Imperial College of Science, Technology and Medicine
Department of Computing

On the Feasibility of Using Fully-Convolutional Variational Autoencoders to Advance Deep Symbolic Reinforcement Learning

D. G. Sherburn

Submitted in part fulfilment of the requirements for the degree of
Master of Engineering in Joint Mathematics and Computing
of Imperial College, June 2017

Acknowledgements

I would like to express (whatever feelings I have) to:

- My supervisor
- My second supervisor
- Other researchers
- My family and friends

Dedication

Dedication here.

‘Quote text here.’

Guy Quoted

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

A long term goal of artificial intelligence (AI) is the development of artificial general intelligence (AGI). Since the field's inception in the 1950s, it has swung between hype and breakthroughs, followed by disappointment and reduced funding, known as AI winters [9]. During the first period of hype from the 50s to the early 70s, Marvin Minsky made the following prediction: [4]

> "In from three to eight years we will have a machine with the general intelligence of an average human being." - Marvin Minsky, 1970

This prediction was clearly not realised, and the first AI winter would shortly follow.

Symbolic AI was developed during this winter, which encodes knowledge as human-readable rules and facts, making it easy to comprehend chains of actions and abstract relationships [15]. For instance, given the unary relations `red` and `strawberry`, and the binary relation `bigger`, we can say that `A` is the smallest red strawberry by writing

```
red(A)  strawberry(A)  ∀B bigger(B, A)
```

But given the unary relations `yellow` and `banana` we could also write that `A` is the third biggest yellow strawberry, or a red banana, and so on. We can see that the rules and facts in symbolic logic can be endlessly recombined and extended. This allows for the manipulation of high-level abstract concepts, which is key to AGI [6].

However, symbolic AI has a major philosophical problem: the facts and rules are only meaningful to the human writing them; their meaning is not intrinsic to the system itself. This is known as the *symbol grounding problem*.

Today we find ourselves in yet another period of hype and exciting breakthroughs not afflicted by the symbol grounding problem. Reinforcement learning (RL) has become a prominent area of research, with many considering it fundemental for AGI [7], as have deep neural networks. Recently, deep reinforcement learning (DRL) systems have achieved impressive feats, including mastering a wide range of Atari 2600 games to a superhuman level using only raw pixels and score as input, and the board game Go [13, 17].

Figure 1.1: May 1997: Gary Kasparov makes his first move against IBM's Deep Blue. Deep Blue would later emerge the victor in the best of six games; the first time a reigning world chess champion is defeated by a computer. [16]

Figure 1.2: March 2016: Lee Sedol, one of the greatest modern Go players, plays his first move of game three against AlphaGo. AlphaGo won four of five games. This feat was considered by many to be a decade away. [14]

Though DRL systems are not afflicted by the same problems as symbolic AI, they have a number of drawbacks of their own. Namely, they are: [6]

1. **Slow to learn**. Neural networks require large data sets and are therefore slow to learn.

2. **Unable to transfer past experience**. They often fail to perform well on tasks very similar to those they have mastered.

3. **Unable to reason abstractly**. They fail to exploit statistical regularities in the data.

4. **Hard to reason about**. It's often difficult to extract a comprehensible chain of reasons for why a deep neural network operated in the way it did.

Deep symbolic reinforcement learning (DSRL) is a marrying of DRL and symbolic AI; a recent advance which overcomes the symbol grounding problem and the drawbacks associated with DRL [6]. That is, DSRL systems overcome the symbol grounding problem, and are:

1. **Fast to learn**. Large data sets are not necessary.

2. **Able to transfer past experience**. Symbolic AI lends itself to multiple processes associated with high-level reasoning, including transfer learning.

3. **Able to reason abstractly**. The agent is able to exploit statistical regularities in the training data by using high-level processes like planning or causal reasoning.

4. **Easy to reason about**. Since the front end uses symbolic AI, its knowledge is encoded as human-readble facts and rules, making the extraction of comprehensible chains of logic much easier.

An overview of DSRL is shown in Figure 1.3. The neural back end takes a high-dimensional input and outputs a symbolic representation. This symbolic representation is then fed to the symbolic front end, whose role is action selection. The agent then acts on the environment and obtains a reward and the sensory input of the next time step. As the neural back end learns how to represent the raw input data in a compositionally structured representation in an unsupervised manner, and the symbolic front end learns to select the action with maximum expected reward over time, the system as a whole learns end-to-end.

