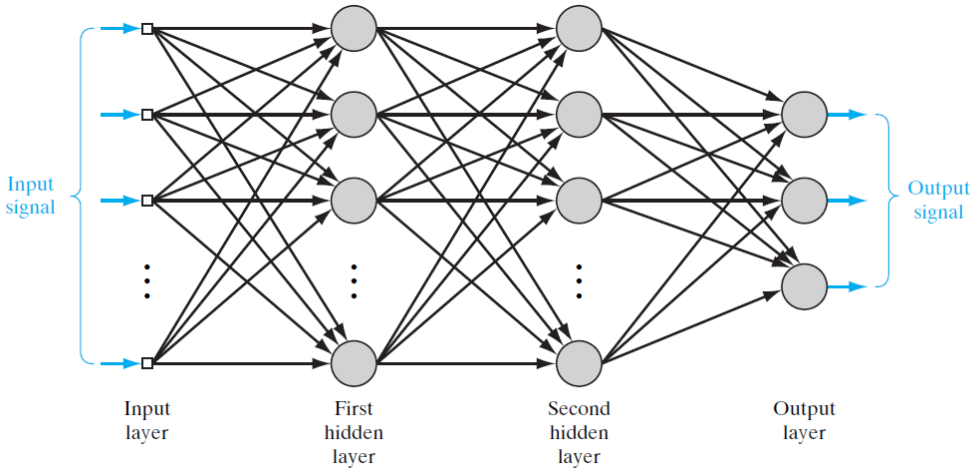

Figure 1.1: Architectural graph of a multilayer perceptron with two hidden layers

1.3.1 Convolutional Networks

The architectural layout of an MLP can be modified to address different kinds of problems. One such architecture is that of a convolutional network which was designed specifically to recognize 2D shapes with a high degree of invariance to translation, skewing, and other forms of distortion. The idea motivated by the locally sensitive and orientation selective neurons of the visual cortex of a cat.

One of the first convolutional architectures LeNet-5 [LeCun et al., 1998], which found wide practical applications in handwritten digit recognition is shown below:

The LeNet-5 convolutional network combines three architectural ideas to ensure some

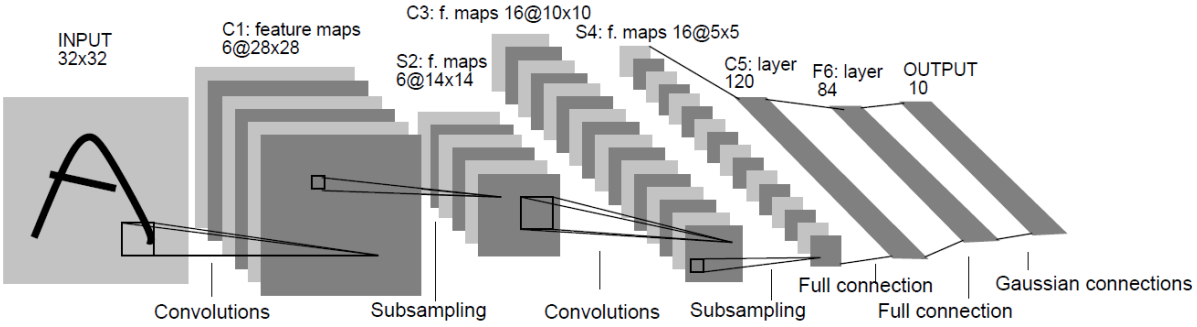

Figure 1.2: Architecture of LeNet-5 [LeCun et al., 1998]

degree of shift, scale and distortion invariance: *local receptive fields*, *shared weights*, *spatial or temporal subsampling*.

- *Convolutional Layer:* The input plane receives images of characters that are approximately size-normalized and centered. Each unit in a layer receives inputs from a set of units located in a small neighborhood in the previous layer. These local receptive fields help in extracting elementary visual features such as oriented edges, endpoints, corners. These features are then combined by the subsequent layers in order to detect higher-order features.

  The elementary feature detectors that are useful in one part of the image are likely to be useful across the entire image. This knowledge can be applied by forcing a set of units, whose receptive fields are located at different places on the image, to have identical weight vectors. Units in a single layer are organized in **planes** within which all the units share the same set of weights. The set of outputs of the units in such a plane is called a feature map. Units in a feature map are all constrained to perform the same operation on different parts of the image. A complete convolutional layer is composed of several feature maps (with different weight vectors), so that multiple features can be extracted at a given location. In figure 1.2, the first layer comprises of 6 feature maps.

A unit in a feature map has 25 inputs connected to 5 by 5 area in the input, called the *receptive field* of the unit. Each unit has 25 inputs, and therefore 25 trainable coefficients plus a trainable bias. The receptive fields of contiguous units in a feature map are centered on correspondingly contiguous units in the previous layer. Therefore receptive fields of neighboring units overlap. In the first convolutional layer 'C1' of figure 1.2, the receptive fields of horizontally contiguous units overlap by 4 columns and 5 rows. All the units in a feature map share the same set of 25 weights and the same bias and hence detect the same feature at all possible locations on the input. Different feature maps use different weights and biases, extracting different types of local features. In figure 1.2, 'C1' has 6 different feature maps and 6 different features are extracted by 6 units in identical locations in the six feature maps. A sequential implementation of a feature map would scan the input image with a single unit that has a local receptive field and stores the states of this unit at corresponding locations in the feature map. This operation is equivalent to a convolution followed by an additive bias and squashing function, hence the name convolutional network. The kernel of the convolution is the set of connection weights used by the units in the feature map. An interesting property of convolutional layers is that, if the input image is shifted the feature map output will be shifted by the same amount, but will be left unchanged otherwise. This property is at the basis of the robustness of convolutional networks to shifts and distortions of the input [LeCun et al., 1998].

- *Sub-sampling Layer:* Once a feature has been detected, its exact location becomes less important. Only its approximate position relative to other features is relevant. Not only is the precise position of each of those features irrelevant for identifying the pattern, it is potentially harmful because the positions are likely to vary for different instances of the character. A simple way to reduce the precision with which the position of distinctive features are encoded in a feature map is to reduce the spatial resolution of the feature map. This can be achieved with a *sub-sampling layers* which performs

a local averaging and sub-sampling reducing the resolution of the feature map and reducing the sensitivity of the output to shifts and distortions. In figure 1.2, the second layer has 6 feature maps, one for each feature map in the previous layer. The receptive field of each unit is a 2 by 2 area in the previous layer's corresponding feature map. Each unit computes the average of its four inputs multiplies it by a trainable coefficient, adds a trainable bias and passes the result through a sigmoid function. Contiguous units have non-overlapping contiguous receptive fields. Consequently a sub-sampling layer feature map has half the number of rows and columns as the feature maps in the previous layer [LeCun et al., 1998].

In figure 1.2 there is successive layers of convolutions and sub-sampling are typically alternated resulting in a "bi-pyramid": at each layer the number of feature maps is increased as the spatial resolution is decreased.

1.4 Information Theory

A few basic terms and definitions that will help us understand the key concepts to be elucidated in the following chapters. If X and Y are two continuous random variables, $\mathcal{X}$ and $\mathcal{Y}$ with probability density functions (pdf) $p(x)$ and $p(y)$.

Differential Entropy: The differential entropy of a continuous random variable X with pdf $p(x)$ is

$$h(X) = -\int p(x) \log p(x) dx = -\mathbb{E}[\log p(x)]$$

Joint Entropy: Let X and Y be distributed jointly according to the pdf $p(x, y)$. Then the joint entropy is

$$h(X, Y) = -\int \int p(x, y) \log p(x, y) dx\, dy$$

Conditional Entropy: The conditional entropy $h(X|Y)$ is defined as

$$h(X, Y) = -\int \int p(x, y) \log p(x|y) dx\, dy$$

Relative Entropy: Given two probability density functions $p(x)$ and $q(x)$, the *relative entropy* or *Kullback-Leibler distance* is a measure of the distance between them and is defined as

$$D(p||q) = \int p(x) \log \frac{p(x)}{q(x)} \, dx$$

Mutual Information: The *mutual information* $I(X;Y)$ is the relative entropy between the joint distribution and product distribution $p(x)p(y)$, i.e.

$$\begin{aligned} I(X;Y) &= D(p(x,y)||p(x)p(y)) \\ &= \int \int p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \, dx \, dy \end{aligned}$$

Relation between Entropy and Mutual Information: The relation is derived as follows

$$\begin{aligned} I(X;Y) &= \int \int p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \, dx \, dy \\ &= \int \int p(x,y) \log \frac{p(x|y)}{p(x)} \, dx \, dy \\ &= \int \int p(x,y) (\log p(x|y) - \log p(x)) \, dx \, dy \\ &= -\int \int p(x,y) \log p(x) \, dx \, dy + \int \int p(x,y) \log p(x|y) \, dx \, dy \\ &= -\int p(x) \log p(x) \, dx - \Big(-\int \int p(x,y) \log p(x|y) \, dx \, dy \Big) \\ &= h(X) - h(X|Y) \end{aligned}$$

By symmetry, $I(X;Y) = h(Y) - h(Y|X)$

Chapter 2

INTRODUCTION

Humans have been a largely successful species due to power of abstraction. *Abstraction* and *Imagination* are two key elements that endow an immense advantage in solving complex problems. The ability to be creative is a crucial aspect in problem-solving. A few applications which motivate creative problem solving are listed below.

Consider the task of designing proteins. Proteins are the fundamental units of living organisms that initiate key processes such as break down of starch from food into forms that can be used by body for different purposes. Made up of long chain of amino acids, proteins fold up into compact blob, keeping some pairs of amino acids near center of the blob and others outside. Every protein folds up into a specific shape, the same shape every time. The possibilities are astronomical simply due to the number of degrees of freedom. Beating nature at it's creative best is a formidable task, and hence predicting the shape a protein folds into is one of the hardest problems in biology today.

Data compression is at the heart of modern technology. Digital revolution owes a big chunk to data compression methods. New techniques in image, speech, text, audio and video compression is becoming vital due to growing data volume in each of these arena. Can we build a compression scheme that is adaptive to the kind of data used as input.

Images are 2D representation of scenes. Including a time dimension would become a video. Given 2D images it may be imperative to reconstruct 3D model. Given a scene, creating a 3D representation has numerous applications in visual arts, video games, etc. Generation of images, text, sound, finds relevance in real world. A suitable paradigm to solve such complex problems is needed.

2.1 Limitations of Supervised Learning

Supervised learning or learning with a teacher has a set of labeled data given to the system and the system learns by minimizing the error between desired and obtained response. The problems of classification/regression has been efficiently solved by this method. In the supervised setting, finding a discriminant function $f : X \rightarrow Y$ that maps the input data X to corresponding class label Y_k, $k \in 1, 2, 3, \ldots$ is sufficient to successfully complete the task. Modeling the input and output data distribution is not imperative. Generative modeling explicitly addresses this.

2.2 The Problem of Generative Modeling

In computer graphics and computer aided design, complex shapes are generated by the continuous transformation of a generator. For example, a cylinder is generated by a circle in the xy-plane, translated along the z-axis. In general, a lower dimensional generator is transformed to produce a higher dimensional shape. Consider a curve $\gamma(u) : \Re^1 \rightarrow \Re^3$, a parametric transformation $\delta(p, v) : \Re^3 \times \Re$ that acts on points in $p \in \Re^3$ given parameter v. A surface, $S(u, v)$ may be generated by using the transformation $\delta(p, v)$:

$$S(u, v) = \delta(\gamma(u), v)$$

As another example consider a color image. Pixels form the fundamental elements of an image and let us denote them by the N random vectors $X_1, X_2, X_3, \ldots, X_N$. Each vector can take values $\{x, y, color\} \in \Re^+$. Now, defining a joint distribution over all the pixels $P(X_1, \ldots, X_N)$ would be sufficient to generate new image samples.

