Dissertation Type: Machine Learning/Deep Learning/Statistics

DEPARTMENT OF COMPUTER SCIENCE

# Optimal Continuous Relaxations

## of Discrete Bernoulli Random Variables in Variational Autoencoders

Alex Sadier-Gane

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Bachelor of Science in the Faculty of Engineering.

Friday 15th May, 2020

# Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of BSc in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author. This dissertation does not require ethics review as it does not contain any human or animal experimental data.

[illegible]

Alex Sadier-Gane, Friday 15th May, 2020

# Contents

# Abstract

Proposed by Diederik P Kingma and Max Welling in 2013 [17] variational autoencoders (VAEs) are a type of generative neural network, which means they can generate new data. The VAE achieves this by first encoding features of the original training data into an abstract latent space, then points in this space can be decoded to produce new data distinct from training data. This can make them far more powerful than traditional neural networks for classification. Though in simplified terms they work in much the same way: iteratively propagate your input training data forward through the network, use a loss function to quantify how the output of your network differs from the required output, and back-propagate through the network updating its parameters (weights and biases) to minimise loss. The minimisation happens by the method of stochastic gradient descent, and it requires that the loss function be differentiable [3]. This is not the case for all VAEs, specifically when the latent space is discrete rather than continuous.

The latent space is described by random variables and backpropagation is not possible through discrete random variables, preventing the model from being trained. Fixing this issue requires that we approximate, or relax, the underlying discrete random variables with continuous ones.

This project will focus on the case when the latent space is described by a discrete Bernoulli distribution, i.e. when the space has binary coordinates. Current relaxation approaches lead to poor approximations of such a distribution. We introduce a new latent distribution for this task, the StepLogistic which leads to stronger performance when compared to other approaches. This project is largely theoretical in nature, however for testing the theory and comparing models, the training data to be used will be the standard MNIST data-set composed of 60 000 handwritten digits.

# Supporting Technologies

- I used the PyTorch machine learning framework in Python to write the code which ran the neural networks. When training the network, PyTorch automatically computes the relevant gradients to optimise the network's loss function. PyTorch is obtained from `www.pytorch.org`
- I used the MNIST and Fashion MNIST data-sets as training data for the networks. Both data-sets are downloaded and installed automatically alongside PyTorch, or can be downloaded at `http://yann.lecun.com/exdb/mnist/` `https://github.com/zalandoresearch/fashion-mnist`.
- I used Matplotlib and Numpy through Python to graph the data produced by the networks. `https://matplotlib.org/` `https://numpy.org/`.
- I used Desmos to collect some numerical integration values `www.desmos.com`.

# Notation and Acronyms

| | |
|---|---|
| VAE | Variational Autoencoder |
| PMF | Probability mass function |
| PDF | Probability density function |
| CDF | Cumulative density function |
| $p(x\|z)$ | The PMF (discrete) or PDF (continuous) of random variable $x$ conditioned on $z$. $q$ and upper case letters will be used to differentiate between distrubutions. When specifically referring to a PDF the letter $f$ may be used instead. |
| log | The natural (base e) logarithm. |
| $\mathbb{E}_p(x)$ | Expectation of random variable $x$ taken w.r.t distrubution/PMF/PDF p. |
| $x \in \{0,1\}$ | $x = 0$ or $x = 1$. |
| $x \in [0,1]$ | $0 \leq x \leq 1$ |
| $x \in (0,1)$ | $0 < x < 1$ |
| $x^{(i)}$ | the $i$-th dimension of vector $x$. |
| $x_i$ | the i-th vector in the set of vectors $x$. |
| $x \in (0,1)^n$ | $x$ is an n-dimensional vector where each dimension takes value in $(0,1)$. |
| $\phi$ | the network parameters (weights and biases) for the encoder network of a VAE. |
| $\theta$ | the network parameters (weights and biases) for the decoder network of a VAE. |

# Acknowledgements

I would like to thank my Supervisor for helpful discussions and guidance through what is a relatively new and abstract topic. I would also like to thank Ben Sheppard for helping me make Figure 1.2.

# Chapter 1

# Introduction

## 1.1 The variational autoencoder (VAE)

A lot of real-world data such as images of handwritten digits or occurrences of words in the English language can be abstractified by an underlying distribution. What we observe is a projection of that distribution into an ”observation space”; such as pixel values with coordinates, or an English text document with a particular ordering of words. Generative models such as a VAEs describe how the data may have been generated using probability distributions [9].

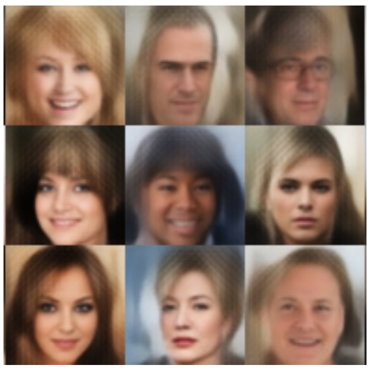


Figure 1.1: Example of faces generated using a VAE. It is clear the training data consisted of unlabelled male and female faces, which means the random images generated don’t separate genders [30].

The new data VAEs generate is a variation of the original data used for the training process. As a rough example, you could train your network using many images of human faces and then be able to generate entirely new samples of faces (see 1.1). These would be distinct from your training data, but no more complex than a variation (the network can’t now be used to generate dog faces). In other words the data generated lies along a smooth manifold of the original data [27]. VAEs are autoencoders in the sense that the input and output space is the same, the ”observation space”, in contrast to more conventional neural networks that map your input to some class vector.

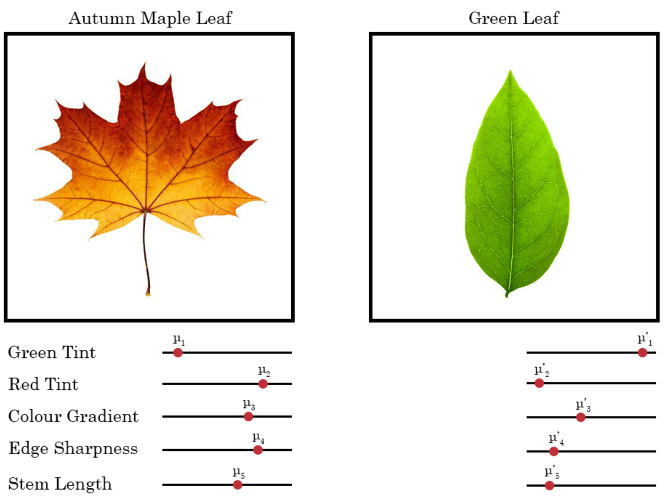
The VAE is effectively formed of two networks. The first uses the input data to form the latent space, and the second samples from this space to generate an output approximate to the input data. These will be called the encoder and decoder respectively.

## 1.2 The latent space

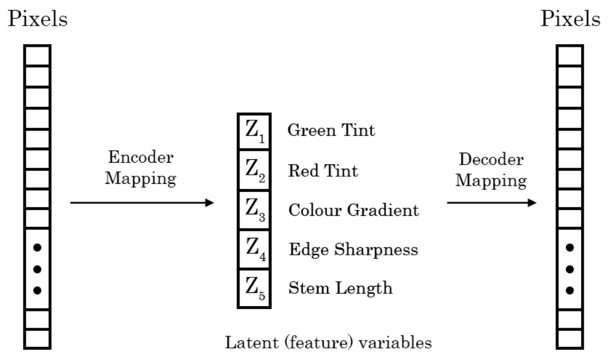
The latent space is defined by a probability distribution. The type of distribution depends on the input data. In the standard case it will be a (continuous) Gaussian distribution, but others (including discrete latent spaces) will be examined later. During the standard training process of the VAE, it is the role of the encoder to find the parameters of the defining distribution (see Chapter 1.3).

In the context of this project, the latent space can be thought of as a vector space where an alternate representation of the input data exists, based on its features. These vectors are latent in the sense that they are not immediately apparent from the data (e.g, not a vector of pixel values), but rather found by your network, and only the network can infer meaning from these vectors.

As an illustrative example, let us consider Figure 1.2. Here, the training data is a set of leaf images and we consider a 5-D latent space consisting of features that define a leaf. We can smoothly transition between various types of leaf by adjusting the distribution parameters defining each dimension of the latent space.



(a) Comparing two leaves by their latent features. The red dots denote each dimension's distribution mean.



(b) Network view.

Figure 1.2: Showing the lower-dimensional latent (feature) representation of the data. Each dimension, or feature, is defined by a probability distribution.

The dimension of the latent space is usually much smaller than that of the input space since we want to simplify the data down to its core defining features. For complex data requiring an encoding of a large number of features, the number of dimensions of the latent space would need to be large to accommodate this complexity. For example, the faces in Figure 1.1 come from a network trained with a 512-dimensional latent space. By comparison, with the MNIST data-set used in this project we can obtain suitable, but not optimal, outputs with as little as two dimensions in continuous space or 8 in discrete space.

The space is defined by an n-dimensional probability distribution. Vector samples from this (latent) distribution represent points in the latent space, with points further away from the mean being increasingly unlikely to sample. Part of the goal of the VAE is to learn the parameters that define this distribution, such as the mean and variance in the Gaussian case. New data is generated by first sampling a vector from this distribution, and the random nature of sampling is what provides the observable variations in the generated output data.

As a rough example, for an n-dimensional Gaussian latent space, the encoder network will output two n-dimensional vectors, one for the means and one for the variances. This defines the distribution to be sampled from during training. A complete description of the training process will follow, but for now it's important to realise the decoder first randomly samples a vector from this latent distribution.
As an example using Gaussian distributions let

$$\mu = \begin{pmatrix} 0.1 \\ 1.2 \\ \ldots \\ \mu_n \end{pmatrix}, \sigma^2 = \begin{pmatrix} 0.2^2 \\ 0.5^2 \\ \ldots \\ \sigma_n^2 \end{pmatrix}$$

be our mean and variance vectors. Then a random sample z would be sampled from

$$\begin{pmatrix} Z_1 \sim \mathcal{N}(0.1, 0.2^2) \\ Z_2 \sim \mathcal{N}(1.2, 0.5^2) \\ \ldots \\ Z_n \sim \mathcal{N}(\mu_n, \sigma_n^2) \end{pmatrix}$$

to give z = $\begin{pmatrix} 0.28 \\ 1.65 \\ \ldots \end{pmatrix}$ as a possible example sample. Each dimension (each latent variable Z) of this vector corresponds to a feature of the the image to be decoded by the generative model (the decoder). It is worth reiterating that these are not necessarily features we might expect (like skin tone or hair colour), rather they simply reflect those patterns which the model finds significant. This is why the latent vector z is also called a hidden vector.

For a standard VAE the latent space is continuous (smooth), and not split into a discrete set of classes. In fact training a VAE doesn't require class labels; the ground truth labels attached with the training data, such as the label "5" attached to the image of handwritten digit 5. Following training of a multi-class data-set (such as MNIST handwritten digits), there will be a section of the latent space dedicated to each class, and the smoothness means it is possible generate data that lies *between* classes. Figure 1.3 shows this smooth interpolation between different different digits. Although it is abstract

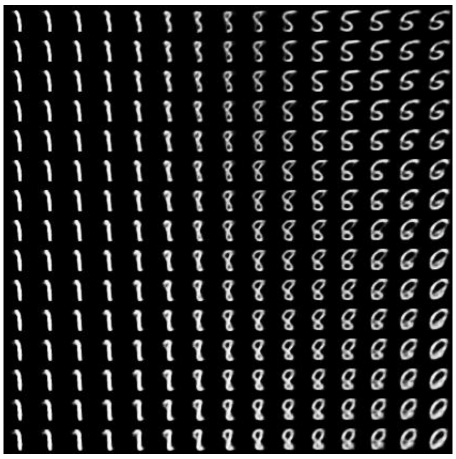


Figure 1.3: Small section of my latent space trained on the MNIST data-set. Samples taken at regular intervals along a segment in the space, then passed to the decoder to generate an image. For this example the space is 56-dimensional, much smaller than the 784-dimensional input/ouput pixel vectors.

at first, exploring and analysing the structure of the space can allow one to generate data with desirable characteristics combined from multiple classes [25].

However it is not always the case that a continuous latent distribution is the most appropriate for your data. For data with spiking activity, such as neural activity in the brain [12] [10] continuous interpolation between spikes may lead to generated outputs which could not possibly exist in the real world. Thus making it unsuitable to describe this type of data using continuous distributions. In this case a discrete latent space is more suitable. In Chapter 4 we consider a binary latent space described by Bernoulli distribution.

## 1.3 Standard Gaussian model outline

### 1.3.1 The encoder

The encoder is a map from the input space to the latent space defined by a probability distribution. In the standard VAE we start by using a simple multi-dimensional Gaussian distribution, though other distributions will be explored in this project.

An input data-point $x^{(i)}$ is passed to the encoder, which will use this to find the parameters of the distribution $Q_\phi(z|x^{(i)}) \sim \mathcal{N}(\mu_\phi(x^{(i)}), \Sigma_\phi(x^{(i)}))$, where $i$ indexes all data-points in the training set. In the case of the MNIST data-set of hand-written digits, each $x^{(i)}$ is a 784-dimensional vector of pixel values corresponding the the 28x28 image. Here, the subscript $\phi$ indicates that the distribution parameter is learned by the encoder network ($\phi$ denotes the weights and biases of the encoder). This network will output the mean vector and diagonal covariance matrix, $\mu_\phi(x^{(i)})$ and $\Sigma_\phi(x^{(i)})$ respectively.

The network type of the encoder and decoder is called a feedforward network or multilayer perceptron [14]. Here, each $x^{(i)}$ undergoes a series of transformations through multiple, fully-connected (linear) layers. These form a multi-dimensional linear composite function (of $x^{(i)}$), and you end up with a vector for the mean and a covariance matrix as the output. This is why our distribution parameters are a function of $x^{(i)}$.