TODO: Finish description of DSRL

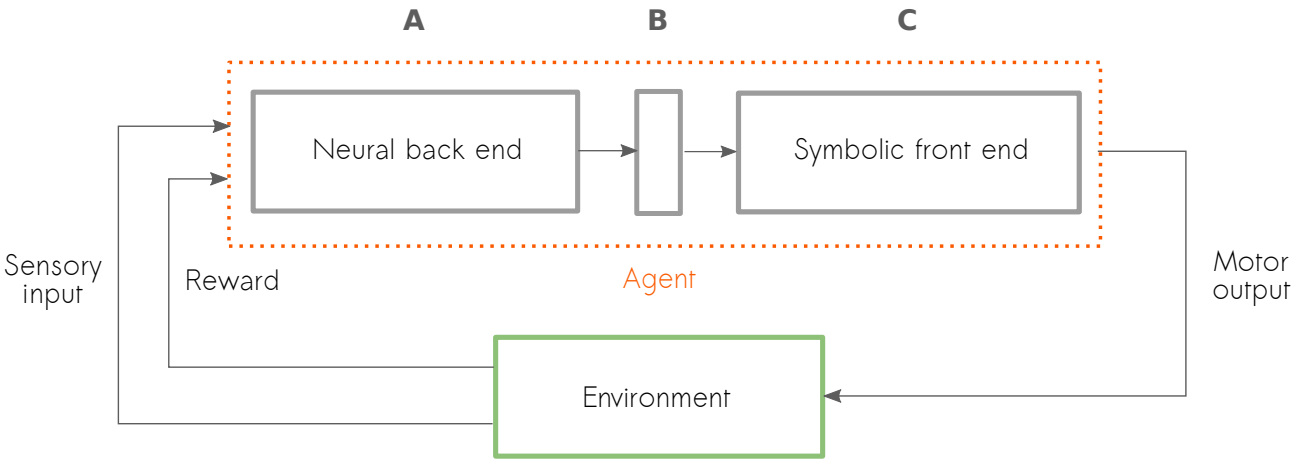

Figure 1.3: Overview of deep symbolic reinforcement learning system architecture. **A**: The neural back end maps high-dimensional raw input data to a compositionally structured symbolic representation. **B**: The compositionally structured symbolic representation. **C**: Reinforcement learning of mapping from symbolic representation to action with maximum expected reward over time. *Source: Garnelo et al.* [6].

1.2 Objectives

We'll use the image in Figure 1.4 as an example of a high-dimensional input to the neural back end. This world consists of only two shapes (circle and square) and four spaces occupied by at most one shape (top left, top right, bottom left and bottom right). The neural back end maps this raw high-dimensional input to a low-dimensional symbolic representation, shown in Table ??.

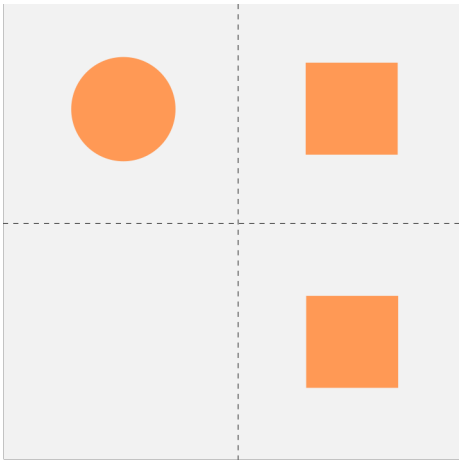

Figure 1.4: A toy example of a raw high-dimensional input.

How this is done will be explained in Chapter 2, but as for now we can just take it as fact that the current method doesn't scale. That is, for very simple scenes, as in Figure ..., This is done

Type	Location
1	[0, 0]
2	[0, 1]
2	[1, 1]

Table 1.1: Low-dimensional symbolic representation

by passing the high-dimensional input through a series of convolutional layers and extracting the activation spectra in the latent space. These spectra are then used to classify the

relies on the unsupervised extraction of disentangled features, allowing for transfer learning and high-level cognitive processes. However, the unsupervised extraction of features from a wide range of scenes is still a challenge in AI research [?]. Fortunately methods are getting better, and the first unsupervised scalable model β-VAE was developed recently.

TODO: Finish objectives

1.3 Contributions

Contributions here.

Chapter 2

Background Theory

2.1 Introduction

We will cover how deep symbolic reinforcement learning extracts symbolic representations from raw input data, which will motivate a discussion of loss functions and the introduction of autoencoders. Seeing the limitations of the current approach in extracting symbolic representations, we can appreciate the recent development of β-VAE, a variant of the variational autoencoder used to learn disentangled representations. Finally, we can conclude by mentioning less technical matters, such as libraries and hardware used.

2.2 Arithmetic in Neural Networks

2.2.1 Neurons

2.2.2 Activation Functions

2.2.3 Convolutions

2.2.4 Deconvolutions

2.2.5 Pooling and Up-Sampling

2.3 Loss functions

The idea of image reconstruction plays a vital role throughout this project. Although it's possible to qualitatively compare the original to its reconstruction, it's important to be able to quantify the difference, which lends itself to automation. The loss function will quantify how similar two images are.

To compare loss functions, we'll use the MNIST data set. MNIST is a collection of $70,000$ black-and-white images of handwritten digits, with $60,000$ in the training set containing and $10,000$ in the test set. These images will be represented as vectors without loss of generality.