Also consider the example of speech production. Speech is produced by forcing air through an elastic opening (the vocal cords) and then through cylindrical tubes with nonuniform diameter (the laryngeal, oral, nasal and pharynx passages), and finally through cavities with changing boundaries(the mouth and nasal cavity). Everything past the vocal cords is generally referred to as the vocal tract. The first action generates the sound, which is

then modulated into speech as it traverses through the vocal tract. The vocal tract can be modeled as a series of tubes of varying diameter. A model where the output from each tube depends on the previous one, and also accounts for the stochasticity of the process may help in understanding speech production better.

In all the above cases, we observe that the underlying physical process is a complex one and building a mathematical model is challenging. Be the transformation $\delta(p, v)$, joint probability distribution $P(X_1, \ldots, X_N)$, or the autoregressive process, each provides enough information to help in generation of surfaces, images, sounds respectively. All of them are known as generative models.

2.3 Generative Models

The set of all approaches which model the input and output distribution explicitly or implicitly are known as generative models. To elaborate on this, consider examples of continuous and discrete data. In both cases let there be two types of distributions(or classes) hidden, $\mathcal{C}_1, \mathcal{C}_2$. A simple discriminative algorithm such as perceptron or logistic regression would try to find the best linear discriminant iteratively (assuming the data is linearly separable). The discriminative algorithms only aim at successfully separating the different classes by finding the posterior class probabilites $P_k(\mathcal{C}_k|x)$. In case of a generative algorithm, both the class priors $P(\mathcal{C}_k)$ and class conditional densities $P(x|\mathcal{C}_k)$ are obtained separately. Posterior class probabilities is found using the Bayes' theorem [Bishop, 2006].

$$P(\mathcal{C}_k|x) = \frac{P(x|\mathcal{C}_k)P(\mathcal{C}_k)}{P(x)} \tag{2.1}$$

$$P(x) = \sum_k P(x|\mathcal{C}_k)$$

As an example, for the continuous case, $x \in \Re^n$, Gaussian discriminant analysis models class prior as a Bernoulli distribution and conditional densities as a Gaussian distribution. In the discrete case, $x_i \in \{0, 1\}(i = 1, \ldots, n)$, the Naive Bayes makes a strong assumption

that, conditioned on class $\mathcal{C}_k$, x_i's are independent. Applying the definition of conditional probability recursively we get,

$$
\begin{aligned}
& P(x_1, x_2, \ldots, x_n|\mathcal{C}_k) \\
&= P(x_1|\mathcal{C}_k)P(x_2|x_1, \mathcal{C}_k) \ldots P(x_n|x_1, x_2, \ldots, x_{n-1}, \mathcal{C}_k) \\
&= P(x_1|\mathcal{C}_k)P(x_2|\mathcal{C}_k), \ldots, P(x_n|\mathcal{C}_k) \\
&= \prod_{i=1}^{n} P(x_i|\mathcal{C}_k)
\end{aligned}
$$

In the equation (2.1), there are three parts - likelihood $P(x|\mathcal{C}_k)$, prior $P(\mathcal{C}_k)$ and marginal probability $P(x)$. There are two generative models widely used - **Variational Autoencoders**(VAEs) and **General Adversarial Networks**(GANs). In this thesis we are mainly concerned with VAEs.

Chapter 3

VARIATIONAL AUTOENCODERS

The concept of autoencoders was originally proposed by LeCun [Lecun, 1987] in his PhD thesis. *Autoencoders* consist of three key parts - source, an *encoder* and a *decoder*; a communication system sans a channel. The encoder, decoder are generally implemented using a neural network. A set of *recognition weights* in the encoder convert the input space(source) vector into a code vector; and *generative weights* in the decoder reconstructs an approximation to the input vector [Hinton and Zemel, 1994]. A schematic of an autoencoder is shown in 3.1.

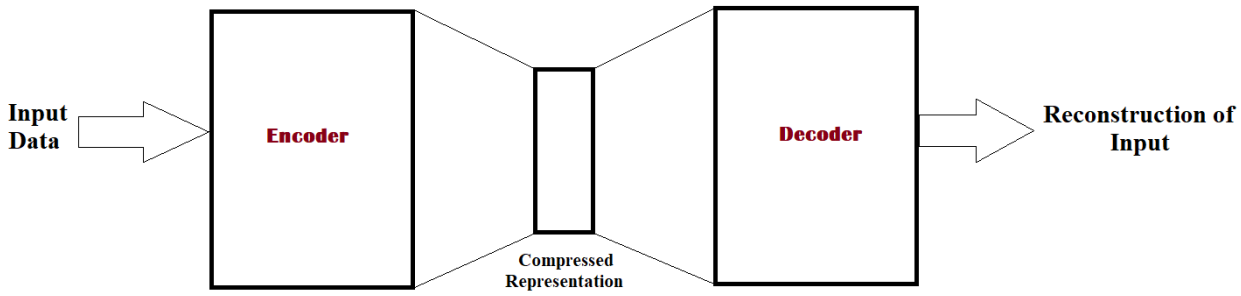

Figure 3.1: Schematic of an Autoencoder

Traditionally, autoencoders were used for dimensionality reduction or feature extraction [Hinton and Salakhutdinov, 2006]. While PCA (principal components analysis) corresponds to a linear method of dimensionality reduction, an autoencoder is it's non-linear counterpart. However, with the popularity of various deep learning models, autoencoders have been brought to the forefront of generative modeling. Many extended autoencoder models have

been proposed by different researchers. The extended autoencoder models can be roughly classified into three categories [Zhai et al., 2018]:

- **Models based on instantiation of encoder and decoder functions**: An example is the Convolutional neural networks (CNN) based autoencoder, where encoder and decoder functions are instantiated with a CNN [Zhai et al., 2018].

- **Models based on regularization technique**: An example is the sparse autoencoder where sparsity constraints are imposed on the encoded variable space/latent variable space. The idea is to constrain the number of active hidden units at a given point of time [Zhai et al., 2018].

- **Models based on variational inference**: These refer to the variational autoencoders [Zhai et al., 2018].

We are primarily interested in the *Variational Inference* based autoencoders.

3.1 Variational Inference

Latent variable models are an important resource for solving complex problems. With latent variable models, we posit a rich latent structure that governs our observations, infer that structure from large data sets, and use our inferences to summarize observations, draw conclusions about current data and make predictions about new data. In the latent variable modeling, a point z in latent space $\mathcal{Z}$ is generated according to a distribution $p(z)$ and it is mapped onto data space $\mathcal{X}$ by a smooth, nonsingular mapping $f : \mathcal{Z} \rightarrow \mathcal{X}$. The joint probability distribution is $p(z, x)$ where $z \approx p(z)$, $x = f(z)$. Marginalizing this distribution over the latent space gives the distribution in data space.

$$p(x) = \int_{\mathcal{Z}} p(z, x) \ dz = \int_{\mathcal{Z}} p(x|z) \ p(z) \ dz \tag{3.1}$$

For many interesting models, computing the posterior exactly is intractable: practitioners must resort to approximation methods such as *Variational Inference*.

Variational inference tries to find a member from the family of simple probability distributions which is in close proximity (quantified through KL-divergence) to the true posterior distribution [Wainwright and Jordan, 2008a], [Wainwright and Jordan, 2008b]. Let us assume that $y(x)$ is the true posterior. Then defining $Y(x)$ as the family of comparison functions

$$Y(x) = (1 - \epsilon)\, y(x) + \epsilon\, \eta(x) \tag{3.2}$$

where $\eta(x)$ is an arbitrary differentiable function and ϵ is a variational parameter such that when $\epsilon = 0$, we have $Y(x) = y(x)$. The estimate becomes equals to the true posterior. For a given $\eta(x)$, each value of ϵ designates a single member of the family $Y(x)$ [Weinstock, 1974].

The variational inference transforms the problem of approximating a conditional distribution into an optimization problem [Bishop, 2006]. The idea is to posit a simple family of distributions over the latent variables and find the member of the family that is closest in KL-divergence to the conditional distribution.

In a probabilistic model, let x be observations, z be latent variables, and λ be the free parameters of variational distribution $q(z|\lambda)$. The goal is to approximate $p(z|x)$ with a setting of λ. In our approach, λ corresponds to the variational parameters ϕ and θ; weights learnt by the encoder and decoder network respectively.

3.2 Mathematical Basis of VAEs

The mathematical basis of variational autoencoders has relatively little to do with classical autoencoders such as the sparse autoencoders. Using "autoencoders" along with "variational" only refers to the architecture having an encoder and a decoder, similar to the traditional autoencoder. Unlike sparse autoencoders, there are generally no tuning parameters analogous to the sparsity penalties. And unlike sparse and denoising autoencoders, we can sample directly from $P(X)$ (without performing Markov Chain Monte Carlo) [Doersch, 2016].

The core equation of latent variable models is:

$$P(X) = \int P(X|z;\theta)\, P(z)\, dz \tag{3.3}$$

To solve (3.3), there are two questions that need to be answered. How to choose the latent variables z such that they give the best representation of input data X. Also, how to solve the integral in (3.3) in closed form?

To answer the first question, consider the simple example of handwritten digits sampled from the MNIST dataset (`http://yann.lecun.com/exdb/mnist/`:

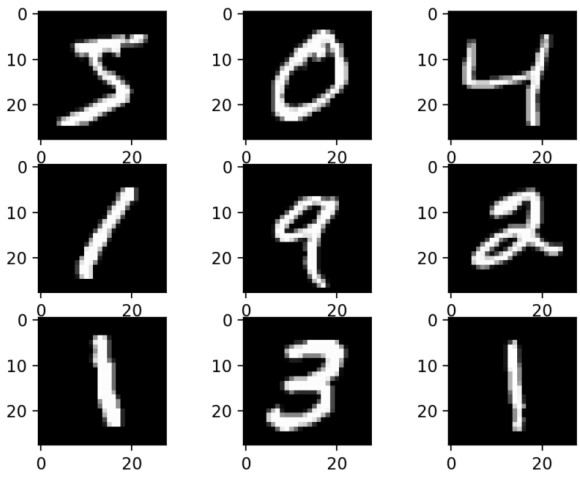

Figure 3.2: Sample of MNIST dataset

The digits have different representations and even among the same set of digits the strokes are varied based on the handwriting. Learning a few reference vectors denoting the angle of stroke, it's width, etc, which may be correlated to generate a digit is a rather complicated problem. Ideally, we want to avoid deciding by hand what information each dimension of z encodes. We also want to avoid explicitly describing the dependencies, i.e., the latent structure between the dimensions of z.

3.2.1 ***Learning Strategy***

Since there is no simple interpretation of dimensions of z, the *variational autoencoder* assumes that z can be sampled from a standard normal distribution $\mathcal{N}(0, I)$. The basis for this assumption is that, any distribution in d dimensions can be generated by taking a set of d 1-D normally distributed random variables and mapping them through a sufficiently complicated function. To see this, consider z to be sampled from the 2-D isotropic Gaussian, $\mathcal{N}(0, I)$. Let this be transformed such that $X = g(z)$, where $g(.)$ is a deterministic function. Various examples are shown in Figure 3.3.