As a simplifying assumption, all the dimensions of the latent distribution will be

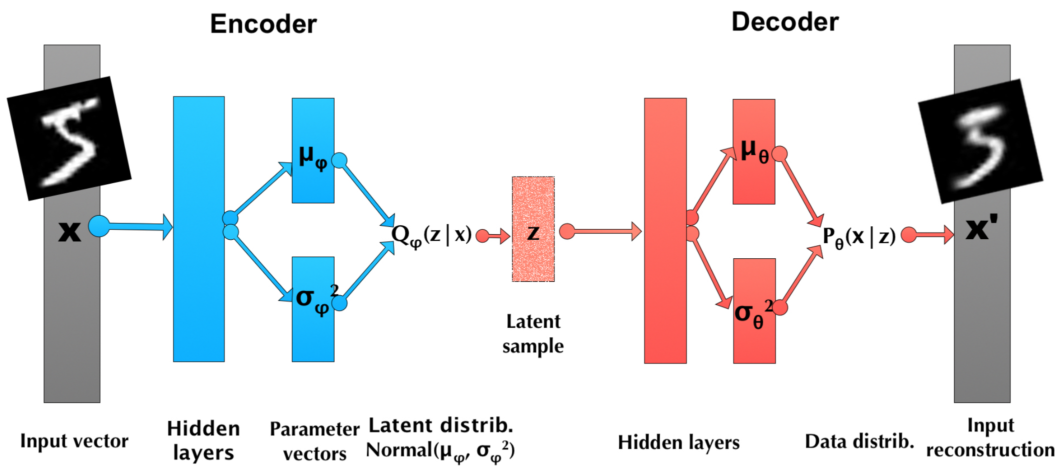


Figure 1.4: Very rough outline of a Gaussian VAE. Encoder is in blue, Decoder is in Red. Reconstruction of the digit "5" done using the model from Chapter 3.3.

independent. More complex models exist such as using "Deep Latent Gaussian Models" where each latent variable dimension is conditioned on the one above it [23], though this adds unnecessary complexity for our purposes. This independence of dimensions means the covariance matrix will be diagonal and so can be simplified by a vector $\sigma_\phi(x^{(i)})$ formed of the log of the diagonal elements of $\Sigma_\phi$. Here we take the log variances for numerical stability purposes. This vector is output by the network in place of the covariance matrix. A latent vector $z$ can now be sampled from $Q_\phi(z|x^{(i)})$, which is why I will also refer to it as the latent distribution (or approximate posterior, see Chapter 2.2). These mean and log-variance vectors are iteratively optimised in accordance with the objective function (see Chapter 2.2).

### 1.3.2 The decoder

The decoder is a map from the abstract latent space to the space defined by the input, whether that be pixel values in the case of images (employed in this project) or even string representations of molecules [13]. In the standard VAE we may assume the input/output space to be defined by a continuous Gaussian distribution $P_\theta(x|z) \sim \mathcal{N}(\mu_\theta(z), \Sigma_\theta(z))$, where now the conditioning is on latent sample $z$ passed to the decoder. This choice of output space will be refined as we start training in the context of MNIST images, where a Gaussian distribution is not the most appropriate.

## 1.4 Project outline and objectives

We will begin by introducing relevant background information important for understanding future work. Then, constructing a working VAE implementation in Pytorch, trained on the MNIST data-set and with the standard continuous latent space. The remainder of the project is structured following these objectives:

1. Construct a fully functioning VAE model with a (discrete) binary latent space using standard approaches as a basis for comparisons with future models.

2. Highlight and visualise issues with the standard approaches, as well as acknowledge useful properties to be retained.

3. Propose a new approximate posterior distribution, the StepLogistic, with new properties motivated by a simplified model. The challenge with finding a suitable approximate posterior is the unknown shape of the true posterior.

4. Describe how to parameterise such a distribution and how to sample from it.

5. Fully implement and analyse its performance against other approximate posterior distributions.

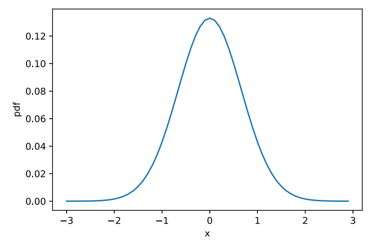
The final section will discuss issues with the novel approach when it comes to implementation, and describe the mathematical complications preventing an easy resolution of such issues.

# Chapter 2

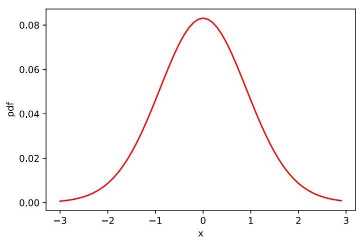
# Technical background

## 2.1 Kullback-Leibler divergence

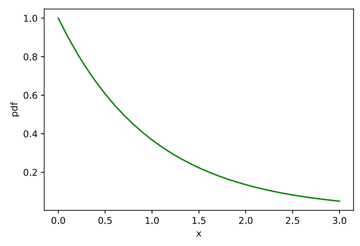
As is the case with any neural network, we must have a loss function to penalise the network for making incorrect predictions. The weights and biases of the network are iteratively updated in such a direction that minimises this loss. A common loss function used for training neural networks is the sum-squared error [16]. However training VAEs involves optimising distributions. It would therefore be useful for the loss function to be able to quantify how much one distribution differs from another. The Kullback-Leibler (K-L) divergence [7] is a tool that does exactly this. As an example take three distrubutions A, B and C shown below: It is clear A and B are quite similar but A and C are very



(a) A



(b) B



(c) C

different. Indeed A and B are both Gaussian distributions but C is Exponential. This would translate to a very low K-L Divergence value close to zero for A and B, and a very high value for A and C. Using proper notation we have

$$D_{KL}(A||B) \approx 0$$

$$D_{KL}(A||C) \gg D_{KL}(A||B)$$

In general the K-L Divergence is defined as follows: For discrete random variables $P$ and $Q$ with probability mass functions $p(x)$ and $q(x)$ respectively

$$\begin{aligned} D_{KL}(P||Q) &= \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)} \\ &= \mathbb{E}_p[\log \frac{p(x)}{q(x)}] \end{aligned} \tag{2.1}$$

where we can swap the summation for an integral and the mass function for a density function for continuous random variables. In information theory terms it measures the expected number of bits (or nats for log base e) of information lost when using P to approximate Q. K-L Divergence has the property that it is non-negative and only equals zero if both distributions are identical. It is worth mentioning that K-L Divergence is non-symmetric, meaning that it would be incorrect to call it a "distance" between two distributions. K-L Divergence will be used in the derivation of the VAE's loss function.

## 2.2 The loss function

### 2.2.1 Variational inference

Let $x$ be the set of observed variables and $z$ be the set of latent variables with joint probability $p(x,z)$. From Baye's Theorem [18] we have that,

$$p(z|x) = \frac{p(x|z)p(z)}{p(x)}. \tag{2.2}$$

The (latent) prior distribution $p(z)$ is chosen by us and in the standard case it is taken to be the unit Gaussian $\mathcal{N}(0,1)$ (more on the role of the prior in a VAE in Chapter 3.2). It would be convenient to be able to compute the posterior $p(z|x)$ directly, using our encoder network. Unfortunately, finding $p(x)$ requires the computation of an intractable (exponential time) integral:

$$p(x) = \int_z p(x|z)p(z)dz.$$

This means the true posterior $p(z|x)$ cannot be computed directly, instead it is approximated using another distribution $Q(z|x)$ which does have a tractable solution. This process is called variational inference [9]. The VAE is essentially just a model which performs variational inference to find the parameters of the latent distribution Q (also referred to as the approximate posterior throughout this document), such as the mean and variance vectors. There exists also the Markov Chain Monte Carlo inference methods [21] which don't rely on using an "approximate" posterior. However, they are usually too slow or computationally expensive to use in practise (except for very simple models) [20]. Other generative models such as Pixel Recurrent Neural Networks [22] can be used for modelling the distribution of images when there *is* a tractable solution.

In the standard variational inference process Q is chosen to be a Gaussian distribution (we will later experiment with other distributions that may better approximate the underlying true posterior).
We want to find the model parameters $\phi$ which make $Q_\phi$ as close as possible to the true posterior. The Kullback-Leibler divergence is used to quantify this difference (information lost in nats), and we seek to minimise it [17].

$$D_{KL}(Q_\phi(z|x)||p(z|x)) = \mathbb{E}_\mathbb{Q}[\log Q_\phi(z|x)] - \mathbb{E}_\mathbb{Q}[\log p(x,z)] + \log p(x)$$

The evidence $p(x)$ appears in the equation so we cannot compute this divergence directly. However, rearranging this equation gives

$$\log p(x) = \mathbb{E}_\mathbb{Q}[\log p(x,z)] - \mathbb{E}_\mathbb{Q}[\log Q_\phi(z|x)] + D_{KL}(Q_\phi(z|x)||p(z|x)).$$

We set

$$ELBO(\phi) = \mathbb{E}_{\mathbb{Q}}[\log p(x,z)] - \mathbb{E}_{\mathbb{Q}}[\log Q_\phi(z|x)] \\ = \underbrace{\mathbb{E}_{\mathbb{Q}}(\log p(x|z))}_{\text{Reconstruction loss}} - \underbrace{D_{KL}(Q_\phi(z|x)||p(z))}_{\text{Regulariser}} \tag{2.3}$$

where the last equality can be derived by expanding terms and re-grouping differently. We now have

$$\log p(x) = ELBO(\phi) + D_{KL}(Q_\phi(z|x)||p(z|x)) \tag{2.4}$$

Since one of the properties of K-L Divergence is non-negativity and $\log p(x)$ is constant, minimising the K-L Divergence term in equation (2.4) is equivalent to maximising the $ELBO(\phi)$ term. ELBO stands for Evidence Lower BOund, the evidence in this case being $p(x)$. Log$p(x)$ is referred to as the Variational Bound, it is the theoretical upper bound for the ELBO but cannot ever be fully reached (unless $p(x)$ is tractable). The distribution parameters $\mu_\theta$ and $\Sigma_\theta$ of the likelihood $p(x|z)$ are calculated by training the decoder network with parameters (weights and biases) $\theta$. This distribution will hence now be referred to as $P_\theta(x|z)$.

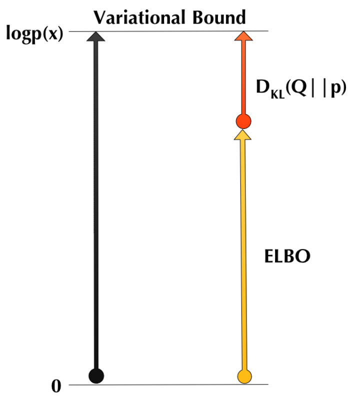


Figure 2.2: Visualising the variational bound. The log probability of the data is the sum of the ELBO and the K-L Divergence term, as we maximise the ELBO we minimise the K-L Divergence between the true and approximate posterior, which is our goal. *Adapted from Figure 11 on page 44 of [9]*

## 2.2.2 The reconstruction loss

Looking back at the ELBO (2.3), we take the log of the (n-dimensional) multivariate Gaussian PDF [28] (with the assumption each dimension is independent).

$$\log P_\theta(x^{(i)}|z) = -\frac{1}{2}(x^{(i)} - \mu_\theta)^T \Sigma_\theta^{-1}(x^{(i)} - \mu_\theta) - \frac{1}{2}\text{Tr}(\log \Sigma) - \frac{n}{2}\log 2\pi$$

where "Tr" denotes the Trace function which sums the diagonal elements of the matrix $\Sigma$. The first term is the squared reconstruction loss between the decoder network's distribution for $x$ and the original data-point $x^{(i)}$ (then multiplied by -0.5). This can be thought of as a measure of how far the decoder is from reconstructing the input perfectly. It therefore makes sense that in maximising the ELBO we minimise this reconstruction loss. The second term is the sum of the log variances of this joint distribution (assuming each dimension is independent) and when maximising the ELBO this term means we minimise the variance of each dimension. So that, in a perfect reconstruction of the latent sample $z$, the output is fixed (it cannot vary). The last term is a constant which does not impact optimisation.

### 2.2.3 Final expectation and approximation

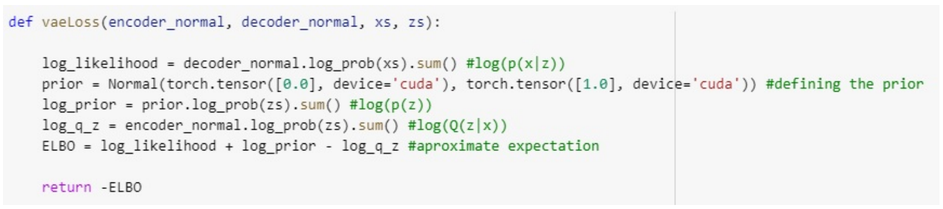
The ELBO can be rewritten as an expectation leading to the final form of the loss function which will be used on each input $x^{(i)}$. Maximise

$$ELBO(\phi, \theta, x^{(i)}) = \mathbb{E}_{Q_\phi}[\log P_\theta(x^{(i)}|z) + \log p(z) - \log Q_\phi(z)] \tag{2.5}$$

Let $l_i = -ELBO(\phi, \theta, x^{(i)})$ denote the loss corresponding to data-point $x^{(i)}$. The total loss for n data points is then given by

$$L = \sum_{i=1}^{n} l_i$$

We want to optimise the network parameters to get $\phi^*, \theta^* = \text{argmin}_{\phi,\theta}(L)$



```
def vaeLoss(encoder_normal, decoder_normal, xs, zs):

    log_likelihood = decoder_normal.log_prob(xs).sum() #log(p(x|z))
    prior = Normal(torch.tensor([0.0], device='cuda'), torch.tensor([1.0], device='cuda')) #defining the prior
    log_prior = prior.log_prob(zs).sum() #log(p(z))
    log_q_z = encoder_normal.log_prob(zs).sum() #log(Q(z|x))
    ELBO = log_likelihood + log_prior - log_q_z #aproximate expectation

    return -ELBO
```

Figure 2.3: Loss function used in Pytorch. The expectation is approximated by the terms which make it.

Such an expectation in (2.5) is often difficult to compute directly (it is an integral over all possible values of z of a complicated function). Fortunately the goal of the VAE is to maximise this expectation, which can be done by just maximising the function inside the square brackets (Monte Carlo expectation approximation [23]). Training is done in batches of inputs and the corresponding batch of latent samples $z$ passed to the decoder are also passed to the loss function each time it is called to compute the relevant log probabilities. This simplifies implementation and still achieves the same goal.