2.3.1 Euclidean Distance

The Euclidean distance between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ is defined by

$$\sqrt{\sum_i (x_i - y_i)^2}$$

where x_i and y_i are the i^{th} components of $\boldsymbol{x}$ and $\boldsymbol{y}$ respectively.

Euclidean distance is an intuitive measure of the distance between two points in space. Unfortunately, this doesn't also translate to visual similarity, as illustrated by Doersch et al. [5]. Figure 2.1 is a digit drawn from the MNIST dataset, and Figures 2.3 and 2.2 are attempted reconstructions. Of the reconstructions, Figure 2.2 looks most like the original, but Figure 2.3 is closer in space.

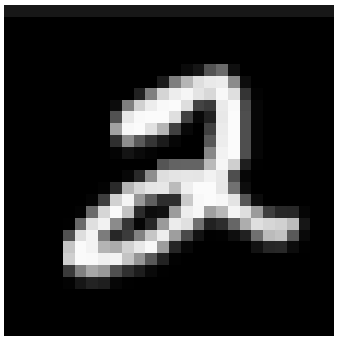

Figure 2.1

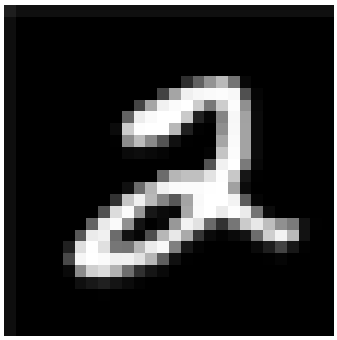

Figure 2.2

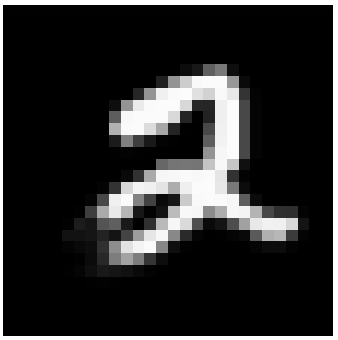

Figure 2.3

This leads to an alternative measure, binary cross-entropy, which gives a much better quantification of how visually similar two images are.

2.3.2 Binary Cross-Entropy

Consider a single black-and-white pixel with probability $p(0) = c$ of being 0 and $p(1) = 1 - c$ of being 1. Here $p(x)$ is a probability distribution over the possible pixel values $x \in \{0, 1\}$. Suppose a given model tries to learn the distribution described by $p(x)$, and says that the pixel has probability $q(0) = \hat{c}$ of being 0 and $q(1) = 1 - \hat{c}$ of being 1. The model is perfect if it learns the true distribution, that is, if $q(x) = p(x)$ for $x \in \{0, 1\}$. We'd like to quantify how similar the distributions p and q are.

This is done by computing the binary cross-entropy between p and q, which is defined by

$$H(p, q) = -c \log \hat{c} - (1 - c) \log(1 - \hat{c})$$

To see how we may use this as a similarity measure among images, consider a 1×1 image.

Normalising this image yields a pixel value in the interval $[0, 1]$, which may now be interpreted as a probability, corresponding to c above. In the normalised reconstructed image, the pixel value corresponds to $\hat{c}$. We simply compute the binary cross-entropy to measure the similarity of these two distributions, and in turn, the similarity of the images themselves! (Note: we could have also assigned the probabilities to $1 - y$ and $1 - \hat{y}$ by symmetry of binary cross-entropy).

For images larger than 1×1, we may take the component-wise binary cross-entropy, then, for example, average the components. How the component-wise binary cross-entropies are suitably combined to give a single floating point number will vary from problem to problem.

2.4 Autoencoders

An autoencoder is a neural network that learns a compression algorithm for its input data in an unsupervised manner [12]. This is achieved by placing constraints on a hidden layer, called the latent space, and setting the target values to the input values, effectively learning the identity function. Since the network is trying to reconstruct the original input from the constrained latent space, over time the latent space corresponds to a meaningful compression of the network's input.

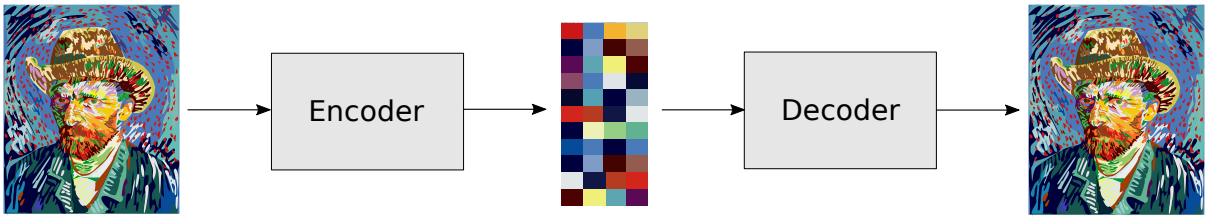

Figure 2.4: A black-box description of an autoencoder. The autoencoder learns the identity function, and in turn, the encoder and decoder learn suitable encoding and decoding algorithms respectively.