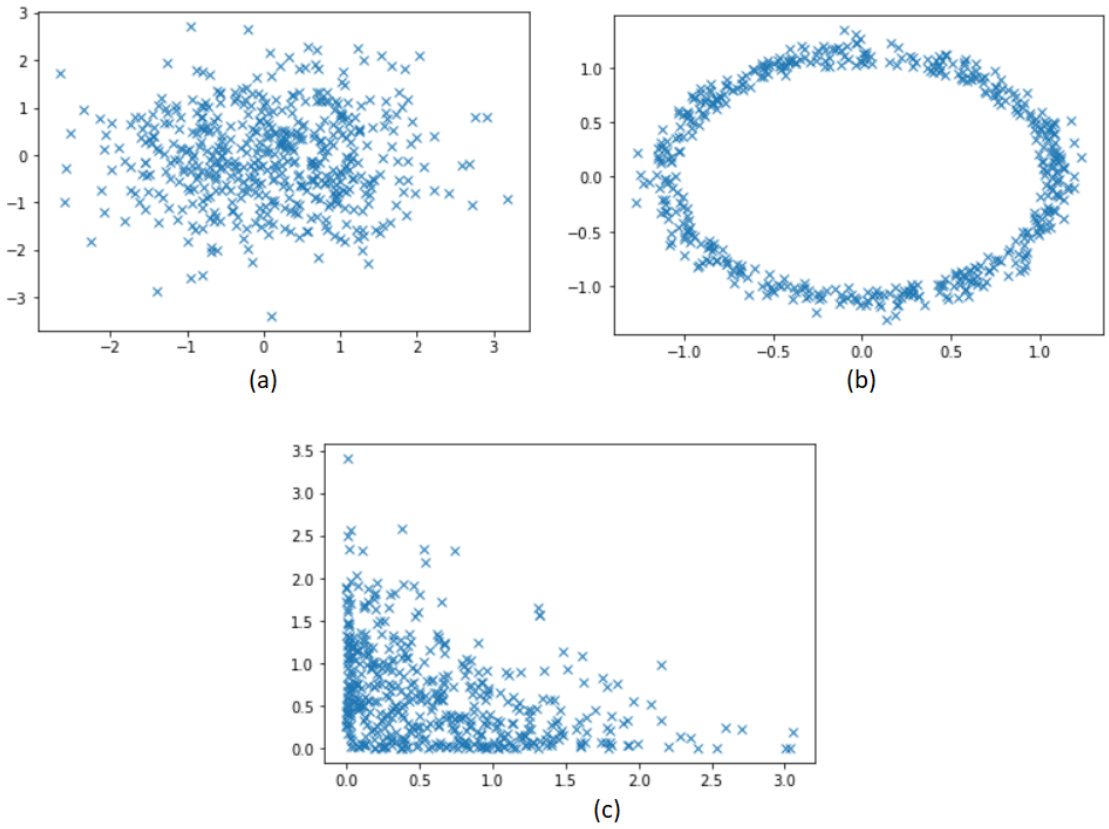

Figure 3.3: (a) $z \sim \mathcal{N}(0, I)$, (b) $g(z) = \frac{z}{10} + \frac{z}{\|z\|}$, (c) $g(z) = \frac{z^2}{\|z\|}$

Multi-layer neural networks can be used as powerful universal function approximators [Hornik et al., 1989] and can be used to learn complex functions such as $g(\cdot)$. Now, let reconstructed data assume distribution of the form $P(X|g(z;\theta)) \approx \mathcal{N}(X|g(z;\theta), \sigma^2 * I)$ with $g(z;\theta)$ being a multi-layer neural network and θ being the variational parameters. This decoder network, $g(z;\theta)$ uses its first few layers to map the normally distributed z's to the latent values. In the case of handwritten digits, the latent values may correspond to stroke weight, angle, width, etc.

In the right hand side of (3.3), reconstructor/decoder $P(X|z;\theta)$ and prior $P(z)$ have been simplified as described above. Yet, the integral cannot be calculated in closed form. $P(X)$ can be approximated using the Monte Carlo estimate for the integral i.e $P(X) = \frac{1}{n}\sum_i P(X|z_i)$, by sampling a large number of z values $z_1, z_2, \ldots, z_n$. But it may require infeasible number of samples of z to get a good estimate of $P(X)$. In this method of approximating $P(X)$ using samples from z, for most z, $P(X|z)$ will be nearly zero and hence contribute almost nothing to our estimate of $P(X)$.

To circumvent this problem, we narrow the space of latent variables z. To do this, we need to introduce an additional structure, the encoder. Let $Q(z|f(X;\phi))$ denote such an encoder with $f(X;\phi)$ being a multilayer neural network and ϕ be the variational parameters learnt by the network. Also denote $P(z|X)$ as the true distribution. Thus, the space of z values that are likely under Q, will be much smaller than the space of all z's that are likely under the prior $P(z)$.

3.2.2 ***Constructing the Objective***

To construct the objective, let us first relate $E_{z\sim Q}P(X|z)$ and $P(X)$. The relationship between $E_{z\sim Q}P(X|z)$ and $P(X)$ is one of the cornerstones of *Variational Bayesian methods*. Kullback-Leibler divergence (KL divergence or D between $P(z|X)$ and $Q(z)$, for some arbitrary Q (which may or may not depend on X) is given by:

$$D[Q(z)||P(z|X)] = E_{z\sim Q}[\log Q(z) - \log P(z|X)] \tag{3.4}$$

We can get both $P(X)$ and $P(X|z)$ into this equation by applying Bayes rule to $P(z|X)$:

$$D[Q(z)||P(z|X)] = E_{z\sim Q}[\log Q(z) - \log P(X|z) - \log P(z)] + \log P(X) \tag{3.5}$$

Here, $\log P(X)$ comes out of the expectation because it does not depend on z. Negating both the sides, rearranging and contracting part of $E_{z\sim Q}$ into a KL-divergence term yields:

$$\log P(X) - D[Q(z)||P(z|X)] = E_{z\sim Q}[\log P(X|z)] - D[Q(z)||P(z)]. \tag{3.6}$$

Note that X is fixed, and Q can be any distribution, not just a distribution which does a good job mapping X to the z's that can produce X. Since we are interested in inferring $P(X)$, it makes sense to construct Q which does depend on X, and in particular, one which makes $\mathcal{D}[Q(z|X)||P(z|X)]$ small:

$$\log P(X) - D[Q(z|X)||P(z|X)] = E_{z\sim Q}[\log P(X|z)] - D[Q(z|X)||P(z)] \tag{3.7}$$

$$\log P(X) \geq E_{z\sim Q}[\log P(X|z)] - D[Q(z|X)||P(z)] \tag{3.8}$$

The equation (3.7) is the basis of a *variational autoencoder*, where in the right hand side (RHS), Q "encodes" X into z and P decodes it to reconstruct X. The left hand side (LHS) of equation (3.8) is known as *evidence*. RHS provides lower bound to this evidence (since $D[Q(z|X)||P(z|X)]$ is nonnegative) and is known as the *evidence lower bound* (ELBO).

From (3.8) we observe that there is a gap between *evidence* and *ELBO*, precisely given by KL-divergence between $Q(z|X)$ and $P(z|X)$. $P(z|X)$ cannot be computed analytically; it describes the values of z that are likely to give rise to a sample like X under the decoder-encoder model described above. Assuming an arbitrarily high-capacity model is used for $Q(z|x)$, then $Q(z|x)$ will hopefully actually match $P(z|X)$, in which case this KL-divergence term will be zero, and we will be directly optimizing $\log P(X)$. Intractable $P(z|X)$ is made tractable; we can just use $Q(z|x)$ to compute it. Starting with the left hand side, we are maximizing $\log P(X)$ while simultaneously minimizing $\mathcal{D}[Q(z|X)||P(z|X)]$.

We had mentioned earlier that $P(z)$ assumed to be isotropic Gaussian. In order to obtain a posterior $Q(z|X)$ close to the form we have assumed for $P(z)$, the usual choice

is $Q(z|X) = N(z|\mu(X;\phi), \Sigma(X;\phi))$; where μ and Σ are arbitrary deterministic functions, with Σ constrained to be a diagonal matrix and ϕ being variational parameters that can be learned from data. In our encoder model, μ and Σ are implemented via neural networks. The last term on the RHS of (3.7), $D[Q(z|X)||P(z)]$ is now a KL-divergence between two multivariate Gaussian distributions, which can be computed in closed form as (see Appendix A):

$$D[\mathcal{N}(\mu_0, \Sigma_0)||\mathcal{N}(\mu_1, \Sigma_1)] = \frac{1}{2}\Big(tr\Big(\Sigma_1^{-1}\Sigma_0\Big) + (\mu_1 - \mu_0)^\top \Sigma_1^{-1}(\mu_1 - \mu_0) - k + \log\Big(\frac{det(\Sigma_1)}{det(\Sigma_0)}\Big) \quad (3.9)$$

where k is the dimensionality of the distribution. With prior $P(z)$ as $\mathcal{N}(0, I)$, the above equation reduces to

$$D[\mathcal{N}(\mu, \Sigma)||\mathcal{N}(0, I)] = \frac{1}{2}\Big(tr(\Sigma(X)) + (\mu(X))^\top(\mu(X)) - k - \log det(\Sigma(X))\Big) \quad (3.10)$$

3.2.3 ***Optimizing the Objective***

The term $E_{z\sim Q}[logP(X|z)]$ on the RHS of the ELBO can be estimated using *Monte Carlo* sampling. But getting a good estimate requires passing of many samples of z through $g(\cdot)$, which may be expensive. Using stochastic gradient descent simplifies the process, since one sample of z is considered at a time and then $P(X|z)$ for that sample is treated as approximate to $E_{z\sim Q}[logP(X|z)]$. The same is done to entire dataset with X being sampled from dataset $\mathcal{D}$. The objective to optimize can be rewritten as:

$$E_{X\sim\mathcal{D}}[\log P(X) - D[Q(z|X)||P(z|X)] = E_{X\sim\mathcal{D}}[E_{z\sim Q}[\log P(X|z)] - D[Q(z|X)||P(z)]] \quad (3.11)$$

Impediments to Optimizing the Objective

Now to optimize, gradient of (3.11) needs to be taken. The forward pass of 3.11 works fine and if the output is averaged over many samples of X and z produces the correct expected value. However, we need to back-propagate the error through a layer that samples z from $Q(z|X)$, which is stochastic in nature and hence has no gradient [Doersch, 2016].

Reparametrization Trick

The solution to the problem described above is called the "reparameterization trick" given by [Kingma and Welling, 2013]. An alternative way of generating samples from $Q(z|X)$ can be devised using reparameterization. The essential parameterization trick is quite simple; the continuous random variable z is expressed as a deterministic variable $z = g_\phi(\epsilon, x)$, where ϵ is an auxiliary variable with independent marginal $p(\epsilon)$, and $g_\phi(.)$ is some vector valued function parameterized by ϕ. Here, ϕ are the variational parameters learnt by the encoder. Using this reparameterization, expectation with respect to $Q(z|x)$ such that the Monte Carlo estimate of the expectation is differentiable with respect to ϕ [Kingma and Welling, 2013] (proof in Appendix A). A differentiable estimator can be constructed thus: $\int Q(z|X)f(z)dz = \frac{1}{L}\sum_{l=1}^{L} f(g_\phi(x, \epsilon^l))$ where $\epsilon \sim p(\epsilon)$, and L denotes total number of samples in a batch. We can apply this trick to obtain an estimate for ELBO.

A Differentiable Estimator for ELBO

We have assumed that the variational approximate posterior $Q(z|X)$ takes the form of a multivariate Gaussian with diagonal covariance.

$$\log Q(z|X^{(i)}) = \log \mathcal{N}(z; \mu^{(i)}, \sigma^{2^{(i)}} I). \tag{3.12}$$

where the mean and standard deviation of the approximate posterior, $\mu^{(i)}$ and $\sigma^{(i)}$ are outputs of the encoder i.e. nonlinear functions of datapoint $X^{(i)}$ and the variational parameters ϕ. The posterior is sampled $z^{(i,l)} \sim Q(z|X^{(i)})$ using reparameterization $z^{(i,l)} = \mu^{(i)} + \sigma^{(i)} \bigodot \epsilon^l$ where $\epsilon \sim \mathcal{N}(0, I)$ and $\bigodot$ denotes element-wise product [Kingma and Welling, 2013].