## 2.3 The autoencoder and the need for a regulariser

To complete the background section on variational autoencoders, it is worth briefly mentioning the standard autoencoder (Section 14, [14]) also know as the vanilla autoencoder. Why vanilla? They're nice and simple but lacking in depth-of-flavour.

In an autoencoder, we have the encoder which maps the input to a latent vector with dimensionality strictly less than that of the input. The decoder maps this latent vector back to the "observation space" and is trained to reconstruct the input as perfectly as possible, using a reconstruction loss objective function such as sum-squared-error. This is similar to the VAE's network structure except there is no random sampling of any "latent distribution", so a mean and variance output from the encoder is replaced with just a single vector, the "latent vector".

Autoencoders are useful for compressing data; using an encoder to force the input into a lower-dimensional representation, which can be stored and decoded later for example. Another use is de-noising data; a model with an appropriate number of latent dimensions should not be reconstructing random noise (overfitting) and instead focus on reconstructing the underlying patterns that define the data.

The problem with only minimising reconstruction loss when training the network, is that it makes the latent space very discontinuous by giving the network too much freedom with how large, and sparse, the latent space can be (see [25] for plots comparing the latent structure of a VAE and standard autoencoder). With a VAE, the regulariser in Equation (2.3) ensures the posterior distributions (one for each input in the data-set) remain close to the prior. This is because we are minimising the the K-L divergence between the approximate posterior and the prior, which prevents the sections of the latent space dedicated to each class being far apart from each other, effectively making the space smooth. Without the regulariser it is not possible to generate data which smoothly transitions from one class to another (as seen in Figure 1.3).

# Chapter 3

# Gaussian VAE implementation

## 3.1 Reparameterisation trick

In neural networks the weights and biases associated with each layer are iteratively updated during training. This is done using gradient descent, the network parameters are updated in a direction that minimises the loss function. In a VAE the loss function involves computing and expectation over the distribution Q with network parameters denoted by $\phi$. Halfway through the network we sample a latent vector z from the encoder distribution, however the we cannot backpropagate through random sampling from $Q$, with which we take our expectation over, to update the weights [17]. Instead we sample an $\epsilon \sim \mathcal{N}(0, 1)$ from random noise of a known distribution and generate our latent sample through the transformation

$$z_\epsilon = \sigma_\phi * \epsilon + \mu_\phi \tag{3.1}$$

This differentiable equation for $z$ allows the network to take derivatives with respect to $\phi$, and thus backpropagate normally. We use the $z$ subscript to show that it is first determined (uniquely) by $\epsilon$.

## 3.2 How to generate new samples

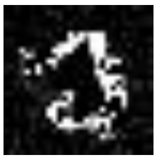


Figure 3.1: A new digit generated from training a simple VAE with the MNIST data-set composed of 60000 handwritten digits. This VAE had just one hidden layer in the encoder and decoder, and was trained using 50 epochs. This example shows the characteristics of different digits blending together.

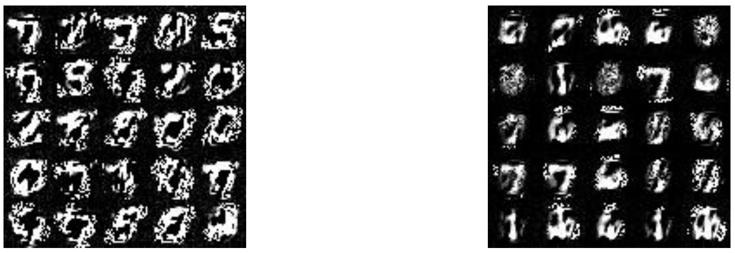
The (approximate) posterior $Q_\phi(z|x)$ is a distribution conditioned on an input $x$. It is used during training. Given an input, a sample from this distribution is passed to the decoder to attempt reconstruction of the input. Once training is complete, how do we generate new samples? We don't want to use the $Q_\phi$ since this would require an input $x$. Luckily, we can see from the K-L divergence term in our ELBO objective function that the approximate posterior is regularised so that it remains close to the unit Gaussian prior $p(z)$.

$$ELBO(\phi, \theta) = \mathbb{E}_{Q_\phi}(\log P_\theta(x|z)) - D_{KL}(Q_\phi(z|x)||p(z))$$

For this reason we can use the unit prior as our latent space. The trained decoder will interpret samples from this latent space and use the learned decoder distribution $p_\theta(x|z)$ to output new (random) images not in the training data. It is also possible to pass an input image $x^{(i)}$ through the trained encoder and sample a $z$ conditioned on this input from $Q_\phi(z|x^{(i)}$, to then illustrate the reconstruction differences. This is shown in Figure 3.2 using MNIST digit 5. At this point, the reconstruction is poor.

(a) Input digit x. (b) Reconstruction after 50 epochs.

Figure 3.2: Reconstruction when sampling from posterior



(a) One hidden layer (b) Three hidden layers

Figure 3.3: Comparing the observed output between using 1 or 3 hidden layers.

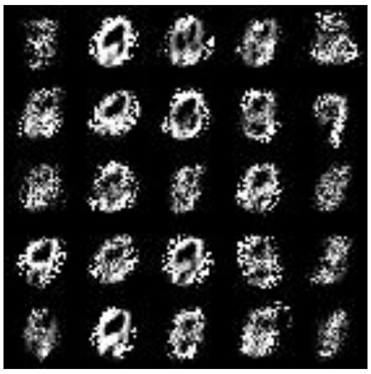


Figure 3.4: Three hidden layers. Latent dim 56

When implementing a VAE there are many ways to modify the network to try and improve results. The trivial way to improve results is to increase the number of training epochs (complete passes throught the data). However it is not possible to train the model indefinitely as the loss value will eventually hover around a local minimum. When working with the MNIST and Fashion MNIST data-sets, 50, 125, 250 and 400 epochs were tested and although some improvements can be seen, these are not significant for such data-sets. Another way may be to use more layers in the network. Figure 3.3 above show a comparison between using 1 hidden layer for both the encoder and decoder, and using 3 hidden layers for each. Using more hidden layers means more parameters are involved, in theory increasing the level of detail in the model. As shown, there is indeed a minor improvement when using the 3-layer model. The images are smoother, but still too abstract; the resulting images don't all look like digits. In both cases training ran for 250 epochs, with a latent dimension of 6. Figure 3.4 below shows the same 3-layer model but with a latent dimension of 56. In all such cases however, the resulting images are rather noisy, with clear shapes and patterns rarely identifiable. To improve results further we need to rethink the distributions involved in our model.

## 3.3 Improvements to the decoder output

Thus far the output distribution from the decoder, the likelihood $P_\theta(x|z)$, has been Gaussian. This allows us to follow the standard variational inference process to derive the loss function. However, in the case of images a Gaussian is not the most natural distribution. As an example in the case of pure black and white images, a Bernoulli output would be desired, because the pixel values are only 0 or 1. We are currently working with 8-bit images and there are methods that can be employed to fit this.

### 3.3.1 Issue with the Gaussian likelihood for images

As a side-note, when training images are loaded into the neural network from file and converted to vector form (shape 784x1), all the 8-bit values are normalised to a number between 0 and 1. Thus far a Gaussian output has worked since, following training, a sample from $P_\theta(x|z)$ will also likely be a number between 0 and 1 close to its mean. This can then easily be de-normalised back to 8-bit grayscale for display purposes. However, samples from a Gaussian distribution *can* take values across the whole real line, leading to nonsensical outputs.

### 3.3.2 Improved output using the Sigmoid function

In this section, the decoder is modified to directly output a reconstructed image vector with values strictly in the interval $[0, 1]$. In terms of the network, the decoder will not have separate layers for the mean and variance vectors, but instead one layer - the reconstructed image vector (784 dimensions for the standard 28x28 MNIST image). To ensure this vector takes values in $[0, 1]$ we use the sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

which maps any real value $x$ to $\sigma(x) \in (0, 1)$.

### 3.3.3 Updating the loss function

The loss function needs to be modified. The only affected term is the log-likelihood $\log p(x|z)$ since the encoder and the prior don't change under this updated model. During training, this term ensures the reconstruction loss is minimised. So this term is small when the input image $x$ and the reconstruction $x'$ are very similar. The same will be true now.

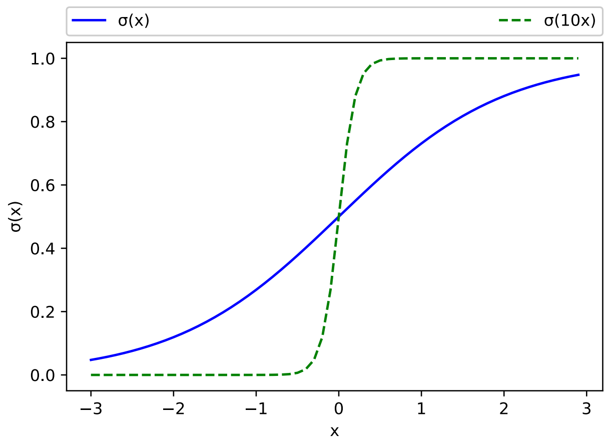


Figure 3.5: The sigmoid function. A steeper sigmoid function also shown.

Mathematically, the values from each of the dimensions of our input image $x$ can be treated as probabilities (they are between 0 and 1 as mentioned in section 3.3.1). These probabilities will be denoted by $p_i$ for all dimensions indexed by $i$. Similarly, the output probabilities from the decoder will be denoted by $q_i$. So we have input image vector

$$x = \begin{pmatrix} p_1 \\ p_2 \\ \dots \\ p_n \end{pmatrix}$$

and output reconstruction vector

$$x' = \begin{pmatrix} q_1 \\ q_2 \\ \dots \\ q_n \end{pmatrix}$$

Since these values are treated as probabilities, we can minimise the reconstruction loss by minimising the cross-entropy $H(p_i, q_i)$ between $p_i$ and $q_i$ for each dimension $i$.

$$\begin{aligned} H(p_i, q_i) &= -\mathbb{E}_{p_i}[\log q_i] \\ &= -(p_i \log q_i + (1 - p_i) \log (1 - q_i)) \\ &= -\mathbb{E}_{p_i}[\log \text{Bernoulli}(q_i)] \end{aligned} \tag{3.3}$$

This value is minimum when $p_i$ and $q_i$ are equal. To further illustrate what this cross-entropy is measuring we can re-write the above expectation as a K-L Divergence [6].

$$H(p_i, q_i) = H(p_i) + D_{KL}(p_i || q_i)$$

The entropy $H(p_i)$ is fixed, so minimising the cross-entropy is effectively minimising the K-L Divergence between $p_i$ and $q_i$, forcing the reconstruction to remain close to its original value.
Looking closer at the appearance of the Bernoulli($q_i$) distribution, value 1 is sampled with probability $q_i$ and value 0 with probability $1 - q_i$. We are not sampling 0s or 1s, the values we are interested in are the probabilities themselves. Despite this, the Bernoulli notation is useful as it simplifies implementation. We can now call the pre-built Pytorch function Bernoulli($q_i$).logprob($p_i$) in our ELBO objective function to be maximised. Note that maximising the Bernoulli log probability is equivalent to minimising the cross-entropy due to the minus sign in Equation (3.3).

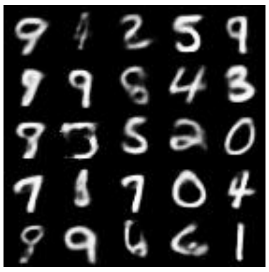
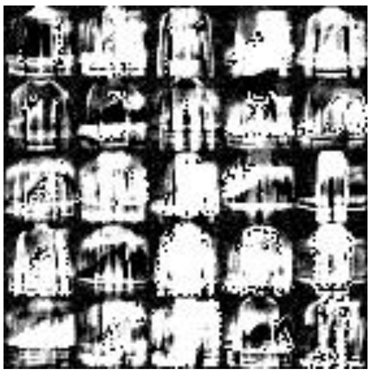


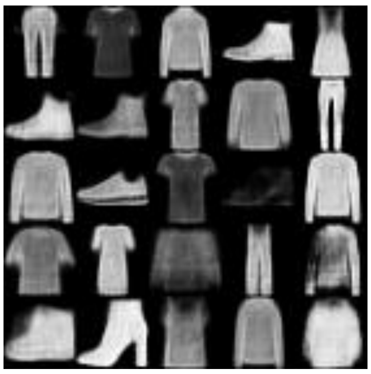
Figure 3.6: Digits generated from the updated model with a sigmoid output from the decoder.

### 3.3.4 Results

The image Figure 3.6 was generated from a simple 3-layer 50 epoch model. And yet, it can clearly be seen that allowing for less randomness in the model leads to less noisy ouput images. Indeed, most of the images could have come from real human handwritten digits, unlike the more random and abstract Gaussian output. The improvements are also very clear when training our network on the Fashion MNIST data-set, which contains 60000 28x28 gray-scale clothing images. These images include t-shirts, boots, jumpers, trousers and more. For examples of digit reconstructions with the update model (similar to what



(a) Gaussian model



(b) Updated model

we see in Figure 3.2), see Figure A.2 in the Appendix.

# Chapter 4

# Continuous relaxations of Bernoulli random variables

We are aiming to have a model which can approximate a Bernoulli posterior. This chapter explores the standard approaches to perform continuous relaxations of discrete Bernoulli variables.