As before, we will use the MNIST data set to compare architectures. Unless specified in the example, the Adam optimiser is used with a learning rate of $1e - 4$, the batch size is 1 and the loss function is binary cross-entropy. Intermediate layers use the ReLU activation function, while the final layer uses sigmoid.

2.4.1 Fully-Connected Autoencoders

In dense feed-forward neural networks we may place a constraint on the latent space by reducing the number of neurons, as shown in Figure 2.5. Images must be flattened into vectors to be fed as input. Consequently, any spatial information is destroyed in dense feed-forward neural networks.

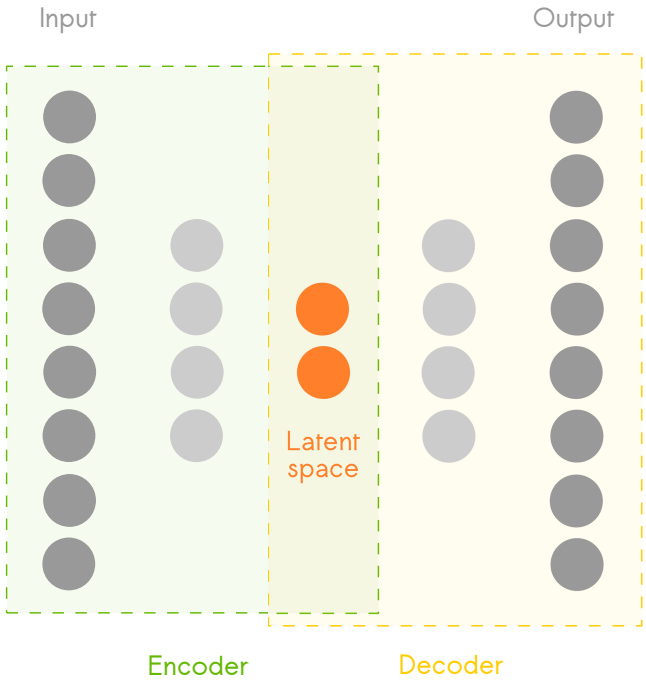

Figure 2.5: An example architecture of a fully-connected autoencoder. The latent space is constrained by having fewer neurons than the input and output layers.

An example architecture is given in Table 2.1, which was trained on MNIST. Despite the latent space being $\sim 4\%$ of the size of the input space, the network is capable of producing realistic reconstructions. For verification, a collection of samples from the dataset and their corresponding reconstructions are shown in Figure 2.6.

Layer Type	**Output Shape**
InputLayer	(1, 28, 28)
Flatten	(784,)
Dense	(32,)
Dense	(784,)
Reshape	(1, 28, 28)

Table 2.1: A simple fully-connected autoencoder with one hidden layer. After 15 epochs, the validation score was recorded to be 71.94.

Figure 2.6: A collection of images from the MNIST data set and their respective reconstructions using the fully-connected autoencoder specified in Table 2.1. The original MNIST images are in odd columns, and their reconstructions to their immediate right.

2.4.2 Fully-Convolutional Autoencoders

In fully-convolutional feed-forward neural networks, we may place a constraint on the latent space by reducing the number and/or size of the filters, as shown in Figure 2.7. To compare the fully-convolutional autoencoder to the fully-connected, we'll train the architecture in Table 2.2 on MNIST. As before, we'll compare the reconstructions to the originals, which can be found in Figure 2.8.

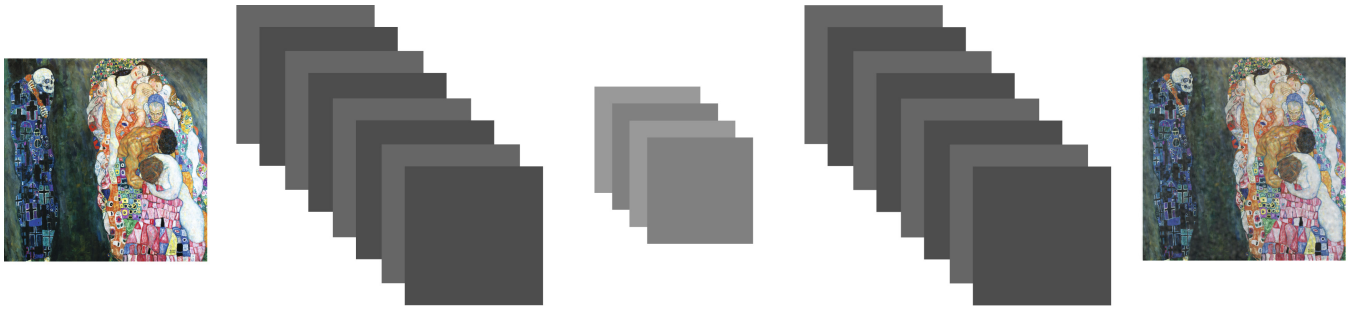

Figure 2.7: An example architecture of a fully-convolutional autoencoder. The latent space is constrained by reducing the number and/or size of the filters.