We have obtained a closed form expression for the term $\mathcal{D}[Q(z|X)||P(z)]$ of the ELBO in (3.10). As for the decoder term, we have $E_{z\sim Q}[\log P(X|z)] = \frac{1}{L}\sum_{l=1}^{L} \log P(X^i|z^{(i,l)})$ where $z^{(i,l)}$ is as described above [Kingma and Welling, 2013].

3.3 β-VAE Framework

A variational autoencoder is not always regularized. But $\beta-$VAE, a constrained variational framework proposed by [Higgins et al., 2016] introduces such a regularization by augmenting a parameter β to vanilla VAE [Kingma and Welling, 2013], that is,

$$E_{X\sim\mathcal{D}}[\log P(X)] \geq E_{X\sim\mathcal{D}}[E_{z\sim Q_\phi}[\log P_\theta(X|z)] - \beta\, D[Q_\phi(z|X)||P(z)]] \tag{3.13}$$

where ϕ and θ are variational parameters and $\beta = 1$ corresponds to vanilla VAE. Varying β changes the learning constraints applied to the objective. The first term on the RHS of (3.13) can be interpreted as the negative reconstruction error and the second term as the complexity penalty that acts as a regulariser [Kim and Mnih, 2018]. The second term can be further broken down as [Hoffman and Johnson, 2016], [Makhzani and Frey, 2017]:

$$E_{X\sim\mathcal{D}}[D[Q(z|X)||P(z)]] = \beta\ [I(X;z) + D[Q(z)||P(z)]] \tag{3.14}$$

where $I(X;z)$ is the mutual information between X and z under the joint distribution $Q(z|X)P(X)$. Penalizing $D[Q(z)||P(z)]$ drives $Q(z)$ towards factorized prior $P(z)$ and thus encourages disentangled learning. But there is also a constraint applied on $I(X;z)$, leading to decrease in amount of information captured in the latent factors z about X [Kim and Mnih, 2018]. This creates a trade-off between reconstruction fidelity and the quality of disentanglement within the learnt representations for high values of β. Disentangled representations emerge when the right balance is found between information preservation (reconstruction cost with regularisation) and latent channel capacity restriction.

Chapter 4

DATASET AND ARCHITECTURE

4.1 Dataset

Frey Faces Dataset

The Frey Faces dataset consists of a series of 1965 images of size $28x20$ of Brendan Frey's face taken from sequential frames of a video. The dimension of the dataset is 1965×560, with each row representing a single image with pixel values ranging from $(0 - 255)$ The data when read is in the dictionary format, having a single key representing the color parameter in hex format.

Figure 4.1: Samples from Frey Faces Dataset

4.2 Architecture

The Variational Autoencoder comprises of an "encoder" and a "decoder" as studied in Chapter 3. While there are several architectures which can be chosen for the encoder and decoder, we adopt a LeNet-5 based architecture (without the sub-sampling layers) for our experimental purposes.

4.2.1 ***Encoder***

A schematic for the encoder is represented by figure 4.2 with respect to dataset under consideration. The architecture in detail is explained below:

- Consists of four 2D convolutional layers, a flattening layer and two dense layers.
- The 2D covolutional layers have filters of window 3×3, number of feature maps 32, 64, 64, 64 and activation function as 'ReLu' (Rectified Linear Units). There are no subsampling layers. The planar feature maps learn the local representation of the input image. Since there is no subsampling and padding, the original size of the input image is preserved.
- The output of the convolutional layers is then flattened to obtained a 1-D vector.
- The flattened vector is then given as an input to Dense layer (fully connected) having 32 units and activation function as 'ReLu'.
- The encoder learns the parameters of the latent distribution $Q(z|X)$.. The prior $P(z)$ on the latent distribution is assumed to be a 2D Gaussian with isotropic variance, and hence we have two parameters *mean* and *variance* to be inferred from the neural network.
- The output of dense layer given to another Dense layer consisting of only 2 units. To generalize, it consists of those many units as that of the latent code dimension.

Decoder

The decoder network maps the latent space into images, $Q(z) \rightarrow P(X|z)$. A schematic for the decoder is represented by figure 4.3. The architecture in detail is explained below:

- The network consists of an up-sampling layer, a Conv2D transpose layer and a Conv2D layer.
- The up-sampling layer(Dense layer) scales the input to the dimension of the flattened output of encoder.
- The Conv2D transpose layer transforms the input in opposite direction of a normal convolution to obtain the shape of the input.
- The last layer is yet again a convolutional layer only with a single feature map to obtain the reconstructed input conform to original size.

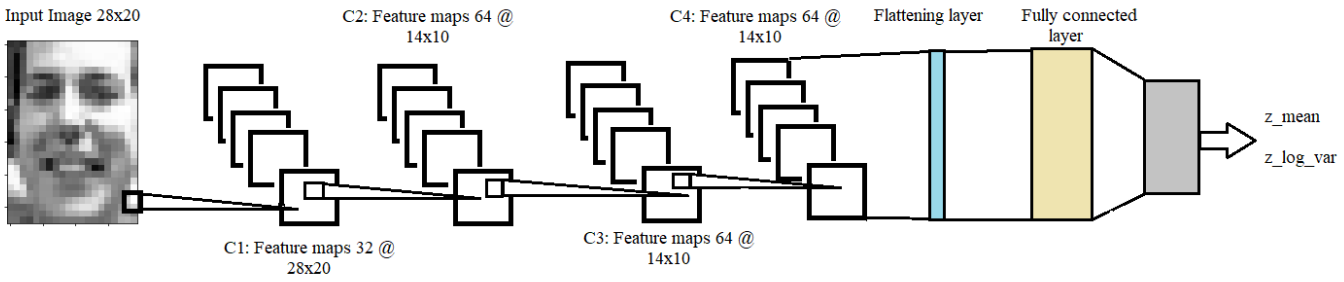

Figure 4.2: Architectural graph of our Encoder

Total Number of Parameters

To calculate the total number of parameters to be trained in either encoder or decoder, there are three important calculations to be considered `https://stanford.edu/~shervine/`

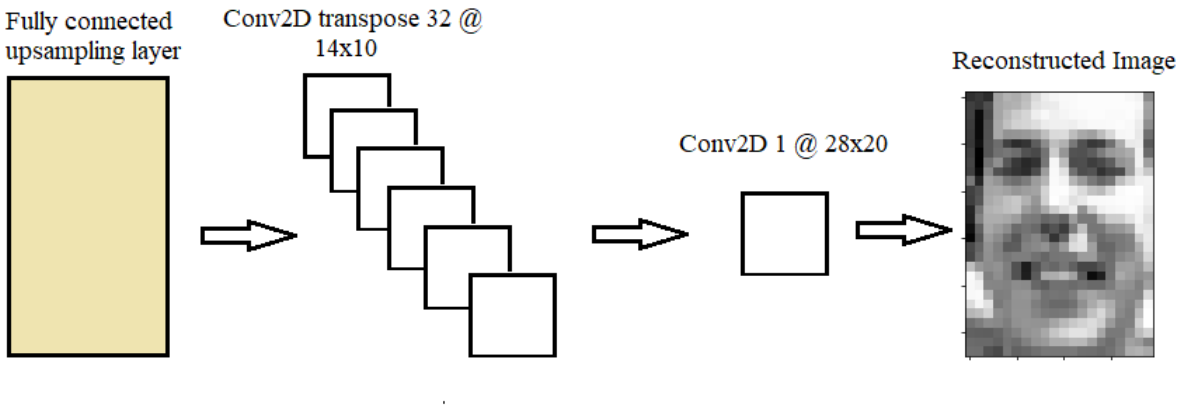

Figure 4.3: Architectural graph of our Decoder

`teaching/cs-230/cheatsheet-convolutional-neural-networks`.

- **Filter Dimensions:** Applying filters of size $F \times F$ to an input containing C channels is an $F \times F \times C$ volume that performs convolutions on an input of size $I \times I \times C$ and produces an output feature map (also called activation map) of size $O \times O \times 1$. If there are K feature maps then output has dimensions $O \times O \times K$.

- **Stride:** For a convolutional or pooling operation, stride S denotes the number of pixels by which the window moves after each operation.

- **Zero-padding:** This denotes the process of adding P zeros to each side of the boundaries of the input. If there is no padding $P = 0$. When the padding is kept same,

$$P_{start} = \left\lfloor \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rfloor \tag{4.1}$$

$$P_{end} = \left\lceil \frac{S\lceil \frac{I}{S} \rceil - I + F - S}{2} \right\rceil \tag{4.2}$$

  The dimensions of the output O is then given by

$$O = \frac{I - F + P_{start} + P_{end}}{S} + 1 \tag{4.3}$$

Layer (type)	Output Shape	No.of Parameters
input_1 (InputLayer)	[(None, 28, 20, 1)]	0
conv2d (Conv2D)	(None, 28, 20, 32)	320
conv2d_1 (Conv2D)	(None, 14, 10, 64)	18496
conv2d_2 (Conv2D)	(None, 14, 10, 64)	36928
conv2d_3 (Conv2D)	(None, 28, 20, 1)	36928
flatten (Flatten)	(None, 8960)	0
dense (Dense)	(None, 32)	286752
z_mean (Dense)	(None, 2)	66
z_log_var (Dense)	(None, 2)	66

Table 4.1: Parameters summary for the Encoder

- **Encoder:** The number of parameters to be trained in case of the encoder is summarized in Table 4.1.

  1. The Input Layer refers to data source, which in our case is a gray-scale image of size 28×20, and has 0 trainable parameters.
  2. The second layer $conv2d$ is a 2D convolutional layer with $K = 32$ feature maps. The filter size is $F = 3$, number of channels $C = 1$, then number of trainable parameters is $(3 \times 3 \times 1 + 1) \times 32 = 320$.
  3. The output from $conv2d\_1$ layer has dimensions 14×10. This is obtained by using equations (4.1), (4.2), (4.3), with $I = 28(20)$, $F = 3$, $S = 2$, $P_{start} = 0$ and $P_{end} = 1$. Since there are 64 feature maps, total number of trainable parameters for this layer is $(3 \times 3 \times 32 + 1) \times 64 = 18496$.
  4. Similarly in the $conv2d\_2$ layer, number of trainable parameters is $(3 \times 3 \times 64 + 1) \times 64 = 36928$.

Layer (type)	Output Shape	No.of Parameters
input_2 (InputLayer)	[(None, 2)]	0
dense_1 (Dense)	(None, 8960)	26880
reshape (Reshape)	(None, 14, 10, 64)	0
conv2d_transpose (Conv2DTranspose)	(None, 28, 20, 32)	18464
conv2d_4 (Conv2D)	(None, 28, 20, 1)	289

Table 4.2: Parameters summary for the Decoder

5. The flattening layer has no trainable parameters since it compresses the input into a vector of size $14 \times 10 \times 64 = 8960$.
6. The fully-connected/dense layer has 32 neural units and trainable parameters for this layer is $8960 \times 32 + 32 = 286752$.
7. The next two dense layers is to obtain the mean and variance of the Gaussian distribution, z_mean, $z\_\log\_var$, and number of trainable parameters for each of these layers is $32 \times 2 + 2 = 66$.

- **Decoder:** The number of parameters to be trained in case of decoder is summarized in Table 4.2

1. The Input Layer of the decoder is the output of encoder and has no trainable parameters.
2. The second layer $dense\_1$ is the dense layer and consists of 8960 neural units and hence trainable parameters is $8960 \times 2 + 8960 = 26880$.
3. The third layer reshapes the output from previous layer to that of the shape obtained before flattening in the encoder. There are no trainable parameters in this layer. This is also called the upsampling layer.

4. The *conv2d_transpose* layer transforms the input to obtain shape of the input. This layer has $(3 \times 3 \times 64 + 1) \times 32 = 18464$.