## 4.1 A start on obtaining a Bernoulli latent distribution

Now we have a working continuous model, it is time to consider a discrete latent space. It may be beneficial to have, for instance, a binary latent space. That is, a sample from the n-dimensional latent distribution would be a vector $\in \{0, 1\}^n$. Such a distribution would be Bernoulli($\boldsymbol{\lambda}$), where $\boldsymbol{\lambda}$ is the vector of probabilities $\lambda_i$, where $\lambda_i$ is the probability of a sample having value 1 on the $i$-th dimension.
The trouble with a discrete latent distribution is that it is not possible to backpropagate, as we cannot do reparameterised sampling that is continuous. During training the decoder must sample from the latent distribution to work out the loss, and in the context of neural networks this requires the reparametrisation trick for gradients to be computed and backpropagated correctly. As seen in the previous section the trick required sampling $\epsilon \sim \mathcal{N}(0, 1)$ and transforming this to obtain a Gaussian latent sample $z$. For a Bernoulli random variable $z$, reparameterisation would first require sampling ramdomly from a Uniform(0,1) distribution, then applying a threshold function to this sample. As seen in Figure 4.1, the threshold function is a step function, where the gradient is either zero or infinite. This is clearly non-differentiable and discontinuous, preventing backpropagation from working. It seems necessary to approximate the discrete latent distribution with another, continuous distribution, that can be used for training the VAE.

A simple approach to approximating a discrete Bernoulli latent distribution is as follows:

- During training, sample reparametrised latent vector $z$ from the Gaussian approximate posterior $Q_\phi(z|x)$.

- Pass this sample through a steep sigmoid function $\sigma(\beta z)$. The output will be a vector with values mostly close to 0 or 1, approximating Bernoulli values.

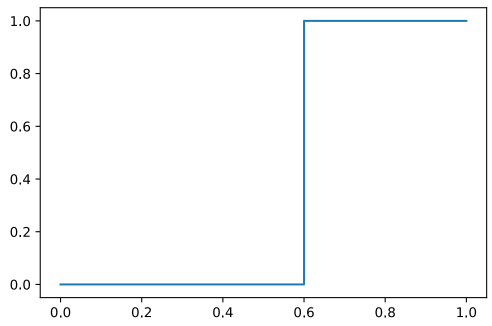
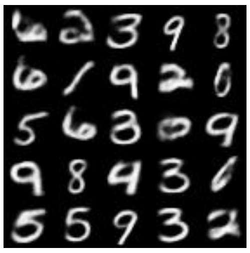


Figure 4.1: Example of the threshold (step) function for a Bernoulli(0.4) random variable, since 40% of U(0,1) random samples will have value 1 under this function.

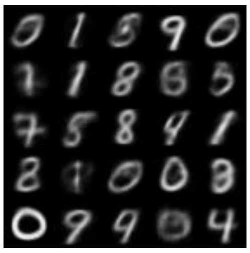
(Similar relaxation method used in [2].)
$z$ was reparametrised using a (continuous) Normal distribution and the sigmoid function is fully differentiable with non-zero and non-infinite gradients, which means backpropagation can be now be performed. The loss function does not need to be updated since no changes have been made to the distributions involved.

The parameter $\beta$ is called the inverse-temperature and determines the steepness of the sigmoid function. As $\beta$ tends to infinity, $\sigma(\beta z)$ tends to the step function centred at the origin. A key point to note: with the sigmoid function sufficiently steep, we assume that $\sigma(\beta z) \approx 1$ for $z > 0$ and $\sigma(\beta z) \approx 0$ for $z < 0$. This means the probability that $\sigma(\beta z) \approx 1$ for example is determined by the approximate posterior.

When training the VAE, the sigmoid function needs to be sufficiently steep to approximate a Bernoulli sample, but not so steep as to prevent the network from being optimised properly, since the gradients would almost always be very close to zero or, more rarely, very large at the origin. As seen in Figure 4.2 training using the inverse-temperature $\beta = 1000$ leads to slightly worse generated digits than using $\beta = 10$, even though the former more closely approximates Bernoulli samples.



(a) Steepness $\beta = 10$



(b) Very steep $\beta = 1000$

Figure 4.2: 56-dimensional binary latent space (Bernoulli(0.5)). 50 epochs for training.

As mentioned previously the approximate posterior is normalised during training to remain close to the prior $p(z) \sim \mathcal{N}(0, 1)$, which is used as the latent distribution during testing to generate new samples. This normalisation still occurs now, but during training we are pushing the samples through our steep sigmoid function before being decoded. Both the prior distribution and sigmoid function are centered at the origin, so a sample from the prior has equal probability of being close to 0 or close to 1 (there are some samples that get mapped directly in-between 0 and 1, but we assume these to be very few as the sigmoid as the is steep). This means we have approximated the discrete latent distribution Bernoulli(0.5). Hence, during testing we can just pass vector samples from Bernoulli(0.5) to the decoder to produce an output.

## 4.2 The Logistic model

Continuous relaxations of Bernoulli latent variables were introduced in the previous section using the Gaussian distribution. However, we will see it makes more sense to do this using a Logistic distribution. The Logistic distribution is defined to have probability density function [4]

$$f(z;\mu,s) = \frac{1}{4s}\text{sech}^2(\frac{z-\mu}{2s}) \tag{4.1}$$

where $\mu$ denotes the mean (or the location of the peak stationary point), $s$ denotes the scale which is proportional to the standard deviation and sech is the hyperbolic secant function (see Appendix A.1). Importantly the cumulative density function (CDF)

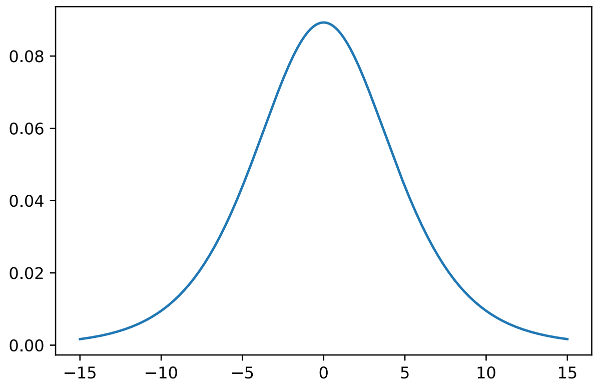


Figure 4.3: Direct plot of the Logistic PDF $f(z;\mu=0,s=2.8)$

$$\begin{aligned} F(x;\mu,s) &= Pr(z \leq x) \\ &= \int_{-\infty}^{x} f(z;\mu,s)dz \\ &= \frac{1}{1+e^{-(x-\mu)/s}} \\ &= \sigma(\frac{x-\mu}{s}) \end{aligned} \tag{4.2}$$

is a sigmoid function! Although the logistic distribution looks very similar to the Gaussian (see Figure 4.3), albeit with a heavier tail, it has a closed form CDF (unlike the Gaussian).

We will use this distribution during training to approximate a Bernoulli($\boldsymbol{\lambda}$) posterior distribution (which itself is an approximation of the true posterior) as before. In this model let

$$Q_\phi(z|x^{(i)}) \sim \text{Logistic}(\mu_\phi(x^{(i)}),1)$$

be the approximate posterior and set

$$p(z) \sim \text{Logistic}(0,1)$$

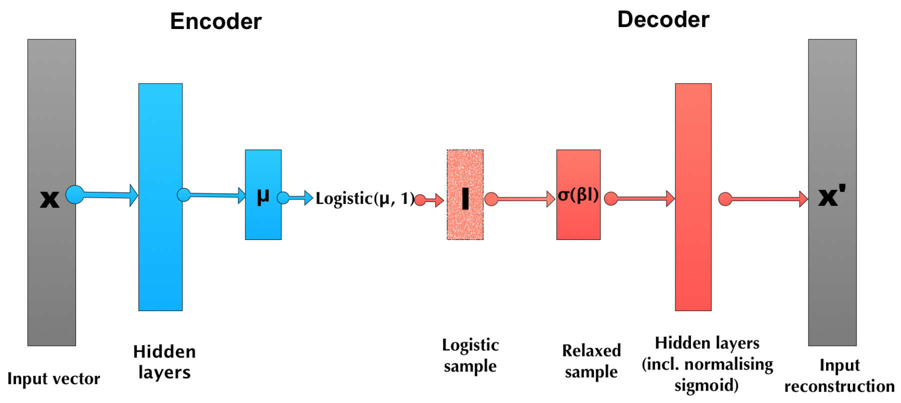


Figure 4.4: Current VAE neural network structure.

as the prior. We will continue writing $\mu_\phi$ instead of $\mu_\phi(x^{(i)})$ for simplicity. Let $z_d$ denote the discrete random variable associated with the Bernoulli($\boldsymbol{\lambda}$) distribution and let $z_r$ denote the continuous relaxation of $z_d$. We use $l$ $\sim$Logistic$(\mu_\phi, 1)$ to denote the latent (approximate posterior) sample and we have $z_r = \sigma(\beta l)$.

It should be clear that

$$z_d = \lim_{\beta \to \infty} z_r$$

since at this limit, the sigmoid function is the step function, so takes value strictly 1 or 0. During training how is the $\boldsymbol{\lambda}$ parameter determined and trained? Let's consider the probability $\lambda_i$ that $z_d$ has value 1 on the i-th dimension. $z_{r_i} = \sigma(\beta l_i) \approx 1$ occurs when $l_i > 0$, where $l_i$ is the value in the $i$-th dimension of $l$. This means $\lambda_i = Pr(l_i > 0)$, which we can directly work-out, and train, using the closed-form CDF.

$$\begin{aligned} \lambda_i &= 1 - Pr(l_i < 0) \\ &= 1 - F(0; \mu_{\phi i}, 1) \\ &= 1 - \frac{1}{1 + e^{\mu_{\phi i}}} \\ &= \frac{1}{1 + e^{-\mu_{\phi i}}} \end{aligned} \tag{4.3}$$

We deduce simply

$$\boldsymbol{\lambda} = \sigma(\mu_\phi) \tag{4.4}$$

So using a Logistic distribution with scale 1 during training means our posterior takes the form Bernoulli($\sigma(\mu_\phi)$), where the mean has subscript $\phi$ to show that it is learned by the encoder network with weights and biases denoted by $\phi$.

With the Gaussian model we couldn't express this Bernoulli posterior parameter $\boldsymbol{\lambda}$ as a simple function of the encoder output, instead the probabilities are a more complicated (no closed-form CDF) and numerically unstable function of $\mu_\phi$. So although both distributions lead to similar results visually, the simpler Logistic equations make it easier to prove and note results, benefits which will be useful in the upcoming chapters. The prior logistic has mean 0, so our latent distribution to generate new samples from will be Bernoulli(0.5) since $\sigma(0) = 0.5$.

From Figure 4.5, we can see that even in a discrete space our network is able to give a solid reconstruction of MNIST digits in many cases. There are a few cases where this reconstruction breaks and the wrong digit is output, which will be briefly discussed in the conclusion (Chapter 6). The reconstruction is blurry, but this is more of an issue with VAEs themselves and how reconstruction loss is computed being not the most ideal for images [8] [31], rather than the Logistic model being at fault.

(a) Original digit (b) Trained reconstruction from $z_r$

Figure 4.5: Taking digit 0 as input and reconstructing with the decoder from the relaxed latent variable. We see that reconstructions are still accurate in some cases.

### 4.2.1 Reparameterised sampling for the Logistic Distribution

When training our VAE, the Logistic samples $l$ need to be reparameterised. This means first sampling from random noise, which does not depend on the model parameters, and then transforming this sample to generate $l$ (see Chapter 3.1). This reparameterisation will be done using the inverse cumulative distribution (inverse-CDF) method [26], a common way to generate exact samples of any distribution with a tractable inverse-CDF. Recall for a continuous distribution defined over the whole real line we have the CDF

$$F : \mathbb{R} \mapsto (0,1) \text{ where } F(x) = Pr(l < x) = u \tag{4.5}$$

is a one-to-one mapping. The inverse CDF

$$F^{-1} : (0,1) \mapsto \mathbb{R} \text{ where } F^{-1}(u) = \min\{l : F(l) > u\} \tag{4.6}$$

is also a one-to-one mapping. For this sampling method, we take a random sample $u \sim \text{Unif}(0,1)$ which lies in the required interval and pass it through $F^{-1}$ to output a sample from the original distribution whose CDF is F. This is reparameterised sampling, since the Uniform distribution does not depend on our model parameters and the required transformation to obtain $l$ is done using the inverse-CDF.
For the Logistic distribution, the CDF is the sigmoid function which is easy to invert to obtain the inverse-CDF

$$l_u = F^{-1}(u; \mu, s) = \mu + s \log\left(\frac{u}{1-u}\right) \tag{4.7}$$

## 4.3 The Concrete model

An alternative, but similar way of obtaining relaxed latent samples is as a special case of the Concrete distribution [19], proposed in 2016. There, the Bernoulli posterior parameter $\boldsymbol{\lambda}$ is controlled by an additional parameter $\gamma \in (0, \infty)$ in the sigmoid function itself, rather than the Logistic mean $\mu_\phi$. When using the Concrete-based approach we have $l \sim$Logistic$(0, 1)$ and

$$\begin{aligned} z_r &= \frac{1}{1 + \exp\left(-\beta(\log \gamma_\phi + l)\right)} \\ &= \sigma(\beta(\log \gamma_\phi + l)) \end{aligned} \tag{4.8}$$

where $\gamma_\phi$ is the trainable parameter. The $\log(\gamma)$ term controls how the sigmoid function is translated along the horizontal $l$-axis; increasing $\gamma$ translates the curve to the left and decreasing $\gamma$ translates it to the right. This means a larger value of $\gamma$ increases the probability that the relaxed sample has value 1, in a similar way that having a larger value of $\mu$ has this same effect under the Logistic model. Under this approach we have $\boldsymbol{\lambda} = \frac{\gamma_\phi}{1+\gamma_\phi}$. The Concrete model has benefits in that it can be adapted to relax many types of discrete random variables (not just Bernoulli). Unfortunately in the Bernoulli special case, this approach relies on initially sampling from the standard Logistic distribution which cannot produce optimal continuous relaxations in the context of training a VAE. Stronger relaxations will require work on the underlying sampling distribution, see Chapter 5. For consistency with further work, we will stick with with the standard steep sigmoid $\sigma(\beta l)$ approach for relaxations where $\boldsymbol{\lambda}$ is determined by the parameters, of the underlying sampling distribution rather than an additional sigmoidal parameter.