Layer Type	**Output Shape**
InputLayer	(1, 28, 28)
Conv2D	(32, 28, 28)
MaxPooling2D	(32, 14, 14)
Conv2D	(4, 14, 14)
MaxPooling2D	(4, 7, 7)
UpSampling2D	(4, 14, 14)
Conv2DTranspose	(32, 14, 14)
UpSampling2D	(32, 28, 28)
Conv2DTranspose	(1, 28, 28)

Table 2.2: A simple fully-convolutional autoencoder with 2D convolutions and max pooling, plus the corresponding deconvolutional layers. After 15 epochs, the validation score was recorded to be 64.89.

Convolutional layers have been shown to be effective in tasks with images as input [10, 19, 18]. This is because spatial information is preserved in convolutional layers, and the number of trainable parameters is far less in a convolutional layer than it is in a fully connected layer. Convolutional layers will be used from here on as we'll be using images as input.

Figure 2.8: A collection of images from the MNIST data set and their respective reconstructions using the fully-convolutional autoencoder specified in Table 2.2. The original MNIST images are in odd columns, and their reconstructions to their immediate right.

2.4.3 Variational Autoencoders

The variational autoencoder is central to this project, and we'll therefore dedicate a considerable amount of time exploring it. First we'll precisely define the problems the variational autoencoder solves, after which we may develop its loss function and detail its implementation. This section will conclude by developing an intution of the theory covered by way of examples.

A Probabilistic Perspective

We'll begin by making necessary definitions and describe the input data as samples from a generative process.. This sets the context to detail the problems the variational autoencoder solves.

Let $X = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$ be the data set of N independent and identically distributed samples of the variable $\boldsymbol{x}$. (X may be a data set of images, for instance). Let us assume that these samples are generated by a random process with parameters $\boldsymbol{\theta}^*$ involving an unobserved latent variable $\boldsymbol{z}$ in the following way:

Algorithm 1 Generate data set X

1: **for** $i = 1 \rightarrow N$ **do**
2: $\quad \boldsymbol{z}^{(i)} \sim p_{\boldsymbol{\theta}^*}(\boldsymbol{z})$ // Sample from true prior
3: $\quad \boldsymbol{x}^{(i)} \sim p_{\boldsymbol{\theta}^*}(\boldsymbol{x}|\boldsymbol{z}^{(i)})$ // Sample from true conditional
4: $\quad$ Append $\boldsymbol{x}^{(i)}$ to X
5: **end for**

We only observe the data set X in this process. The parameters $\boldsymbol{\theta}^*$ and latent variables $Z = \{\boldsymbol{z}^{(i)}\}_{i=1}^{N}$ are unknown to us. Let us assume that the prior $p_{\boldsymbol{\theta}^*}(\boldsymbol{z})$ and conditional $p_{\boldsymbol{\theta}^*}(\boldsymbol{x}|\boldsymbol{z})$ are parameterised by the distributions $p_{\boldsymbol{\theta}}(\boldsymbol{z})$ and $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ respectively. In this context, the variational autoencoder provides [8]:

1. ML or MAP estimation for the parameters $\boldsymbol{\theta}$
2. An approximation of the latent variable $\boldsymbol{z}^{(i)}$ given $\boldsymbol{x}^{(i)}$ and set of parameters $\boldsymbol{\theta}$
3. Approximate marginal inference of the variable $\boldsymbol{x}$

Overcoming the Intractable Posterior

The variational autoencoder solves the problems above by approximate inference of the latent variable $\boldsymbol{z}$. Exact inference is not possible, and to see this we may use Bayes' theorem to find an expression for the posterior:

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})} \tag{2.1}$$

The marginal likelihood

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})d\boldsymbol{z} \tag{2.2}$$

involves an exponential-time integral over every combination of the latent variable $\boldsymbol{z}$, and is therefore computationally intractable [8]. Instead, we define an approximation $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ to the intractable posterior. Since $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ gives a distribution over the possible latent variables $\boldsymbol{z}$ that generated the given data point $\boldsymbol{x}$, it is known as the probabilistic decoder. By the same reasoning, $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is known as the probabilistic decoder.