5. The last layer *conv2d_4* has a single feature map and trainable parameters $3 \times 3 \times 32 + 1 = 289$.

Testing the Learned Model

To test the learned model, new samples needs to be generated from $P(z|X)$. This is, in general, not tractable. The ELBO cannot be computed in closed form due to the expectation over z, which requires sampling. However, sampling z from Q gives an estimator for the expectation which generally converges much faster than sampling z from $\mathcal{N}(0, \mathcal{I})$. Hence, this lower bound can be a useful tool for getting a rough idea of how well our model is capturing a particular data point X.

At test time, we simply input values of $z \sim \mathcal{N}(0, I)$ into the decoder. That is, we remove the "encoder," including the multiplication and addition operations that would change the distribution of z.

Chapter 5

EXPERIMENTS

Experiments were conducted along two different tracks. One - with fixed value of β, reconstruction quality was examined for different latent dimensions. Two - varying β to observe the reconstructions.

5.1 Experimental Observations

1. When β is gradually increased from $0.000001 - 1000$ (in steps of $\times 10$), we obtain sampled reconstructions and corresponding losses as shown in figures 5.4, 5.6 and 5.8.

2. Figures 5.4, 5.6 and 5.8 refer to reconstructions and losses of latent dimension being 2, 3 and 5 respectively.

3. Now, consider the faces in 5.4. Here, for $\beta < 0.01$, the individual reconstructions in the 6×6 grid seem distinguishable from each other. We can even recognize 'orientation of the head' and 'smile' as two distinct dimensions.

4. When $\beta \geq 0.01$ we observe that faces in the grid become indistinguishable.

5. An important observation is also made with respect to losses. Again for $\beta < 0.01$ there is a steady decline in training losses across all ten epochs. But, for $\beta \geq 0.01$ the slope of training losses remains almost a constant.

6. The experiment is repeated for latent dimensions 3, 5 and found that observations are similar.

7. Earlier we had hypothesized that there is a trade-off created between reconstruction fidelity and the quality of disentanglement within the learnt representations for high values of β. An experiment is conducted to investigate this.

Degree of Disentanglement

Higher values of β encourage disentangled learning, which was explained theoretically in the earlier chapter. This characteristic is experimentally observed by obtaining a distance metric for various β values. The metric is obtained as follows:

1. After training the network, representation $r(X)$ learnt by the encoder has mean and covariance as $\mu_i(X)$ and $\Sigma_i(X)$ respectively.

2. The empirical mean and covariance $\tilde{\mu}(X)$, $\tilde{\Sigma}(X)$ are calculated by finding the mean and covariance of the samples generated by $r(X)$. This is called the mean representation $\tilde{r}(X)$, used for reconstruction of the original image in VAE and β-VAE model.

3. Then we calculate the mean of the Frobenius norm $\|\Sigma(X), \tilde{\Sigma}(X)\|_F$ between the covariance matrices $\Sigma(X)$ and $\Sigma(\tilde{X})$ i.e $\tilde{\mathcal{F}} = \frac{1}{N}\sum_{i=1}^{N} ||\Sigma_i(X) - \tilde{\Sigma}(X)||$.

As previously stated, we optimize the KL-loss between $Q(z|X)$ and the prior $P(z)$, which is taken to be an isotropic Gaussian. Taking the Frobenius distance between sampled and mean representations shows the convergence of sampled representations toward more disentangled learning as β increases. This can be seen in figure 5.1 for latent dimensions 2, 3, and 5.

In figure 5.2, samples from the mean representation $Q(z|X) = \mathcal{N}(\tilde{\mu}(X), \tilde{\Sigma}(X)$ are plotted against prior $P(Z) = \mathcal{N}(0, I)$. The plot 5.2 (a) in blue represents samples (≈ 10000) drawn from a $2D$ standard Gaussian ($P(z)$). Plots 5.2 (b) - (l) are samples (≈ 10000) drawn from the encoded representation/aggregate posterior $Q(z|X)$ for various values of β. We observe that there is a 'shearing' effect until $\beta = 1$. We observe that as β increases the approximate posterior approaches that of the prior and has better disentanglement.

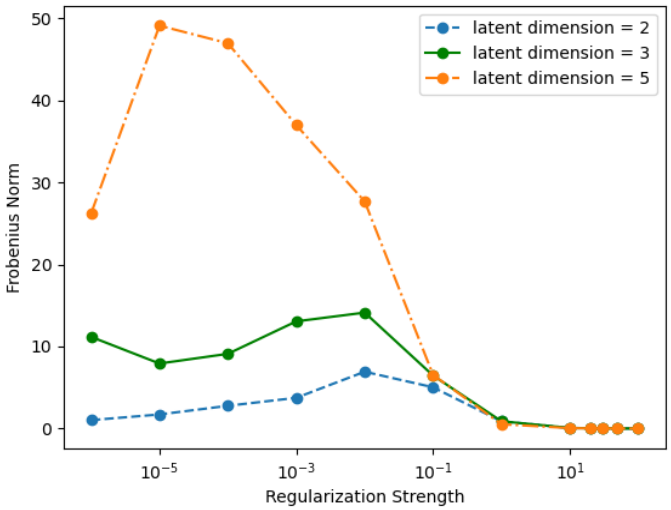

Figure 5.1: $\|\Sigma(X), \tilde{\Sigma}(X)\|_F$ for various values of β

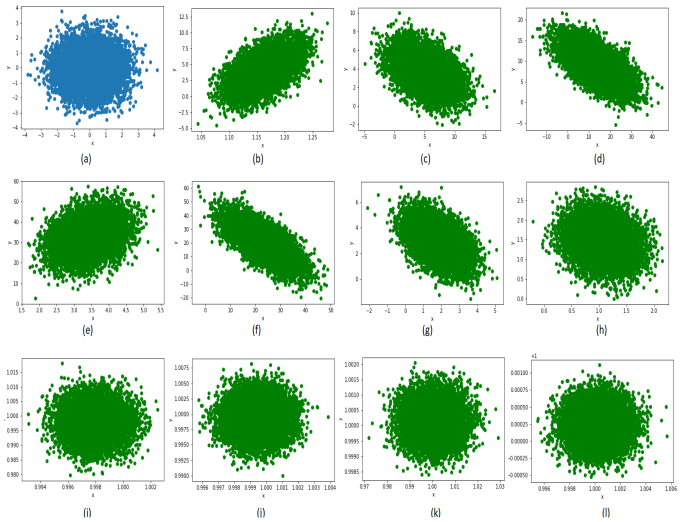

Figure 5.2: Samples from prior $P(z)$ and aggregate posterior $Q(z|X)$ for increasing β values (a) $P(z) = \mathcal{N}(0, I)$, (b) $Q(z|X)$ for $\beta = 0.00001$, (c) $Q(z|X)$ for $\beta = 0.0001$,(d) $Q(z|X)$ for $\beta = 0.001$, (e) $Q(z|X)$ for $\beta = 0.01$, (f) $Q(z|X)$ for $\beta = 0.1$, (g) $Q(z|X)$ for $\beta = 1$, (h) $Q(z|X)$ for $\beta = 10$, (i) $Q(z|X)$ for $\beta = 20$, (j) $Q(z|X)$ for $\beta = 30$, (k) $Q(z|X)$ for $\beta = 50$, (l) $Q(z|X)$ for $\beta = 100$

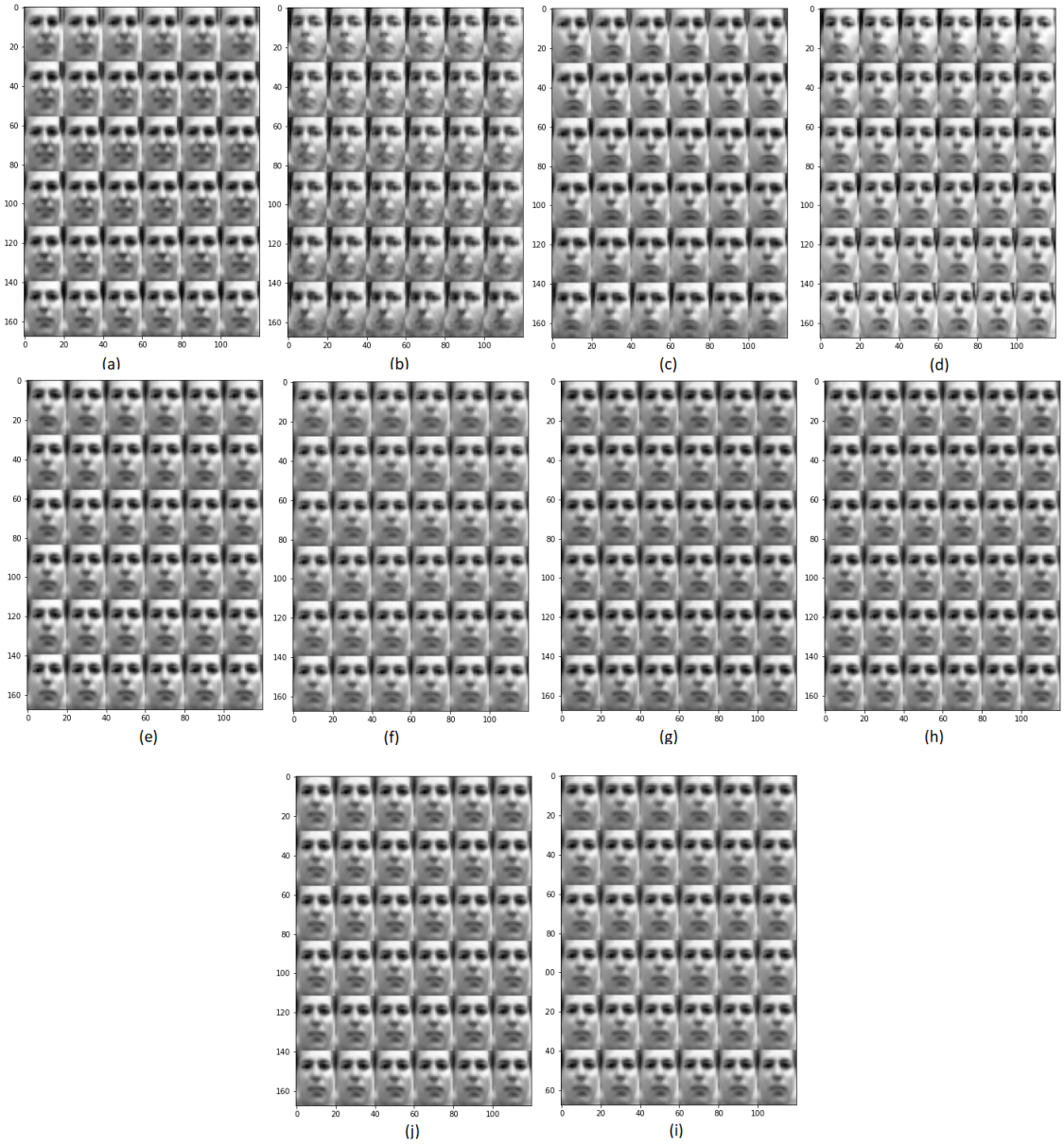

Figure 5.3: Sampled reconstructions for $z \in \Re^2$ (a) $\beta = 0.000001$, (b) $\beta = 0.00001$, (c) $\beta = 0.0001$,(d) $\beta = 0.001$, (e) $\beta = 0.01$, (f) $\beta = 0.1$, (g) $\beta = 1$, (h) $\beta = 10$, (i) $\beta = 100$, (j) $\beta = 1000$