## 4.4 Problem with the ELBO

As seen in Chapter 2.2 on the loss function, during training the goal of the VAE is to maximise the Evidence Lower BOund (ELBO) function. The "bound" is on the log probability of the data, and it is also referred to as the "variational bound". An optimal bound occurs when the approximate posterior $Q(z|x)$ is as "close" as possible to the true posterior $p(z|x)$, where we used the Kullback-Leibler divergence to quantify how much the the two distributions differed. Since the true posterior is intractable we cannot have the two distributions be equal, so there are some sacrifices with using the approximate posterior. If we are working in a discrete space the true posterior will be a complex intractable discrete distribution, analogous to the true posterior in a continuous space.

We are currently approximating a binary (Bernoulli) latent space using a Logistic approximate posterior combined with a steep sigmoid function applied to the logistic samples before being decoded. The ELBO objective function reflects this; we have set a Logistic prior mean 0 and Logistic (approximate) posterior with mean $\mu_\phi(x^{(i)})$, and the reconstruction loss term ensures the relaxed posterior samples resemble the input $x^{(i)}$ when decoded.
The current ELBO posterior log probabilities are computed as

$$\log p(z) = \log \frac{1}{4}\text{sech}^2(\frac{l}{2}) \tag{4.9}$$

$$\log Q_\phi(z|x^{(i)}) = \log \frac{1}{4}\text{sech}^2(\frac{l - \mu_\phi(x^{(i)})}{2}) \tag{4.10}$$

(log likelihood term omitted as it remains unchanged from Chapter 3.3)
where $l$ is a posterior $Q_\phi$ sample. As seen in the previous section, we approximate a Bernoulli(0.5) prior and a Bernoulli($\sigma(\mu_\phi(x^{(i)})$)) posterior. Why not directly write the ELBO function in terms of these Bernoulli distributions? The network would have to keep passing relaxed Logistic samples $z_r$ to the decoder (rather than actual Bernoulli samples

$z_d$) due to the backpropagation issue, but the network parameters can be optimised using the new ELBO. Instead of optimising over the Logistic distribution which is then used to approximate the Bernoulli, we directly optimise over the Bernoulli latent distribution. In theory if the approximation is strong, the ELBO function should converge to a similar value in both cases. However this is not the case.

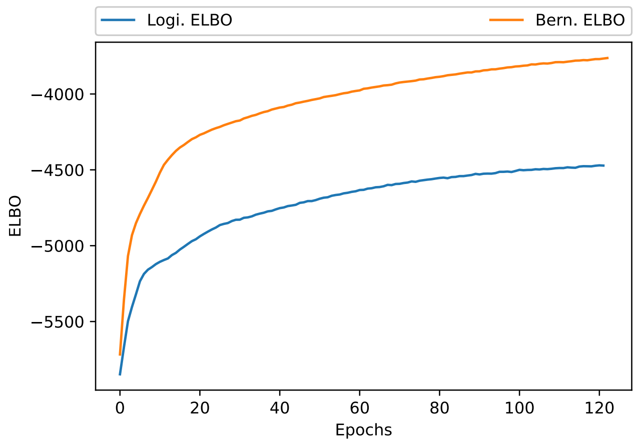


Figure 4.6: Logistic ELBO (blue). Bernoulli ELBO (orange). The ELBO value at each epoch represents an average over the whole MNIST data-set. Higher is better.

As seen in Figure 4.6, the Bernoulli ELBO gives a better bound over the Logistic ELBO in every case. Recall these are both bounds on the log evidence of the data $\log p(x)$ and higher values are better. However there is still a problem with optimising over this Bernoulli ELBO.
The new log probabilities under the discrete ELBO are

$$\log p(z) = z_r \log 0.5 + (1 - z_r) \log 0.5 \tag{4.11}$$

$$\log Q_\phi(z|x^{(i)}) = z_r \log \sigma(\mu_\phi) + (1 - z_r) \log (1 - \sigma(\mu_\phi)) \tag{4.12}$$

(log likelihood term omitted as it remains unchanged from Chapter 3.3)
The log probability equalities follow from the probability mass function of a Bernoulli($\lambda$) distribution,

$$p(z|\lambda) = \lambda^z (1 - \lambda)^{1-z} \tag{4.13}$$

where $z \in \{0, 1\}$. This highlights an immediate issue with our log probabilities, the relaxed sample $z_r \notin \{0, 1\}$. These ELBO equations are expecting $z_d$ instead, which is why we cannot obtain a valid (posterior) bound (inspired by Aitchison et al. [2]).

Using the Logistic ELBO we are directly optimising the bound, that is we are directly finding the required vector $\mu_\phi$ using gradient descent which optimise the correct posterior log probabilities (Equation (4.11, 4.12)). However, as seen, optimising the Logistic($\mu_\phi$, 1) is far from the strongest approximation of a Bernoulli. Even though using the Bernoulli ELBO instead give a stronger bound, it is not the correct bound that is being optimised. It would be ideal to combine the benefits of both ELBO cases, that is optimising the correct bound whilst obtaining stronger results. This requires finding an alternative approximate posterior. Moving towards this goal we will begin by finding the true posterior in a simplified single-dimension model, see Chapter 5.1. First we observe these phenomena experimentally.

## 4.5 Looking at the true bound

We cannot optimise the network using true discrete samples $z_d$ in the Bernoulli ELBO. However we can build a separate function which records ELBO values passing $z_d \sim \text{Bernoulli}(\sigma(\mu_\phi))$ to the log probability functions. $z_d$ is also passed to the decoder so that the log likelihood term in this discrete ELBO is correct. We are still optimising over $z_r$, but call this new function every time we call the usual loss function to compare bounds. We refer to the discrete ELBO as the "true bound" as it represents the objective (loss) function of an actual Bernoulli model, but this is not the same as the variational bound (log probability of the data, Chapter 2.2) since that can only be reached with the intractable true posterior.

Let's first illustrate in Figure 4.7 the difference in the bounds when optimising using the Logistic ELBO. We observe two interesting phenomena

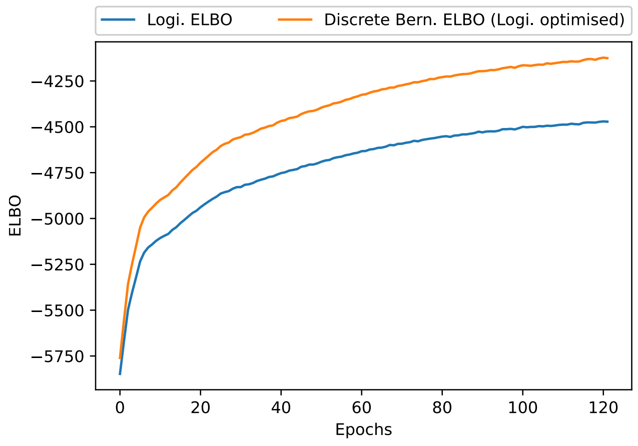


Figure 4.7: Optimising using Logistic ELBO and comparing with recorded values of the discrete Bernoulli ELBO.

- The discrete Bernoulli ELBO (the "true bound") shows a much stronger bound than than Logistic ELBO, despite the fact that we are optimising the latter.
- As we optimise the Logistic ELBO, the discrete ELBO appears to follow the same rate of optimisation.

This is experimental indication that the Logisitic ELBO is indeed optimising the correct bound. However the drawback here is that we are unable to get close to this true bound using this network; the disparity between the two curves is too strong.

Now looking at the difference in the bounds when optimising using the Bernoulli ELBO, in Figure 4.8.

The values are quite close, and so the graph shows only the last 25 epochs of training, effectively "zooming in". However it is still odd that using the relaxed samples appears to give a stronger bound than using correct discrete samples. This phenomenon highlights a problem with not training the correct bound.

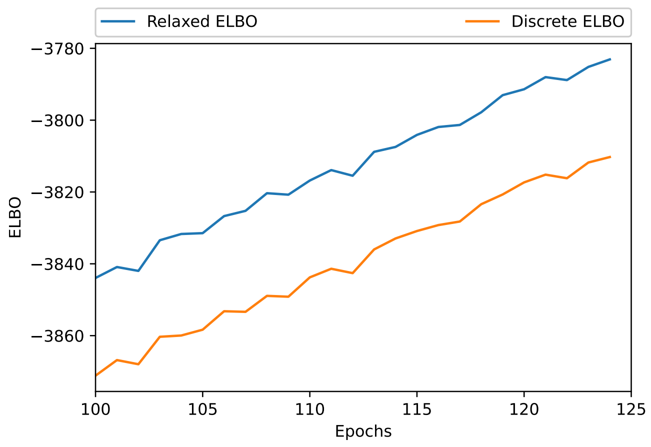


Figure 4.8: Optimising using the Bernoulli ELBO with relaxed samples and comparing with recorded value using discrete samples. Values for epochs 100 to 125.

# Chapter 5

# Optimal Bernoulli continuous relaxations

The quality of the continuous relaxation is directly linked to the approximate posterior, since it determines the Bernoulli probabilities we want when samples are relaxed using the steep sigmoid function. One of the main challenges in finding a suitable approximate posterior is we don't know the shape of the true posterior. In this chapter we attempt to find such a distribution, with a suitable shape more closely resembling the underlying true posterior. Insight into what that may look like comes from a simplified model.

## 5.1 Simple toy-model

We now consider a simple single-dimension model, where the true posterior is tractable.

In the previous section we used the relaxed latent variable $z_r$, which has a value in the interval [0,1], most often close to 0 or to 1 to approximate discrete Bernoulli samples $z_d$. We can create strict $\{0, 1\}$ samples by thresholding Logistic samples which we will just call $z$ in this section. For our input data $x$, we believe it to be generated by applying low-variance $(0.5^2)$ Gaussian noise to the Bernoulli samples. This is not an unrealistic assumption for real-world (observed) data generated by discrete processes ([10] Figure 1) even though it does not quite match how the MNIST data used to test our models was generated. These assumptions give rise to our likelihood distribution

$$p(x|z) \sim \mathcal{N}(\Theta(z), 0.5^2)$$

where $\Theta$ is the step function centred at the origin, which has value 1 if $z > 0$ and 0 if $z < 0$ (note that with continuous distributions the probability that $z = 0$ is zero). The prior - how we believe the latent variables to be distributed is

$$p(z) \sim \text{Logistic}(0, 1).$$

so that $\Theta(z) \sim \text{Bernoulli}(0.5)$ as a prior, since $z$ has equal probability of being greater than zero or less than zero. Using Baye's theorem [18] we can formulate the posterior as

$$f_{Z|X}(z|x) = \frac{f_{X|Z}(x|z) f_Z(z)}{f_X(x)} \tag{5.1}$$

where f denotes the respective probability density function. The evidence $f_X(x)$ for some x is a constant which normalises the posterior distribution so that it integrates to 1 over

the real line. It is tricky to compute so for now we will ignore it as the general form of the posterior is obtained from the numerator.
Conditioning on $x = 1$ as an example we obtain

$$f_{Z|X}(z|1) \propto \frac{e^{-2(1-\Theta(z))^2}\text{sech}^2(\frac{z}{2})}{2\sqrt{(2\pi)}} \tag{5.2}$$

Recall that we take samples from the approximate posterior and then relax them before being decoded. So even though this distribution is almost all continuous (except for the step), what matters most is whether $z$ is positive or negative, not its exact value. We see this density function places much larger probability mass (area under curve) over $z > 0$ compared with $z < 0$, when conditioning on $x = 1$. This makes sense as the relaxed $z_r = \sigma(\beta z) \approx 1$ when $z > 0$.
As expected when conditioning on $x = 0$, we obtain the same plot but reflected in $z = 0$ so that most mass is over $z < 0$. The pdf is also shaped by the choice of variance for our likelihood, a larger variance makes it more likely for $\Theta(z)$ to deviate from the $x = 1$ conditioning, shifting some mass away from $z > 0$.

How well does a $Q(z|x = 1) \sim \text{Logistic}(\mu(x = 1), 1)$ approximate this posterior? We can apply the variational inference method trained by a neural network on this simple model. This network has only single dimension hidden layers and the only data supplied is $x = 1$. Other than that everything is the same as the Logistic model. Figure 5.1 displays the learned logistic distribution when trained for 400 iterations (same as 400 epochs since we are using a single-element dataset). We see it places much more probability mass over $z < 0$ than the true posterior, which would lead to many samples giving an incorrect relaxed output to be decoded.

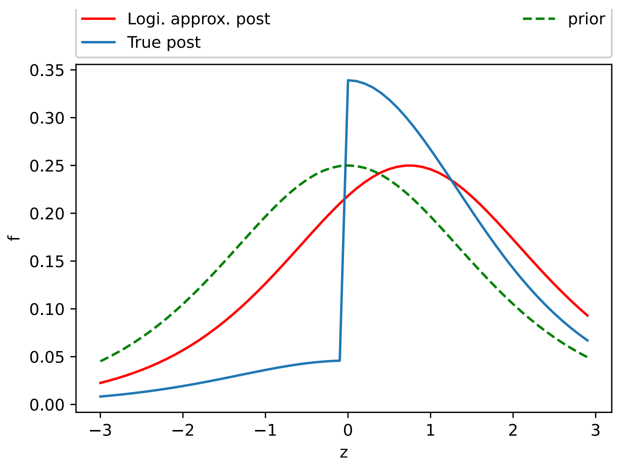


Figure 5.1: Comparing Logistic approximate posterior with the true posterior. Trained for 400 iterations with data $x = 1$.

If the main goal is to place the majority of probability mass over either $z > 0$ or $z < 0$ depending on the input, why is it not possible to just shift the Logistic means into large positive or negative values respectively? This would make the relaxed samples very predictable, effectively being able to closely match the input data the posterior is conditioned on. This type behaviour is exhibited in a standard autoencoder, who's only goal is to recreate the input as closely as possible. In a VAE the posterior is regularised so that it remains close to the prior (the Logistic(0,1) in this case), preventing means

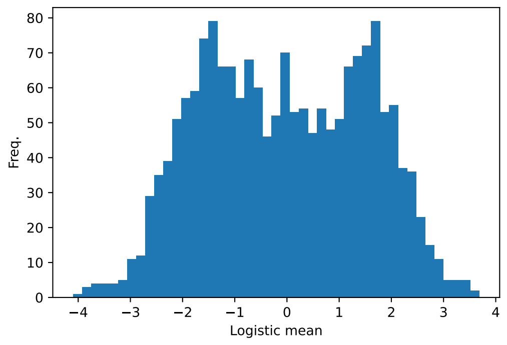


Figure 5.2: Logistic posterior means for a batch of 32 MNIST images. Latent dimension is 56, so we have a total of 1792 means split across 45 bins. Network trained for 125 epochs, which is sufficient as can be see with nearly-converged ELBO in Figure 4.6.

from deviating too far. This has benefits in allowing us to generate new data which doesn't exist in the data-set, but means to posteriors lose some degree of freedom. As an illustration, Figure 5.2 shows a histogram of the posterior means for a batch of 32 images trained on the Logistic model. Indeed, few means are greater than 2 in magnitude.