Finding a Suitable Loss Function: the ELBO

The variational autoencoers's ability to learn the generative parameters $\boldsymbol{\theta}^*$ relies on how closley $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ approximates $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. In the interest of training a model, this difference will be quantified. For this we use the KL divergence

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{2.3}$$

which measures how much information is lost when we represent $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ with $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ (measured in nats) [3]. Using the KL divergence, our problem now amounts to the optimisation problem [11]:

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\theta}^*, \boldsymbol{\phi}^*} D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) \tag{2.4}$$

To see how we can start to minimise the KL divergence, we'll start by rewriting it in a different form:

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{2.5}$$

$$= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log_e q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\big] - \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log_e p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\big] \tag{2.6}$$

$$= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log_e q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\big] - \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\right] \tag{2.7}$$

$$= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}\right] + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log_e p_{\boldsymbol{\theta}}(\boldsymbol{x})] \tag{2.8}$$

$$= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}\right] + \log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{2.9}$$

Here we see that the KL divergence depends on the intractable marginal likelihood $p_{\boldsymbol{\theta}}(\boldsymbol{x})$! There's no way we can minimise it if we can't write down $p_{\boldsymbol{\theta}}(\boldsymbol{x})$. However, we can get around this: we'll minimise the KL divergence, but not directly. Instead, we try to find a quantity which we can maximise, and show that in turn this minimises the KL divergence. The trick is not obvious, but is simply done by finding a lower bound on the log marginal likelihood.

Using Jensen's inequality

$$f(\mathbf{E}\big[X\big]) \geq \mathbf{E}\big[f(X)\big] \tag{2.10}$$

we can write down a lower bound on the log marginal likelihood:

$$\log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log_e \int p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) d\boldsymbol{z} \tag{2.11}$$

$$= \log_e \int p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z}) \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \tag{2.12}$$

$$= \log_e \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{2.13}$$

$$\geq \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{2.14}$$

$$:= \mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}) \tag{2.15}$$

Expression (2.14) is called the *ELBO* (short for expected lower bound) [2, 8].

How does the $ELBO$ help us with minimising the KL divergence? First recall the alternative form of the KL divergence written in Equation (2.9):

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}\right] + \log_e p_{\boldsymbol{\theta}}(\boldsymbol{x})$$

Writing this in terms of the $ELBO$ we have:

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) = -\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}) + \log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{2.16}$$

Since the KL divergence is the negative of the $ELBO$ up to an additive constant (with respect to q), minimising the KL divergence is equivalent to maximising the $ELBO$ [1]. Now we may make a revision to our original optimisaiton problem (2.4). Our problem is written as [11]:

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\theta}^*,\boldsymbol{\phi}^*} \mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}) \tag{2.17}$$

Writing the ELBO in Closed-Form

We've found that we can maximise the $ELBO$ to minimise the KL divergence between the approximation $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ and the intractable posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$. Now we will seek to write the $ELBO$ in closed-form, after which we will be able to implement the variational autoencoder.

To write the $ELBO$ in closed-form, we'll start with a useful manipulation:

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta},\boldsymbol{\phi};\boldsymbol{x}) &= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] && (2.18)\\
&= -\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x},\boldsymbol{z})}\right] && (2.19)\\
&= -\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}\right] && (2.20)\\
&= -\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z})}\right] + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] && (2.21)\\
&= -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] && (2.22)
\end{aligned}$$

Thus for a single data point $\boldsymbol{x}^{(i)}$, the $ELBO$ becomes:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big] \tag{2.23}$$

With some assumptions and a bit of work, we're finally in a position to write the $ELBO$ in closed-form. The first term is the KL divergence between the probabilistic encoder and the prior. To make our lives simpler, we can choose the prior to be the isotropic multivariate Gaussian:

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{2.24}$$

We will also assume that the posterior is a k-dimensional Gaussian with diagonal covariance. It follows that the approximate posterior should take the same form. That is,

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I}) \tag{2.25}$$

where $\boldsymbol{\mu}^{(i)}$ and $\boldsymbol{\sigma}^{(i)}$ are the mean and standard deviation (respectively) for data point $\boldsymbol{x}^{(i)}$. With these two assumptions, and the KL divergence for k-dimensional Gaussians,

$$D_{KL}(\mathcal{N}_0||\mathcal{N}_1) = \frac{1}{2}\bigg[\operatorname{tr}(\boldsymbol{\Sigma_1}^{-1}\boldsymbol{\Sigma_0}) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^T\boldsymbol{\Sigma_1}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - k + \ln\left(\frac{\det\boldsymbol{\Sigma_1}}{\det\boldsymbol{\Sigma_0}}\right)\bigg] \tag{2.26}$$

we may write the first term in closed-form:

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) = \frac{1}{2}\bigg[\sum_{j=1}^{k}\sigma_j^{2(i)} + \sum_{j=1}^{k}\mu_j^{2(i)} - k - \ln\prod_{j=1}^{k}\sigma_j^{2(i)}\bigg] \tag{2.27}$$

Now we turn our attention to the second term of the $ELBO$:

$$\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big] \tag{2.28}$$

Luckily, maximising terms like this is encountered regularly in statistics, and is known as maximum likelihood estimation. This can be taken to be the reconstruction loss (defined it

earlier) [11].