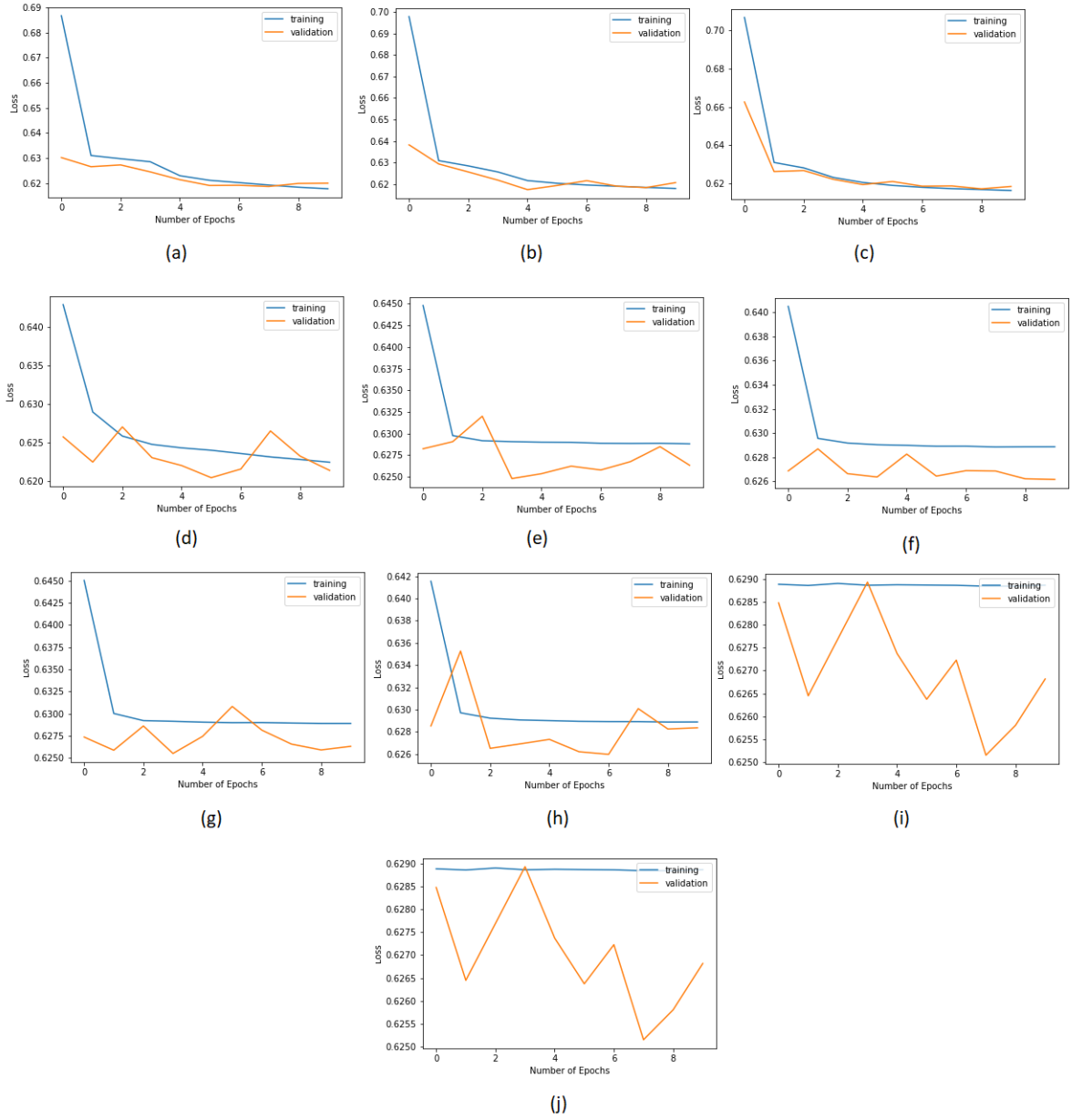

Figure 5.4: Sampled losses for $z \in \Re^2$ (a) $\beta = 0.000001$, (b) $\beta = 0.00001$, (c) $\beta = 0.0001$,(d) $\beta = 0.001$, (e) $\beta = 0.01$, (f) $\beta = 0.1$, (g) $\beta = 1$, (h) $\beta = 10$, (i) $\beta = 100$, (j) $\beta = 1000$

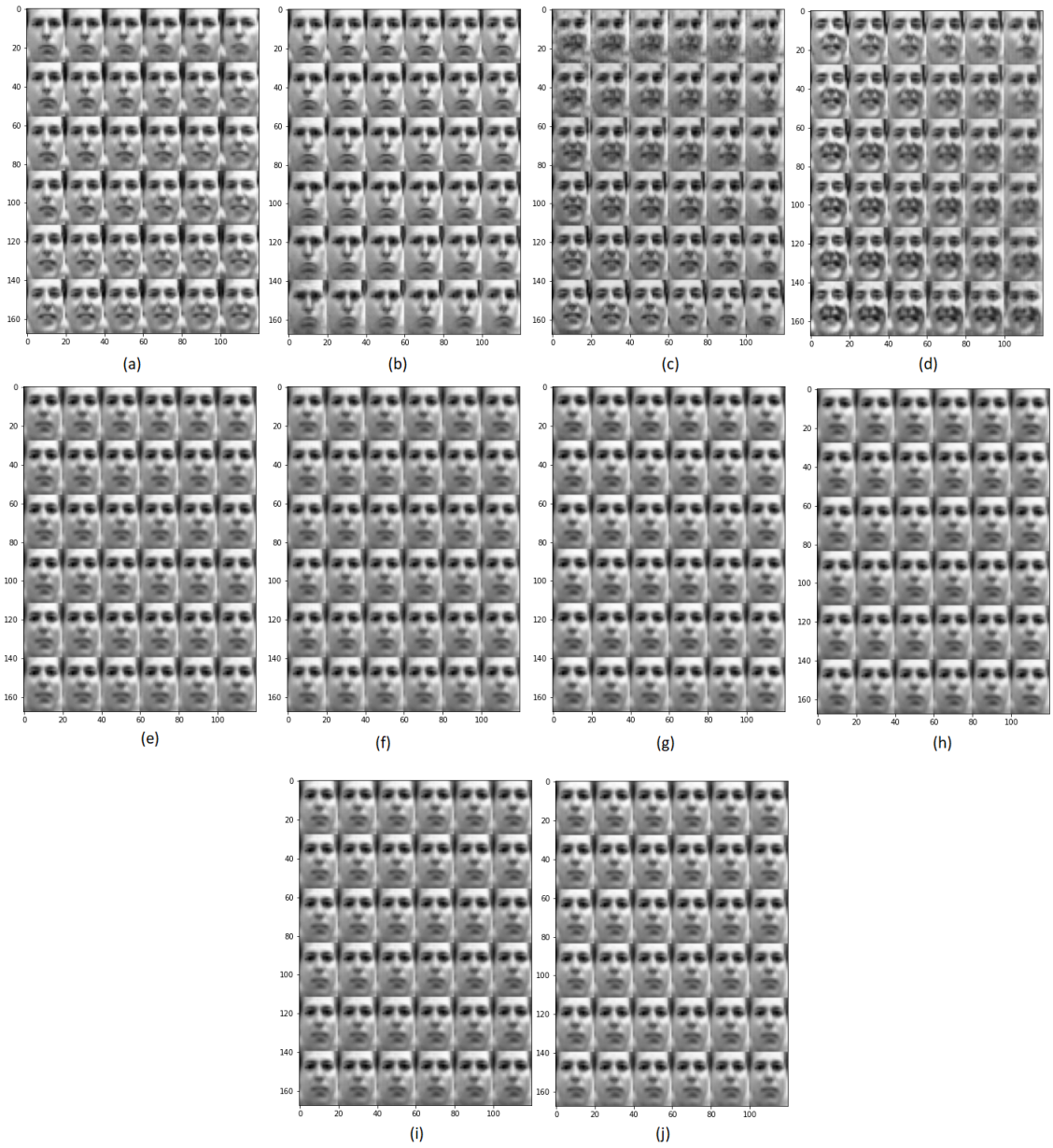

Figure 5.5: Sampled reconstructions for $z \in \Re^3$ (a) $\beta = 0.000001$, (b) $\beta = 0.00001$, (c) $\beta = 0.0001$,(d) $\beta = 0.001$, (e) $\beta = 0.01$, (f) $\beta = 0.1$, (g) $\beta = 1$, (h) $\beta = 10$, (i) $\beta = 100$, (j) $\beta = 1000$

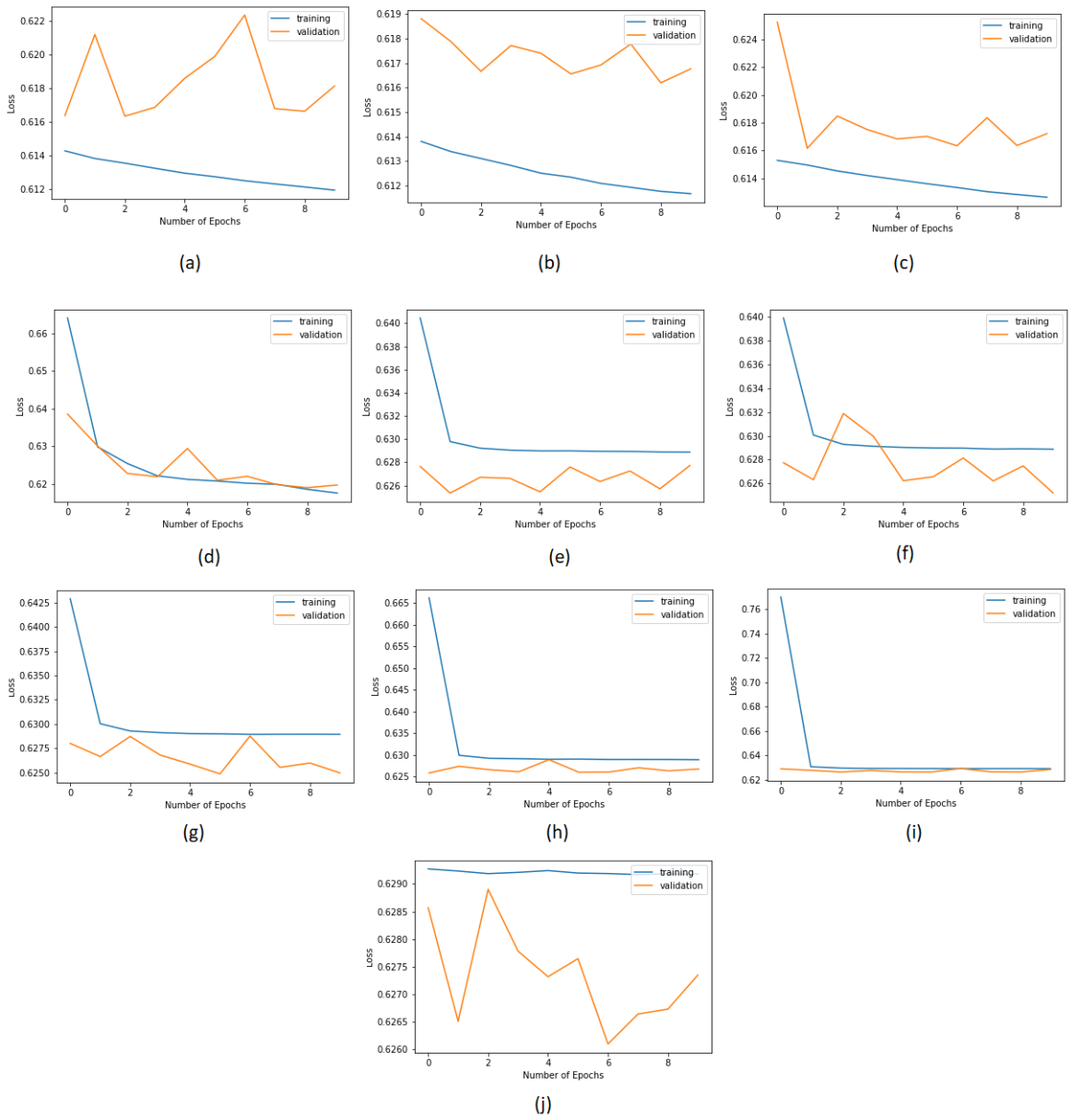

Figure 5.6: Sampled losses for $z \in \Re^3$ (a) $\beta = 0.000001$, (b) $\beta = 0.00001$, (c) $\beta = 0.0001$,(d) $\beta = 0.001$, (e) $\beta = 0.01$, (f) $\beta = 0.1$, (g) $\beta = 1$, (h) $\beta = 10$, (i) $\beta = 100$, (j) $\beta = 1000$