## 5.2 The StepLogistic distribution

The true posterior for our simple toy-model in the previous section can help solve the problem of finding a better approximate posterior than the Logistic distribution for use in approximating a binary latent space. This posterior has a step which better splits the probability mass between $z < 0$ and $z > 0$. We will not directly use Equation (5.2) as the required density function, what matters more is the shape which resembles a Logistic(0, 1) prior split down the middle where a step is added. A Gaussian distribution could have been used instead, as it boasts close resemblance to a Logistic but lacks the closed form cumulative density function which will become useful when writing and proving equations.

In creating our approximate posterior, the step can be achieved by scaling the area of each half separately. We can't scale using arbitrary numbers since the resulting probability density function must still integrate to 1. To work out what values these scalars can take we start with the fact that each Logistic(0, 1) half has area $\frac{1}{2}$ by symmetry. Taking $\alpha$ and $\beta$ to be our area scalars, we end up with the equation

$$\frac{\alpha}{2} + \frac{\beta}{2} = 1 \tag{5.3}$$

since the total area must remain 1 for a probability distribution. From this we deduce the scalars to be $\alpha$ and $2 - \alpha$, and they determine the relative height of the right and left Logistic half. This single parameter $\alpha$ will be learned by our network and be the output of the decoder, instead of the current mean $\mu$. The parameter is dependent on the input vector $x^{(i)}$ and scales each dimension independently so we may refer to it by the vector $\alpha_\phi(x^{(i)}) \in (0, 2)^n$, or just $\alpha_\phi$ for short. This approximate posterior shall hence be called the StepLogistic($\alpha_\phi$), with PDF

$$f(z; \alpha_\phi) = \frac{2 - \alpha_\phi}{4} \text{sech}^2\left(\frac{z}{2}\right) \Theta(-z) + \frac{\alpha_\phi}{4} \text{sech}^2\left(\frac{z}{2}\right) \Theta(z) \tag{5.4}$$

where $\Theta(-z)$ is the step function reflected in the vertical axis, equaling 1 for negative inputs and 0 for positive inputs.
Simplifying notation and considering a single dimension $i$, we have the following probabilities for our latent sample $z$ from scaling each Logistic half

$$Pr(z_i < 0) = \frac{2-\alpha_i}{2},\ Pr(z_i > 0) = \frac{\alpha_i}{2}$$

so that when relaxed our latent distribution approximates Bernoulli($\frac{\alpha_\phi}{2}$).

Let $Q_\phi(z|x) \sim \text{StepLogistic}(\alpha_\phi)$ represent the approximate posterior and $p(z) \sim \text{Logistic}(0,1)$ be the prior. Examining the K-L divergence regulariser between the approximate posterior and prior which forms part of the ELBO function as seen in Equation (2.3), it is revealed that

$$\begin{aligned} D_{KL}(Q_\phi(z|x)||p(z)) &= \frac{(2-\alpha_\phi)\log(2-\alpha_\phi) + \alpha_\phi\log(\alpha_\phi)}{2} \\ &= D_{KL}(\text{Bernoulli}(\frac{\alpha_\phi}{2})||\text{Bernoulli}(0.5)) \end{aligned} \tag{5.5}$$

this K-L divergence is the same as that for the Bernoulli posterior being approximated. Full derivation is in the Appendix A.3. It is directly evident that optimising the StepLogistic ELBO will optimise underlying Bernoulli ELBO. However the ELBO will be much tighter with the true bound when using the StepLogistic due to to Equation (5.5).
The exact regularising K-L divergence when using the Logistic($\mu_\phi$, 1) approximate posterior is difficult to calculate directly which is why we have to use a sampling-based approach to approximate this part of the ELBO (see Chapter 2.2). However, numerical approximations reveal this divergence can get much larger than the underlying Bernoulli divergence (Table 5.1). So using this updated distribution where we optimise the scaling parameter

| K-L div. | Logistic($\mu$,1) | Bernoulli($\sigma(\mu)$) |
|---|---|---|
| $\mu = 0$ | 0.00 | 0.00 |
| $\mu = 1$ | 0.16 | 0.11 |
| $\mu = 10$ | 8.00 | 0.69 |

Table 5.1: Comparing the K-L divergence between the approximate posterior and the prior for the Logistic model and true Bernoulli model (single dimension). Values are the same for positive and negative means. Logisitic K-L divergence values approximated using the *Desmos* numerical integration tool.

$\alpha_\phi$ instead of the mean $\mu_\phi$ should drastically reduce the distance between the two curves as seen in Figures 4.6 and 4.7.

### 5.2.1 Reparameterised sampling for the StepLogistic distribution

If we want to train our model using this distribution we need to be able to sample from it, as is the case with all VAE models.

For the StepLogistic distribution we need to separate the CDF function for when $z$ is negative or positive, due to the difference in the relative height of the PDF for these two cases. When $z < 0$ the CDF function is easy to derive, simply weight the Logistic

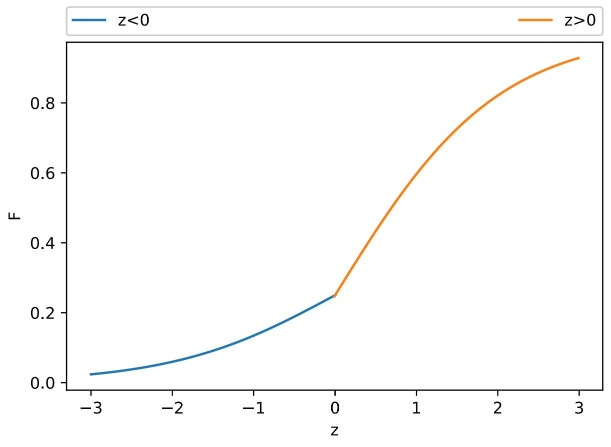


Figure 5.3: The StepLogistic cumulative distribution function when $\alpha = 1.5$.

CDF by the area scale factor $2 - \alpha$. The area when $z$ is positive requires integration, the derivation is included in the Appendix A.2. The results are:

$$F(z;\alpha) = \begin{cases} (2-\alpha)\sigma(z) & : z < 0 \\ \frac{2-\alpha+\alpha\tanh(\frac{z}{2})}{2} & : z > 0 \end{cases} \tag{5.6}$$

As an example when $\alpha = 1.5$ this is what the overall StepLogistic CDF looks like:

This value of $\alpha$ places more probability mass over positive values, the CDF illustrates this by the positive curve having larger area than the negative curve for any fixed interval size.

Next we must compute the inverse CDF functions. This is simple as we can directly invert the expressions in Equation (5.6). We obtain

$$F^{-1}(u;\alpha) = \begin{cases} \log\frac{u}{2-\alpha-u} & : u < \frac{2-\alpha}{2} \\ \log\frac{\alpha+u-1}{1-u} & : u > \frac{2-\alpha}{2} \end{cases} \tag{5.7}$$

where $u \sim \text{Unif}(0,1)$. In practise, we sample $u$ from a slightly smaller interval such as $[0 + 1e - 6, 1 - 1e - 6]$, since when using a computer there is a non-zero probability of sampling value 1 or 0 which would make the expressions in Equation (5.7) undefined. This cannot happen in a purely theoretical model.

For some visual confirmation that these equations are working, they can be used to create samples and we can plot these in a histogram. In a histogram the relative frequency of the samples in a bin is (roughly) directly proportional to the probability of that bin when the number of samples is large enough. This means we can use histograms to approximate the shape of PDF curves. In Figure 5.4 I make 100000 samples using the inverse CDF method using $\alpha = 0.5$, and compare my histogram to the actual StepLogistic($\alpha = 0.5$) PDF.

## 5.3 Implementation and results

Unfortunately the StepLogistic model can be rather numerically unstable when it comes to training. Fully understanding why requires careful analysis of the gradients being computed to train the model, which is left as future work. This makes obtaining data

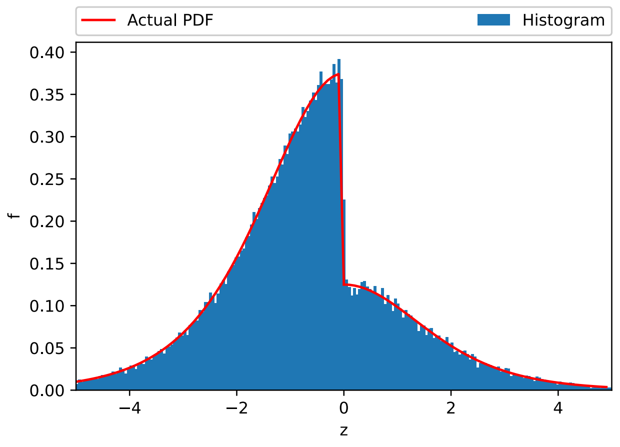


Figure 5.4: Comparing (normalised) histogram of samples with the actual PDF of a StepLogistic($\alpha = 0.5$) distribution.

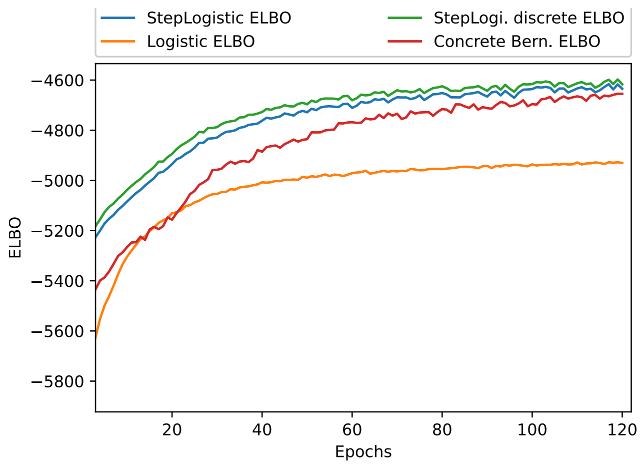
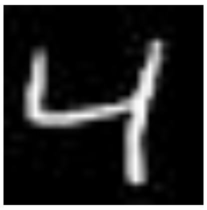
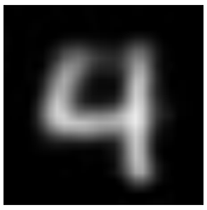


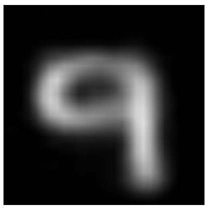
Figure 5.5: Comparing the ELBO of the StepLogistic and Logistic models. Logistic and Concrete models trained using the Bernoulli ELBO are also shown. Training was for 125 epochs and the plot starts from epoch 3.

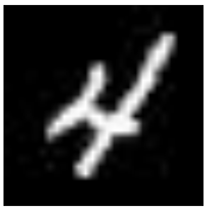
difficult for certain model configurations. Reducing the latent distribution dimension from the usual 56 used thus far to a simpler 8 dimensions helps with numerical stability issues and we are able to train our model. Thus all the following models are trained with an 8-dimensional latent space for comparison. We can see from Figure 5.5 that the ELBO is much stronger under the new StepLogistic model (in blue) than Logistic model (in orange). This should come as no surprise since the StepLogistic regulariser from Equations (2.3, 5.5) is the same as that of the underlying Bernoulli distribution, enabling this much tighter bound with the discrete (true) bound (in green). The main difference between the blue and green curves comes from the log likelihood $\log P_\theta(x|z_r)$ which is conditioned on the ”soft” relaxed posterior samples $z_r$ in the StepLogistic ELBO, instead of the hard Bernoulli samples $z_d$ it is expecting which are present in the discrete ELBO.

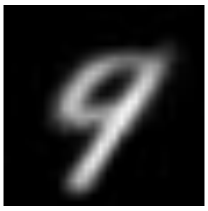
The StepLogistic ELBO is even stronger than the Concrete model (Chapter 4.3) directly optimised using the Bernoulli ELBO, which we saw (Chapter 4.4) gives strong results but which are misleading due to the fact that the incorrect bound is being optimised. So it is pleasing to see our model gives even stronger results while optimising correctly, satisfying our goal from the end of Chapter 4.4. Applying these models to the Fashion MNIST data-set (28x28 grayscale images of clothing items) yields similar results, albeit with more unstable ELBO behaviour from the Concrete model, see Figure A.5 in the Appendix.


(a) Original 1


(b) StepLogistic


(c) Logistic


(d) Original 2


(e) StepLogistic

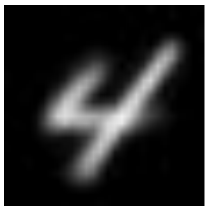

(f) Logistic

Figure 5.6: The StepLogistic and Logistic model attempt to reconstruct two versions of the digit "4".

The stronger performance exhibited by the StepLogistic approximate posterior when considering the ELBO does not translate to any significant visual or numerical improvements in the reconstruction output. In fact, the StepLogistic model occasionally performs worse than the Logistic one. To give examples of the mixed reconstruction performance, in Figure 5.6 we see two versions of the digit "4" and the reconstructed outputs using the StepLogistic and Logistic models. In each case, one of the models reconstructs the wrong digit from its latent samples, unable to convincingly discern between "4" and "9". This is likely attributed to the fact that such a small and binary latent space is unsuitable for the MNIST data set when it comes to accurately reconstructing digits, but suitable enough to compare ELBOs between models as in Figure 5.5. This is discussed in more detail in the Conclusion. Due to the random nature of sampling from the latent distribution, we must take repeated samples and work out an average reconstruction loss. The reconstructions in Figure 5.6 represent typical outputs. The loss values in 5.2 are in-line with what can

| Recons. loss | StepLogistic | Logistic |
|---|---|---|
| Orig. 1 | 201.7 | 208.0 |
| Orig. 2 | 134.0 | 119.0 |

Table 5.2: Reconstruction loss from recreating both "4" digits under our two models. Values come from averaging over 5 recreations.

be seen in Figure 5.6 and confirm the mixed reconstruction performance on the MNIST data-set.