Therefore, using an unspecified reconstruction loss, the $ELBO$ is:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \frac{1}{2}\Big[\sum_{j=1}^{k} \sigma_j^{2(i)} + \sum_{j=1}^{k} \mu_j^{2(i)} - k - \ln \prod_{j=1}^{k} \sigma_j^{2(i)}\Big] + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big] \tag{2.29}$$

Implementing the Variational Autoencoder

The variational autoencoder uses a neural network for the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ [8]. First a naïve implementation of the variational autoencoder will be proposed. As we shall see, this implementation needs one adjustment, known as the "reparameterisatoin trick". This will lead to the final implementation of the variational autoencoder, and conclude our formal study.

The probabilistic encoder (2.25)

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I})$$

requires the mean $\boldsymbol{\mu}^{(i)}$ and standard deviation $\boldsymbol{\sigma}^{(i)}$ for a given data point $\boldsymbol{x}^{(i)}$. These two vectors will be represented by distinct layers in the latent space of a neural network, as shown in Figure (2.9) [11].

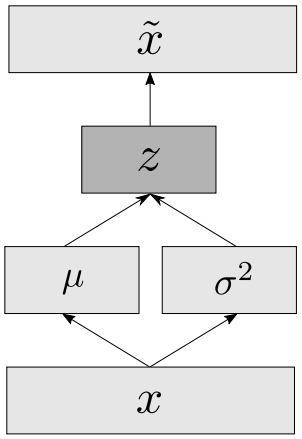

Figure 2.9: A naïve implementation of the variational autoencoder. The input $\boldsymbol{x}$ is mapped to intermediate layers taking the values of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$. The latent variable $\boldsymbol{z}$ is then sampled from the probabilistic encoder $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. Finally $\boldsymbol{z}$ is mapped back to the input dimension to give reconstruction $\tilde{\boldsymbol{x}}$.

Unfortuntely we cannot perform backpropagation, as it makes no sense to differentiate the the random variable $\boldsymbol{z}$ wrt $\boldsymbol{\phi}$ ($\boldsymbol{z}$ is drawn from a distribution, and not a function of $\boldsymbol{\phi}$). To solve this, we introduce an auxillary variable $\boldsymbol{\epsilon}$ and vector-valued function $g_\phi(\boldsymbol{x}, \boldsymbol{\epsilon})$ parameterised by $\boldsymbol{\phi}$ [8]. The auxillary variable $\boldsymbol{\epsilon}$ is governed by a parameterless distribution, which we will take to be the multivariate isotropic Gaussian.

The sampling step can now be written as

$$\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \tag{2.30}$$

where $\odot$ represents the element-wise product. This is a reparameterisation of the random variable $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ with a differentiable function $g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon})$, and is suitable known as the "reparameterisation trick" [8]. The reparameterisation trick allows backpropagation to be used, and thus completes our method of solving optimisation problem (2.17).

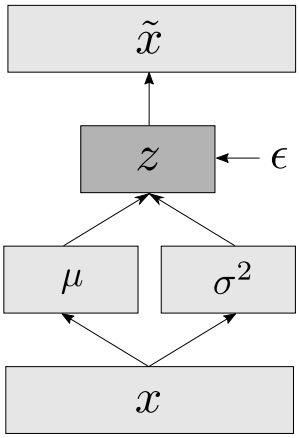

Figure 2.10: A viable implementation of the variational autoencoder. Sampling from the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is simulated by evaluating $\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$.

Intuition Behind the Variational Autoencoder

In this section we'll develop the intuition behind the variational autoencoder. We've been considering the data set $X = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$. In this section we'll take X to be the MNIST data set (the vector $\boldsymbol{x}^{(i)}$ now corresponds to a flattened MNIST image). For visualisatoin purposes, we'll also take the latent space dimension $k = 2$, that is, $\boldsymbol{z} = (z_1, z_2)$.

The probabilistic encoder (2.25)

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I})$$

describes where the latent variable $\boldsymbol{z}$ lies for data point $\boldsymbol{x}^{(i)}$. In Figure (2.11), two mappings are shown from