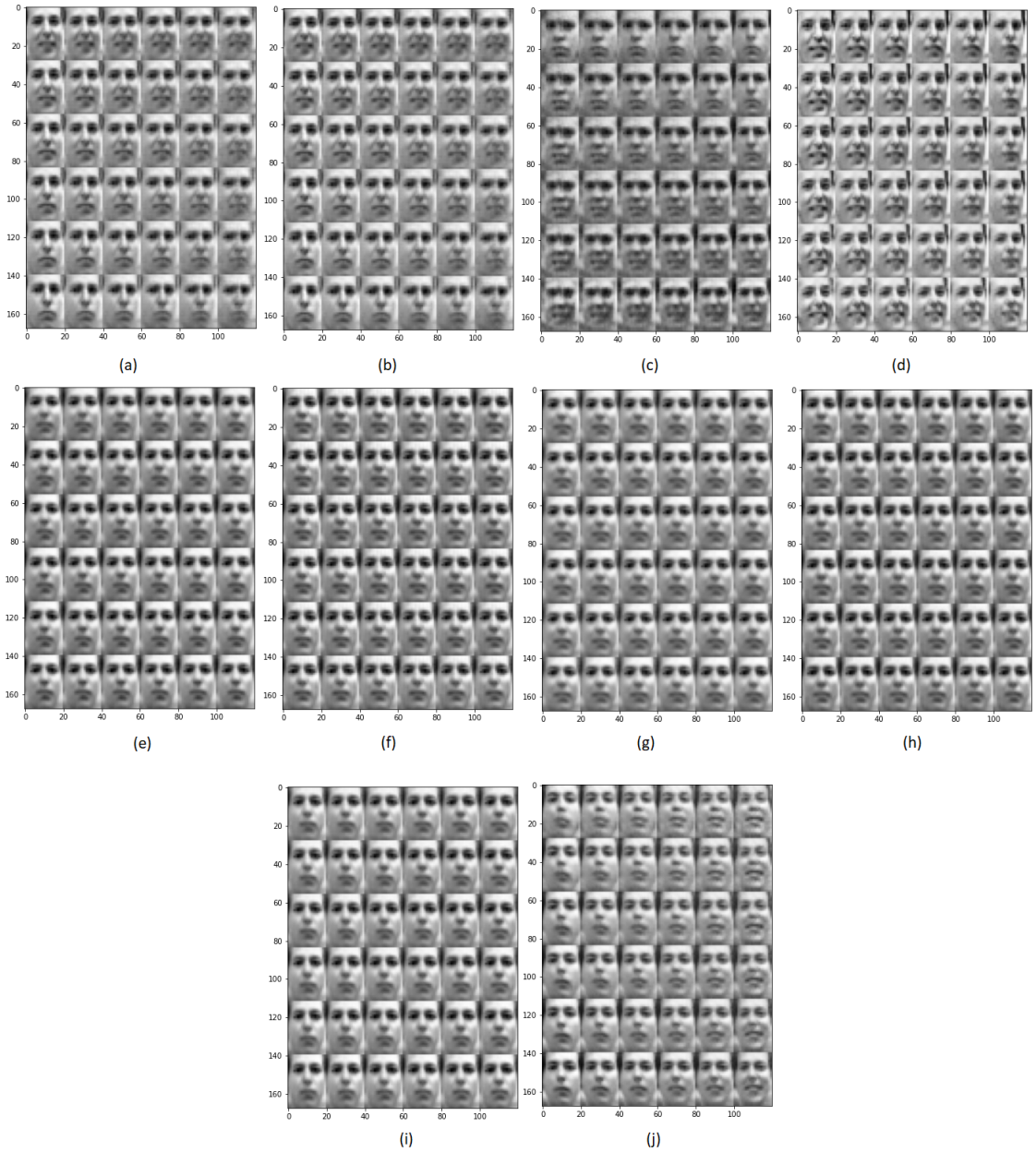

Figure 5.7: Sampled reconstructions for $z \in \Re^5$ (a) $\beta = 0.000001$, (b) $\beta = 0.00001$, (c) $\beta = 0.0001$,(d) $\beta = 0.001$, (e) $\beta = 0.01$, (f) $\beta = 0.1$, (g) $\beta = 1$, (h) $\beta = 10$, (i) $\beta = 100$, (j) $\beta = 1000$

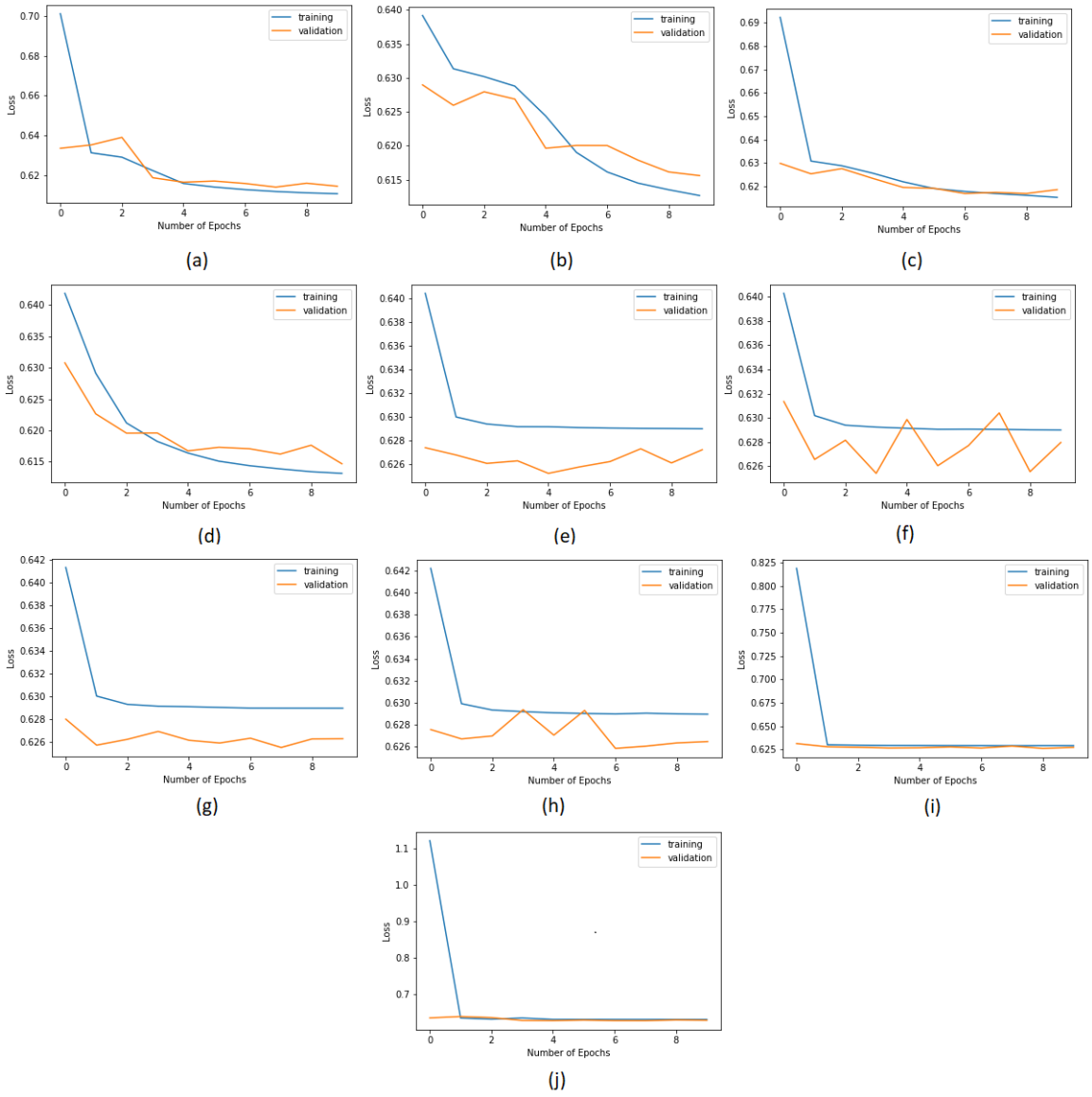

Figure 5.8: Sampled losses for $z \in \Re^5$ (a) $\beta = 0.000001$, (b) $\beta = 0.00001$, (c) $\beta = 0.0001$,(d) $\beta = 0.001$, (e) $\beta = 0.01$, (f) $\beta = 0.1$, (g) $\beta = 1$, (h) $\beta = 10$, (i) $\beta = 100$, (j) $\beta = 1000$

Chapter 6

DISENTANGLEMENT METRICS AND CONCLUSIONS

6.1 Disentanglement Metrics

Good representations disentangle the factors of variation effectively. This is based on the assumption that data is generated by few explanatory factors of variation. Different explanatory factors of the data tend to change independently of each other in the input distribution, and only a few at a time tend to change when one considers a sequence of consecutive real-world inputs. To realize such a representation there have been various models proposed in literature, unsupervised and semi-supervised. What makes one representation better than the other? How do we compare two representations?

The importance of being able to quantify the level of disentanglement achieved by different models was first proposed by [Higgins et al., 2016]. Though designing a metric is not straightforward, analyzing characteristics of good representations gives a great starting point. We assume that the data is generated by a ground truth simulation process which uses a number of generative factors, some of which are conditionally independent, and we also assume that they are interpretable. For example, the simulator might sample independent factors corresponding to object shape, colour and size to generate an image of a small green apple. Because of the independence property, the simulator can also generate small red apples or big green apples. A representation of the data that is disentangled with respect to these generative factors, i.e. which encodes them in separate latent variables would enable robust classification even using very simple linear classifiers (hence providing interpretability). For example, a classifier that learns a decision boundary that relies on object shape would perform well when other generative factors, such as size or colour, are varied. Note that a representation consisting of independent latent variables is not necessarily dis-

entangled, according to our desiderata. Independence can readily be achieved by a variety of approaches (such as PCA or ICA) that learn to project the data onto independent bases. Representations learnt by such approaches do not in general align with the data generative factors and may lack interpretability. For this reason, a simple cross-correlation calculation between the inferred latent variables would not suffice as a disentanglement metric.

The various disentanglement metrics proposed in literature can be categorized based on using two main approaches - first, based on the $\beta - VAE$ metric and second, based on calculating the mutual information between the latent variables.

6.1.1 The β-VAE metric

The $\beta - VAE$ metric proposed by [Higgins et al., 2016] aims at measuring both interpretability and independence. A number of images are generated by fixing the value of one data generative factor while randomly sampling all others. Latent variables are inferred on these images and if the independence and interpretability properties hold for the inferred representations, there will be less variance in the inferred latent variables that correspond to the fixed generative factor [Higgins et al., 2016]. A low capacity linear classifier is used to identify this factor and report the accuracy value as the final disentanglement metric score. Smaller variance in the latents corresponding to the target factor will make the job of this classifier easier, resulting in a higher score under the metric [Higgins et al., 2016]. The algorithm [Higgins et al., 2016] is defined as below:

1. If there are K generative factors, choose one among those i.e $y \sim Unif[1 \dots K]$.

2. For a batch of L samples:

    a) Sample two sets of latent representations, $v_{1,l}$ and $v_{2,l}$, enforcing $[v_{1,l}]_k = [v_{2,l}]_y$ if $k = y$ (so that the value of factor $k = y$ is kept fixed).

    b) Simulate image $x_{1,l} \sim Sim(v_{1,l})$, then infer $z_{1,l} = \mu(x_{1,l})$, using the encoder $Q(z|x) \sim \mathcal{N}(\mu(X), \Sigma(X))$. Repeat the process for $v_{2,l}$.

c) Compute the difference $z^l_{diff} = |z_{1,l} - z_{2,l}|$, the absolute linear difference between the inferred latent representations.