## 5.4 Attempts to smooth the step

Considering scalar samples (but can be applied to each dimension of a vector separately), we made the assumption in Chapter 4.1 that, for the approximate posterior samples $z$, the relaxed $z_r \approx 1$ when $z > 0$ and $z_r \approx 0$ when $z < 0$. While this is true in most cases, the samples $z > 0$ or $z < 0$ very close to the origin will map to similar values under the (steep) sigmoid function, despite a very sudden and potentially large difference in probability mass around these values. The hard step in the StepLogistic makes more sense if we are thresholding our samples so that they take values in $\{0, 1\}$. This means the likelihood $P_\theta(x|z)$ is softer than it ought to be and a more natural posterior for such a likelihood is one with a smoother step.

Additionally, rather than providing a distribution that is broken down into two different cases for positive and negative latent samples using a step, leading to trickier implementation as to avoid any gradient computations on the step, it would be more convenient to provide a single function for the PDF, CDF and inverse-CDF. A smooth step may lead to a more numerically stable model as there wouldn't be such sudden large increase or decrease in probability mass at the origin.

However, introducing a smooth step in a numerically stable way so that the model can be trained by a deep neural network is far from trivial. Here I detail some approaches to obtain such a "SmoothStepLogistic" distribution and the challenges faced when attempting to implement them.

### 5.4.1 Approach 1: Smooth PDF

We can smooth PDF 5.4 by replacing the step functions with steep sigmoid functions as

$$\tilde{f}(z; \alpha, \beta) = \frac{2-\alpha}{4}\text{sech}^2\left(\frac{z}{2}\right)\sigma\left(-\beta z\right) + \frac{\alpha}{4}\text{sech}^2\left(\frac{z}{2}\right)\sigma\left(\beta z\right) \tag{5.8}$$

Using a sigmoid function ensures a smooth transition between the two cases as the contribution from the first term is small (tending to zero asymptotically) when $z$ positive and similarly a small contribution from the second term when $z$ is negative. Remarkably, we can show using symmetry that $\tilde{f}$ does indeed integrate to 1 over the continuous real line for any inverse-temperature (steepness) $\beta$, see Appendix A.4 for proof. This along with the fact that $\tilde{f}$ remains non-negative makes it a valid PDF [24].

Plotting $\tilde{f}$ using $\beta = 10$ reveals a suitable shape for the purpose of splitting probability mass between positive and negative inputs. The hurdle in implementing (5.8) as our density function comes from attempting the compute its CDF. Even though it is possible to compute $\int_{-x}^{x} \tilde{f} dz$ for any real $x$, the symmetry trick breaks down for $\mathrm{F}(x; \alpha, \beta) = \int_{-\infty}^{x} \tilde{f} dz$. While a general closed-form CDF may exist, finding it is no trivial task, and Wolfram Alpha's integration tool only appears to find a solution when $\beta = 1, 2, 3$. This means we currently can't use a closed-form CDF and inverse-CDF, which is problematic since these are necessary to draw samples from this distribution. Intractable PDFs are nothing new for probability distributions, indeed there is no closed form integral for the Gaussian PDF for example. However, Mathematicians have developed strong approximations for these functions for many distributions [15], or other (non-CDF) methods such as using the Central Limit Theorem to sample from Gaussian. No such luxury is bestowed on us yet for the StepLosgistic PDF. Instead, slow ingtegral approximations such as the Trapezium rule have to be used.

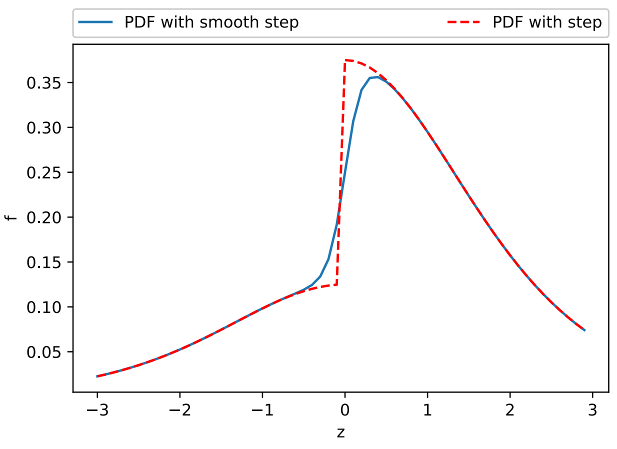


Figure 5.7: "SmoothStepLogistic" PDF from smoothing the StepLogistic PDF. In this example $\alpha = 1.5$ and $\beta = 10$. This is compared with the orginal StepLogistic($\alpha = 1.5$) PDF.

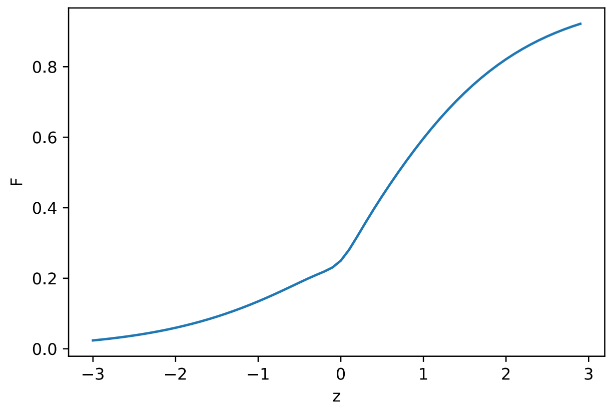


Figure 5.8: Smooth version of the CDF in 5.3.

### 5.4.2 Approach 2: Smooth CDF.

As seen in section 5.4.1 we can smooth the transition between the two curves in Equation 5.6 using (steep) sigmoid functions. This gives the following smooth CDF function:

$$\tilde{F}(z;\alpha) = (2-\alpha)\sigma(z)\sigma(-\beta z) + \frac{2-\alpha+\alpha\tanh(\frac{z}{2})}{2}\sigma(\beta z) \tag{5.9}$$

Inverting this CDF is very tricky, making it difficult to directly use the inverse-CDF method to draw (reparameterised) samples.
Thankfully it is still possible to draw close approximations of actual samples using Newton's Method [1]. This method is used to find the roots of real-valued functions. To obtain a sample we would like to solve the equation $z = \tilde{F}^{-1}(u;\alpha)$, where $u \sim \text{Unif}(0,1)$. So we can obtain a $z' \approx z$ by applying Newton's Method on the equation

$$\tilde{F}(z;\alpha) - u = 0 \tag{5.10}$$

The right-hand side of this equation is differentiable as required and only crosses the $z$-axis at a single point so the iterative algorithm will converge to the correct root: our sample.

We can differentiate (5.10) to directly obtain the resultant "PDF". Unfortunately this "PDF" is negative for certain values when $\alpha$ is close to 0 or 2. This is problematic since PDFs are supposed to remain non-negative by definition [24]. Negative values also make the required ELBO (Equation 2.5) computation impossible since the log probability $\log Q(z)$ becomes undefined.

### 5.4.3 Approaches 3: Smooth inverse-CDF

We can smooth between the two curves in Equation 5.7 using sigmoid functions as such,

$$\tilde{F}^{-1}(u;\alpha) = \log\frac{u}{2-\alpha-u}\sigma\left(-\beta\left(u-\frac{2-\alpha}{2}\right)\right) + \log\frac{\alpha+u-1}{1-u}\sigma\left(\beta\left(u-\frac{2-\alpha}{2}\right)\right) \tag{5.11}$$

where the steep sigmoid is translated so that the smoothing occurs at the correct separation of cases. The problems with this approach come from the fact that the first term is undefined for $u \geq 2-\alpha$ and the second term is undefined for $u \leq 1-\alpha$. And since $u \in [0,1]$ and $\alpha \in [0,2]$ we have that Equation (5.11) is only fully defined precisely when $\alpha = 1$, ie. when our approximate posterior StepLogistic is the same as the standard unit Logistic.

In principle the problem is simple; find a smooth transition from one curve to the other at their point of intersection, which is always along the horizontal coordinate ($u$) axis. The method of doing this by using steep sigmoids breaks since we still to compute the relevant terms when they become undefined. That doesn't mean a method doesn't exist.

# Chapter 6

# Conclusion

The variational autoencoder is a powerful and fast generative model which applies variational inference to approximate the underlying (latent) distribution of the data given some observations. Through the use of a trained decoder network, the VAE maps samples from the latent distribution back to the "observation space". Random sampling of the latent space means the generated outputs are unique, and exploration/knowledge of this space enables the generation of new data with desirable characteristics.

In this project, we attempted to give the VAE a stronger ability to model data whose underlying (latent) distribution must be (discrete) Bernoulli. In order to train our models as neural networks, the discrete latent variables had to be approximated using continuous ones for backpropagation to work. Naturally, the quality of the approximation is directly linked to the shape of the latent distribution (also called the approximate posterior) PDF. Towards the goal of finding a suitable distribution, a VAE was implemented and trained on the MNIST data-set. Although MNIST is not an exemplary data-set for requiring a binary latent space, it was sufficient to reveal issues with approaches involving standard distributions such as the Gaussian and the Logistic. We measure the performance of these approaches using their bound on the log probability of the data, which required maximising the ELBO, Equation (2.3, 2.5). The main issues and benefits when training with the Logistic (and similarly for Gaussian) approximate posterior (Chapter 4.4, 4.5) were

- Poor performance of the ELBO when optimising using the Logistic ELBO, but correct (posterior) bound being optimised.
- Stronger performance when optimising directly with the Bernoulli ELBO, but with the incorrect bound being optimised.

These phenomena were visualised as graphs in Figures 4.6, 4.7 and 4.8. To address these issues whilst retaining any benefits we proposed the new "StepLogistic" distribution (Chapter 5.2), to use as approximate posterior. The motivation behind its shape came from a simplified model where the input data was noisy versions of Bernoulli samples, under such a model the true posterior was tractable. It was parameterised by an area scaling factor for each Logistic(0,1) half, which is different to the other approaches parameterised by the mean (Logistic, Chapter 4.2) or a sigmoidal translation parameter (Concrete, Chapter 4.3). We saw optimising the area scaling factor intrinsically led to a stronger bound since its regulariser (Equation (2.3, 5.5)) is the same as that of the Bernoulli regulariser, unlike the Logistic (Table 5.1). Confirmation of these improvements to the evidence lower bound are plotted in Figure 5.5, where the StepLogistic's ELBO was much closer to the underlying true Bernoulli ELBO than the other approaches (about 7.4% greater bound than Logistic ELBO).

Looking at the ELBO is one way to measure the performance of the model. Other, less reliable approaches include visually comparing new generated outputs, or measuring the reconstruction loss when attempting the reconstruct an input. Both are unreliable because one cannot be quantified and is subject to human opinion and the other requires taking repeated samples from the approximate posterior to work out an average loss, in addition to the fact that VAEs are not designed to recreate their input as closely as possible (see autoencoder Chapter 2.3).
Nevertheless, in terms of generating new MNIST digits there are no noticeable improvements when using the new model (Figure A.3). For reconstructing inputs, we even saw that the new model sometimes performed worse than a standard model (Figure 5.7). However, this may be evidence that the MNIST data-set is not appropriate for our task rather than the model being at fault. Indeed, we were using an 8-dimensional binary latent space, which only allows a maximum of of $2^8 = 256$ distinct digits, far smaller than the 60 000 used for training. Even when increasing the latent dimension size to 96, the mysterious phenomenon where the wrong digit corresponding to the approximate posterior was reconstructed, was present (see Figure A.4). Although the reconstructions did improve slightly when increasing the discrete latent dimension, this phenomenon was not present under the continuous model (Chapter 3.3), with examples in Figure A.2.

The major drawback with the current StepLogistic model was the numerical instability during training when using a higher dimensional latent space. Fixing this issue is no trivial task, and either requires careful implementation to prevent any gradients from exploding (becoming too large for PyTorch to handle) or vanishing (becoming so small PyTorch assumes them to be zero), or requires a PDF with a smooth step where there is no instantaneous change in probability mass at the origin (Chapter 5.4). Unfortunately for the latter, we showed there were difficult mathematical challenges to overcome in order to find such a "SmoothStepLogistic" without having to use slow numerical approximations (which themselves bring further potential for numerical instability).

The StepLogistic displays promise, but ultimately needs further work to make it a viable for more complex latent space configurations.

To that end, here is some relevant further work:

- Test the model with data-sets more suited to a binary latent space. One option may include language and speech (intrinsically discrete) data-sets such as CSTR [29] containing spoken words in the English language from people with various different accents.
  Another strong option is a calcium imaging data-set [5] for measuring the activity of neurons. This data know to display spiking activity, making it suitable for a discrete latent space.
  This should prevent any "mysterious phenomena".

- Fix the numerical stability issues by finding a practical "soft step" version of the StepLogistic, or by re-engineering the model in PyTorch to avoid exploding or vanishing gradients. This is in an attempt to fully satisfy the last objective laid out in Chapter 1.4.

# Bibliography

[1] Weisstein, eric w. "newton's method." from mathworld–a wolfram web resource. https://mathworld.wolfram.com/newtonsmethod.html.

[2] Laurence Aitchison, Vincent Adam, and Srinivas C Turaga. Discrete flow posteriors for variational inference in discrete dynamical systems. *arXiv preprint arXiv:1805.10958*, 2018.

[3] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5:2, 1993.

[4] Narayanaswamy Balakrishnan. *Handbook of the logistic distribution.* CRC Press, 1991.

[5] CodeNeuro. Neuro finder calcium imaging data. *https://github.com/CodeNeuro/neurofinder*, 2016.

[6] Daniel Commenges. Information theory and statistics: an overview. *arXiv preprint arXiv:1511.00860*, page 6, 2015.

[7] Thomas M Cover and Joy A Thomas. *Elements of information theory (pages 19, 28).* John Wiley & Sons, 2012.

[8] Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. In *Advances in neural information processing systems.*

[9] Dr Carl Henrik Ek. *COMS3007 Machine Learning.* University of Bristol, 2019.

[10] Johannes Friedrich, Pengcheng Zhou, and Liam Paninski. Fast online deconvolution of calcium imaging data. *PLoS computational biology*, 13(3):e1005423, 2017.

[11] Brian Gaulter. *Further Pure Mathematics.* Oxford University Press, 2001.

[12] Andrea Giovannucci, Johannes Friedrich, Pat Gunn, Jeremie Kalfon, Brandon L Brown, Sue Ann Koay, Jiannis Taxidis, Farzaneh Najafi, Jeffrey L Gauthier, Pengcheng Zhou, et al. Caiman an open source tool for scalable calcium imaging data analysis. *Elife*, 8:e38173, 2019.

[13] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[14] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[15] Jonathan Goodman. Simple sampling gaussians. *From "Monte Carlo Methods" notes, Courant Institute of Mathematical Sciences, NYU*, 2005, https://www.math.nyu.edu/faculty/goodman/teaching/MonteCarlo2005 /notes/GaussianSampling.pdf.

[16] Chayan Kathuria. Regression - why mean square error. *https://towardsdatascience.com/https-medium-com-chayankathuria-regression-why-mean-square-error-a8cad2a1c96f*, 2019.

[17] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, pages 3–5, 2013.

[18] Scott M Lynch. Basics of bayesian statistics. In *Introduction to applied bayesian statistics and estimation for social scientists*, page 50. Springer, 2007.

[19] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, page 14, 2016.

[20] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014.

[21] Radford M Neal. *Probabilistic inference using Markov chain Monte Carlo methods*. Department of Computer Science, University of Toronto Toronto, ON, Canada, 1993.

[22] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[23] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

[24] Sheldon M Ross et al. *A first course in probability*, volume 7. Pearson Prentice Hall Upper Saddle River, NJ, 2006.

[25] Irhum Shafkat. Intuitively understanding variational autoencoders. *https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf*, 2018.

[26] Karl Sigman. Inverse transform method. *Columbia University, http://www.columbia.edu/ ks20/4404-Sigman/4404-Notes-ITM.pdf*, 2010.

[27] Zachary Singer. Manifolds in data science: A brief overview. *https://towardsdatascience.com/manifolds-in-data-science-a-brief-overview-2e9dde9437e5*, 2019.

[28] Yung Liang Tong. *The multivariate normal distribution*. Springer Science & Business Media, 2012.

[29] Junichi Yamagishi. English multi-speaker corpus for cstr voice cloning toolkit. *http://homepages.inf.ed.ac.uk/jyamagis/page3/page58/page58.html*, 2012.

[30] yzwxx. Variational autoencoder trained on celebrity faces. *https://github.com/yzwxx/vae-celebA*, 2017.

[31] Shengjia Zhao, Jiaming Song, and Stefano Ermon. Towards deeper understanding of variational autoencoding models. *arXiv preprint arXiv:1702.08658*, 2017.

# Appendix A

# Appendix

## A.1 The hyperbolic secant function ”sech”

(Section 10, [11])
We use this function to define the Logistic and StepLogistic PDFs. For $x \in \mathbb{R}$,

$$\operatorname{sech}(x) = \frac{1}{\cosh(x)} = \frac{2}{e^x + e^{-x}}$$

## A.2 Derivation of StepLogistic($z$;$\alpha$) CDF when $z > 0$

Let $\mathrm{F}(z;\alpha, z > 0)$ denote the CDF for the StepLogistic$(z;\alpha)$ distribution when $z > 0$. To find this function we must integrate the StepLogistic PDF, which is done by separating it into its constituent Logistic PDFs. We integrate over dummy variable $x$ until $x = z$.This gives

$$\begin{aligned} F(z;\alpha, z > 0) &= \int_{-\infty}^{z} \text{StepLogistic}(x;\alpha)dx \\ &= (2-\alpha)\int_{-\infty}^{0} \text{Logistic}(0,1)dx + \alpha \int_{0}^{z} \text{Logistic}(0,1)dx \\ &= \frac{2-\alpha}{2} + \alpha \int_{0}^{z} \frac{1}{4}\text{sech}^2\left(\frac{x}{2}\right)dx \\ &= \frac{2-\alpha}{2} + \frac{\alpha}{2}\text{tanh}\left(\frac{x}{2}\right)\Big|_{0}^{z} \\ &= \frac{2-\alpha}{2} + \frac{\alpha}{2}\text{tanh}\left(\frac{z}{2}\right) \end{aligned} \tag{A.1}$$

where $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ is the hyperbolic tangent function. We make use of the fact that the integral of $\operatorname{sech}^2(x)$ is $\tanh(x)$ (+c).

## A.3 Proof of equality in K-L divergence between StepLogistic model and Bernoulli model

Let $f_S$ and $f_L$ denote the StepLogistic$(\alpha)$ and Logistic$(0,1)$ PDFs respectively. We have

$$D_{KL}(\text{StepLogistic}(\alpha)||\text{Logistic}(0,1))$$

$$\begin{aligned}
&= \int_{-\infty}^{\infty} f_S \log\left(\frac{f_S}{f_L}\right) dz \\
&= \int_{-\infty}^{0} \frac{(2-\alpha)}{4}\text{sech}^2(\frac{z}{2}) \log\left(\frac{\frac{(2-\alpha)}{4}\text{sech}^2\left(\frac{z}{2}\right)}{\frac{1}{4}\text{sech}^2\left(\frac{z}{2}\right)}\right) dz + \int_{0}^{\infty} \frac{\alpha}{4}\text{sech}^2\left(\frac{z}{2}\right) \log\left(\frac{\frac{\alpha}{4}\text{sech}^2\left(\frac{z}{2}\right)}{\frac{1}{4}\text{sech}^2\left(\frac{z}{2}\right)}\right) dz \\
&= (2-\alpha)\log(2-\alpha) \int_{-\infty}^{0} \frac{1}{4}\text{sech}^2\left(\frac{z}{2}\right) dz + \alpha\log\alpha \int_{0}^{\infty} \frac{1}{4}\text{sech}^2\left(\frac{z}{2}\right) dz \\
&= \frac{1}{2}(2-\alpha)\log(2-\alpha) + \frac{1}{2}\alpha\log\alpha
\end{aligned} \tag{A.2}$$

For the discrete Bernoulli K-L Divergence we have

$$\begin{aligned}
D_{KL}(\text{Bern}\left(\frac{\alpha}{2}\right)||\text{Bern}(0.5)) &= \left(1-\frac{\alpha}{2}\right)\log\left(\frac{1-\frac{\alpha}{2}}{0.5}\right) + \left(\frac{\alpha}{2}\right)\log\left(\frac{\frac{\alpha}{2}}{0.5}\right) \\
&= \frac{1}{2}(2-\alpha)\log(2-\alpha) + \frac{1}{2}\alpha\log\alpha
\end{aligned} \tag{A.3}$$

This completes the proof.

## A.4 Integral of PDF (5.8) between $-\infty$ and $\infty$

Let be $\tilde{f}$ be the PDF as defined in (26) with area scale factor parameter $\alpha \in [0, 2]$ and inverse-temperature $\beta \in (0, \infty)$. We can split the required integral as the sum of two integrals A and B

$$\int_{-\infty}^{\infty} \tilde{f}(z; \alpha, \beta) dz = A + B \tag{A.4}$$

where

$$A = (2-\alpha) \int_{-\infty}^{\infty} \frac{1}{4}\text{sech}^2\left(\frac{z}{2}\right) \sigma(-\beta z)\, dz \tag{A.5}$$

and

$$B = \alpha \int_{-\infty}^{\infty} \frac{1}{4}\text{sech}^2\left(\frac{z}{2}\right) \sigma(\beta z)\, dz \tag{A.6}$$

Considering B and using the fact that $\sigma(\beta z) + \sigma(-\beta z) = 1$ and $\text{sech}^2(\frac{z}{2}) = \text{sech}^2(\frac{-z}{2})$ we can integrate over $z$ between 0 and $\infty$ then add the relevant term for $-z$ to obtain the integral over the whole real line.

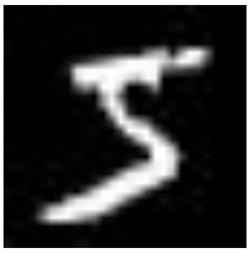
$$\begin{aligned}
B &= \alpha \int_{0}^{\infty} \frac{1}{4}\text{sech}^2\left(\frac{z}{2}\right) \sigma(\beta z) + \frac{1}{4}\text{sech}^2\left(\frac{-z}{2}\right) \sigma(-\beta z)\, dz \\
&= \alpha \int_{0}^{\infty} \frac{1}{4}\text{sech}^2\left(\frac{z}{2}\right) [\sigma(\beta z) + \sigma(-\beta z)] dz \\
&= \alpha \int_{0}^{\infty} \frac{1}{4}\text{sech}^2\left(\frac{z}{2}\right) dz \\
&= \frac{\alpha}{2}
\end{aligned} \tag{A.7}$$

Similarly we have that $A = \frac{2-\alpha}{2}$ and using (33)

$$\int_{-\infty}^{\infty} \tilde{f}(z;\alpha,\beta)dz = 1$$

as required. This symmetry "trick" can also be used to compute $\int_{-x}^{x} \tilde{f}(z;\alpha,\beta)dz$ for any real $x$.

## A.5 Gaussian model with sigmoid output reconstructions


(a) Original "5"

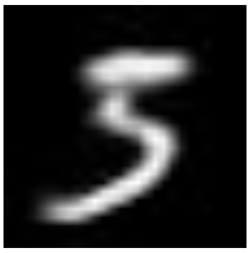
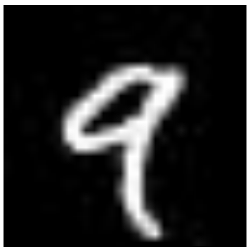

(b) Reconstruction

Figure A.1: Generated digits from the StepLogistic and Logistic models. 8-dimensional binary latent space.


(a) Original "9"

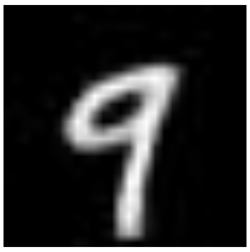

(b) Reconstruction

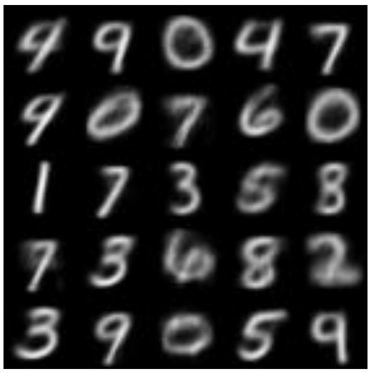
Figure A.2: Typical reconstructions using continuous Gaussian model (with improved sigmoid output) with 56-D latent space.

## A.6 Random generated digits from StepLogistic and Logistic models

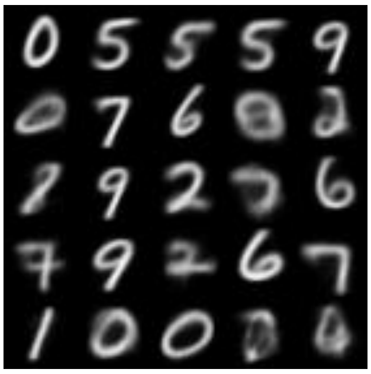
Taking random samples from the 8-dimensional latent space.

## A.7 Digit reconstructions with 96-D Bernoulli latent

## A.8 StepLogistic applied to Fashion MNIST data-set

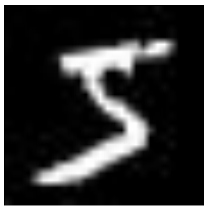


(a) StepLogistic model

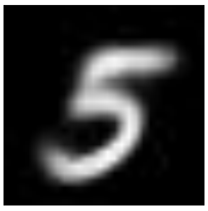


(b) Logistic model

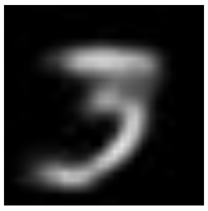
Figure A.3: Generated digits from the StepLogistic and Logistic models. 8-dimensional binary latent space.



(a) Original



(b) Reconstruction 1



(c) Reconstruction 2

Figure A.4: Two possible reconstructions of the digit 5 from two random samples of the same approximate posterior. Logistic model.

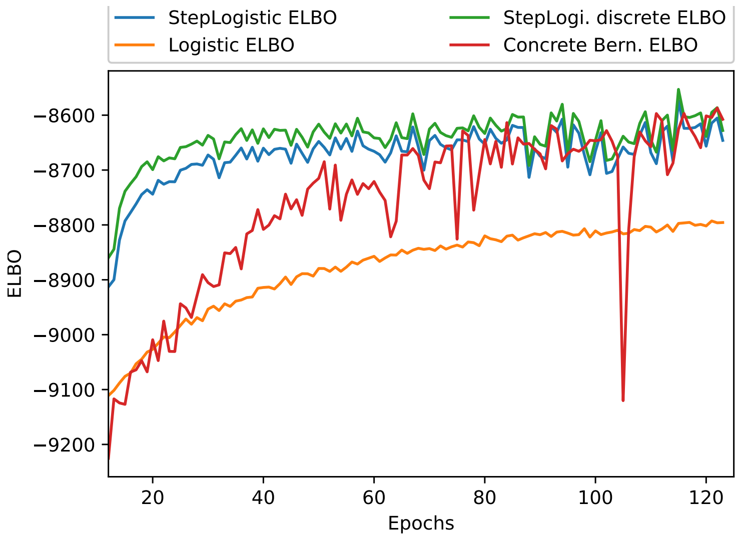


Figure A.5: Comparing ELBO performance for the Logistic, StepLogistic and Concrete models applied to the Fashion MNIST data-set. 8-dimensional Bernoulli latent distribution. Only displaying ELBO averages starting from the epoch 12.