3. Use the average $z^b_{diff} = \frac{1}{L}\sum_{l=1}^{L} z^l_{diff}$ to predict $p(y|z^b_{diff})$ and report the accuracy of this predictor as disentanglement metric score.

The goal of the classifier is to predict the index y of the generative factor that was kept fixed for a given z^b_{diff}. The accuracy of this classifier over multiple batches is used as the disentanglement metric score. A linear classifier is chosen to ensure that it has no capacity to perform nonlinear disentangling by itself. Differences between two inferred latent vectors is taken to reduce the variance in the inputs to the classifier and also reduce conditional dependence on the inputs x.

6.1.2 Mutual Information Gap (MIG)

This metric uses the mutual information between a latent factor z_j and a ground truth factor v_k [Chen et al., 2018]. A higher mutual information implies that z_j contains a lot of information about v_k, and the mutual information is maximal if there exists a deterministic, invertible relationship between z_j and v_k. Since a single factor can have high mutual information with multiple latent variables, axis-alignment is enforced by measuring the difference between the top two latent variables with highest mutual information. The full metric is called *mutual information gap (MIG)* and is given by [Chen et al., 2018]:

$$\frac{1}{K}\sum_{k=1}^{K}\frac{1}{H(v_k)}\Big(I(z_{j(k)};v_k) - \max_{j\neq j^{(k)}} I(z_{j(k)};v_k)\Big) \tag{6.1}$$

where $j^{(k)} = \arg\max_j I(z_{j(k)};v_k)$, and $I(z_{j(k)};v_k)$ is the mutual information between z_j and v_k and $H(v_k)$ is the entropy of ground truth factor v_k.

6.2 Conclusions

In this thesis, we saw that the supervised learning has its own limitations and there is a need for unsupervised learning to solve complex problems. Generative modeling becomes

an important tool in such cases and we study *Variational Autoencoders*, one such generative model using deep neural networks. The theory behind *variational autoencoders*(VAE) is studied in detail. We also study how introducing the regularization term β to vanilla VAE results in creating a trade-off between reconstruction fidelity and quality of disentanglement. We validate this hypothesis by conducting experiments on the "Frey's Face" dataset. Lastly, we study about disentanglement metrics and two main approaches to constructing them by a few examples. The conclusions we derived are:

1. *Variational autoencoder (VAE)* is an effective tool for generative modeling.

2. In regularized VAEs, the hyperparameter β determines the trade off between reconstruction fidelity and the quality of disentanglement within the learnt representations. As we increased β value from $0.00001 - 1000$ in steps of $\times 10$, the training and validation losses from the VAE indicates there is "learning" only until when $\beta = 0.01$.

3. The β-VAE metric and *Mutual Information Gap (MIG)* denote two different approaches to quantifying amount of "disentanglement" achieved by a given model on defined dataset.

BIBLIOGRAPHY

[Becker and Hinton, 1992] Becker, S. and Hinton, G. E. (1992). Self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355(6356):161–163.

[Bengio et al., 2013] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828.

[Bengio et al., 2007] Bengio, Y., LeCun, Y., et al. (2007). Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5):1–41.

[Bishop, 1998] Bishop, C. M. (1998). Latent variable models. In *Learning in graphical models*, pages 371–403. Springer.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

[Chen et al., 2018] Chen, R. T., Li, X., Grosse, R. B., and Duvenaud, D. K. (2018). Isolating sources of disentanglement in variational autoencoders. *Advances in neural information processing systems*, 31.

[Dayan et al., 1995] Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The helmholtz machine. *Neural computation*, 7(5):889–904.

[Doersch, 2016] Doersch, C. (2016). Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908*.

[Eastwood and Williams, 2018] Eastwood, C. and Williams, C. K. (2018). A framework for the quantitative evaluation of disentangled representations. In *International Conference on Learning Representations*.

[Haykin et al., 2009] Haykin, S. S. et al. (2009). Neural networks and learning machines/simon haykin.

[Higgins et al., 2016] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2016). beta-vae: Learning basic visual concepts with a constrained variational framework.

[Hinton et al., 1995] Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The" wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161.

[Hinton et al., 2011] Hinton, G. E., Krizhevsky, A., and Wang, S. D. (2011). Transforming auto-encoders. In *International conference on artificial neural networks*, pages 44–51. Springer.

[Hinton et al., 2006] Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

[Hinton and Salakhutdinov, 2006] Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507.

[Hinton and Zemel, 1994] Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length, and helmholtz free energy. *Advances in neural information processing systems*, 6:3–10.

[Hoffman and Johnson, 2016] Hoffman, M. D. and Johnson, M. J. (2016). Elbo surgery: yet another way to carve up the variational evidence lower bound. In *Workshop in Advances in Approximate Bayesian Inference, NIPS*, volume 1.

[Hornik et al., 1989] Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366.

[Jordan et al., 1999] Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233.

[Kim and Mnih, 2018] Kim, H. and Mnih, A. (2018). Disentangling by factorising. In *International Conference on Machine Learning*, pages 2649–2658. PMLR.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Kumar et al., 2017] Kumar, A., Sattigeri, P., and Balakrishnan, A. (2017). Variational inference of disentangled latent concepts from unlabeled observations. *arXiv preprint arXiv:1711.00848*.

[Lecun, 1987] Lecun, Y. (1987). Phd thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models).

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

[Locatello et al., 2019] Locatello, F., Bauer, S., Lucic, M., Raetsch, G., Gelly, S., Schölkopf, B., and Bachem, O. (2019). Challenging common assumptions in the unsupervised learning of disentangled representations. In *international conference on machine learning*, pages 4114–4124. PMLR.

[Makhzani and Frey, 2017] Makhzani, A. and Frey, B. (2017). Pixelgan autoencoders. *arXiv preprint arXiv:1706.00531*.

[Olshausen and Field, 1996] Olshausen, B. A. and Field, D. J. (1996). Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609.

[Petersen et al., 2008] Petersen, K. B., Pedersen, M. S., et al. (2008). The matrix cookbook. *Technical University of Denmark*, 7(15):510.

[Salakhutdinov and Hinton, 2009] Salakhutdinov, R. and Hinton, G. (2009). Deep boltzmann machines. In *Artificial intelligence and statistics*, pages 448–455. PMLR.

[Sepliarskaia et al., 2019] Sepliarskaia, A., Kiseleva, J., and de Rijke, M. (2019). How to not measure disentanglement. *arXiv preprint arXiv:1910.05587*.

[Snyder, 2014] Snyder, J. M. (2014). *Generative modeling for computer graphics and CAD: symbolic shape design using interval analysis*. Academic press.

[Wainwright and Jordan, 2008a] Wainwright, M. J. and Jordan, M. I. (2008a). *Graphical models, exponential families, and variational inference*. Now Publishers Inc.

[Wainwright and Jordan, 2008b] Wainwright, M. J. and Jordan, M. I. (2008b). Introduction to variational methods for graphical models. *Foundations and Trends in Machine Learning*, 1:1–103.

[Weinstock, 1974] Weinstock, R. (1974). *Calculus of variations: with applications to physics and engineering*. Courier Corporation.

[Zhai et al., 2018] Zhai, J., Zhang, S., Chen, J., and He, Q. (2018). Autoencoder and its various variants. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 415–419. IEEE.

Appendix A

A.1 KL Divergence between two multi-variate Gaussians

By definition, $\mathcal{D}(p||q) = \int p(x) \log \frac{p(x)}{q(x)} dx$.

$$p(X) = \frac{1}{(2\pi)^{k/2}|\Sigma_0|^{1/2}} \exp\Big(-\frac{1}{2}(X-\mu_0)^T\Sigma_0^{-1}(X-\mu_0)\Big)$$
$$q(X) = \frac{1}{(2\pi)^{k/2}|\Sigma_1|^{1/2}} \exp\Big(-\frac{1}{2}(X-\mu_1)^T\Sigma_1^{-1}(X-\mu_1)\Big)$$

where k is the dimension of the multivariate Gaussians.

$$\mathcal{D}(p||q) = \mathbb{E}_p[\log(p) - \log(q)] \tag{A.1}$$
$$= \mathbb{E}_p\Big[\frac{1}{2}\log\frac{|\Sigma_1|}{|\Sigma_0|} - \frac{1}{2}[(X-\mu_0)^T\Sigma_0^{-1}(X-\mu_0) + (X-\mu_1)^T\Sigma_1^{-1}(X-\mu_1)]\Big] \tag{A.2}$$
$$= \frac{1}{2}\log\frac{|\Sigma_1|}{|\Sigma_0|} - \frac{1}{2}\mathbb{E}_p[(X-\mu_0)^T\Sigma_0^{-1}(X-\mu_0)] + \frac{1}{2}\mathbb{E}_p[(X-\mu_1)^T\Sigma_1^{-1}(X-\mu_1)] \tag{A.3}$$

Now, the second term of A.3 $\in \Re$ can be re written as (using the property from [Petersen et al., 2008]):

$$\frac{1}{2}\mathbb{E}_p[(X-\mu_0)^T\Sigma_0^{-1}(X-\mu_0)] = \frac{1}{2}\mathbb{E}_p[Tr(X-\mu_0)(X-\mu_0)^T\Sigma_0^{-1}] \tag{A.4}$$
$$= \frac{1}{2}Tr\Big\{\mathbb{E}_p[(X-\mu_0)(X-\mu_0)^T]\Sigma_0^{-1}\Big\} \tag{A.5}$$

since $\mathbb{E}_p[(X-\mu_0)(X-\mu_0)^T] = \Sigma_0$. Equation A.5 now becomes

$$= \frac{1}{2}Tr\Big\{\Sigma_0\Sigma_0^{-1}\Big\} \tag{A.6}$$
$$= \frac{1}{2}Tr\Big\{I_k\Big\} \tag{A.7}$$
$$= \frac{k}{2} \tag{A.8}$$

The third term of A.3 can be simplified using the equation 380 section 8.2 of [Petersen et al., 2008]:

$$\mathbb{E}_p[(X-\mu_1)^T\Sigma_1^{-1}(X-\mu_1)] = (\mu_0-\mu_1)^T\Sigma_1^{-1}(\mu_0-\mu_1) + Tr\Big\{\Sigma_1^{-1}\Sigma_0\Big\}$$

Combining all the terms we obtain

$$\mathcal{D}(\mathcal{N}(\mu_0,\Sigma_0),\mathcal{N}(\mu_1,\Sigma_1)) = \frac{1}{2}\Big\{\log\frac{|\Sigma_1}{|\Sigma_0|} - k + (\mu_0-\mu_1)^T\Sigma_1^{-1}(\mu_0-\mu_1) + Tr\Big\{\Sigma_1^{-1}\Sigma_0\Big\}\Big\} \tag{A.9}$$

A.2 Proof that Monte Carlo estimate of expectation of $Q(z|X)$ w.r.t variational parameters ϕ is differentiable

Proof is from [Kingma and Welling, 2013]. Given the deterministic mapping $z = g_\phi(\epsilon, X)$ we know that

$$\begin{aligned} Q_\phi(z|X)\prod_i dz_i &= p(\epsilon)\prod_i d\epsilon_i \\ \implies \int Q_\phi(z|X)f(z)\,dz &= \int p(\epsilon)f(z)\,d\epsilon \\ &= \int p(\epsilon)f(g_\phi(\epsilon,X))\,d\epsilon \end{aligned}$$

with $dz = \prod_i dz_i$. It follows that a differentiable constructor can be constructed.

$$\int Q_\phi(z|X)f(z)dz \approx \frac{1}{L}\sum_{l=1}^{L} f(g_\phi(X,\epsilon^{(l)}))$$

where $\epsilon^{(l)} \sim p(\epsilon)$.