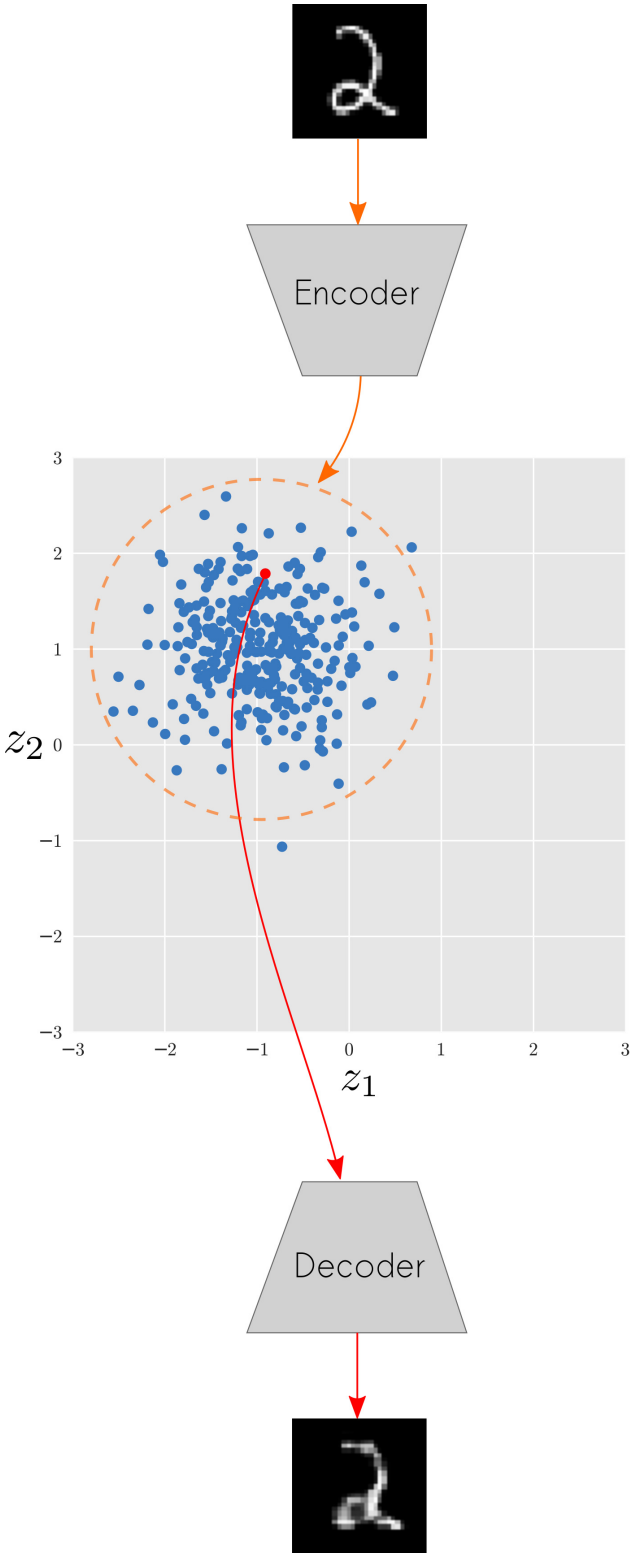

Figure 2.11: The encoder takes a data point and returns a normal distribution (orange); some samples of which are shown (blue). A sample is drawn from the normal distribution (red) and decoded. [?]

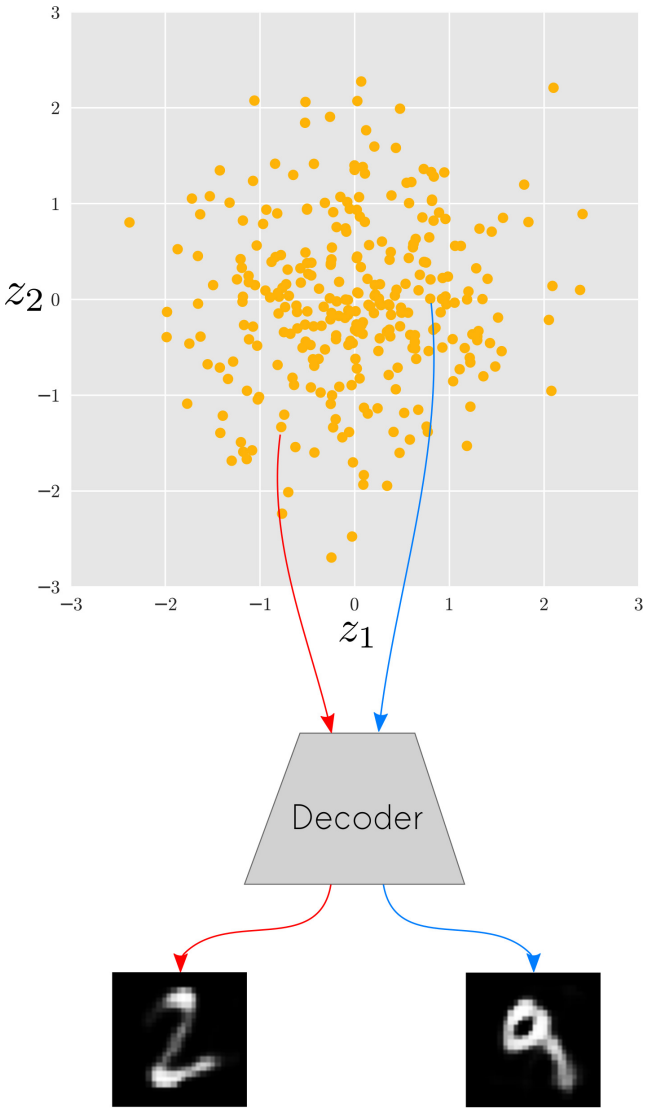

Figure 2.12: Caption. [?]

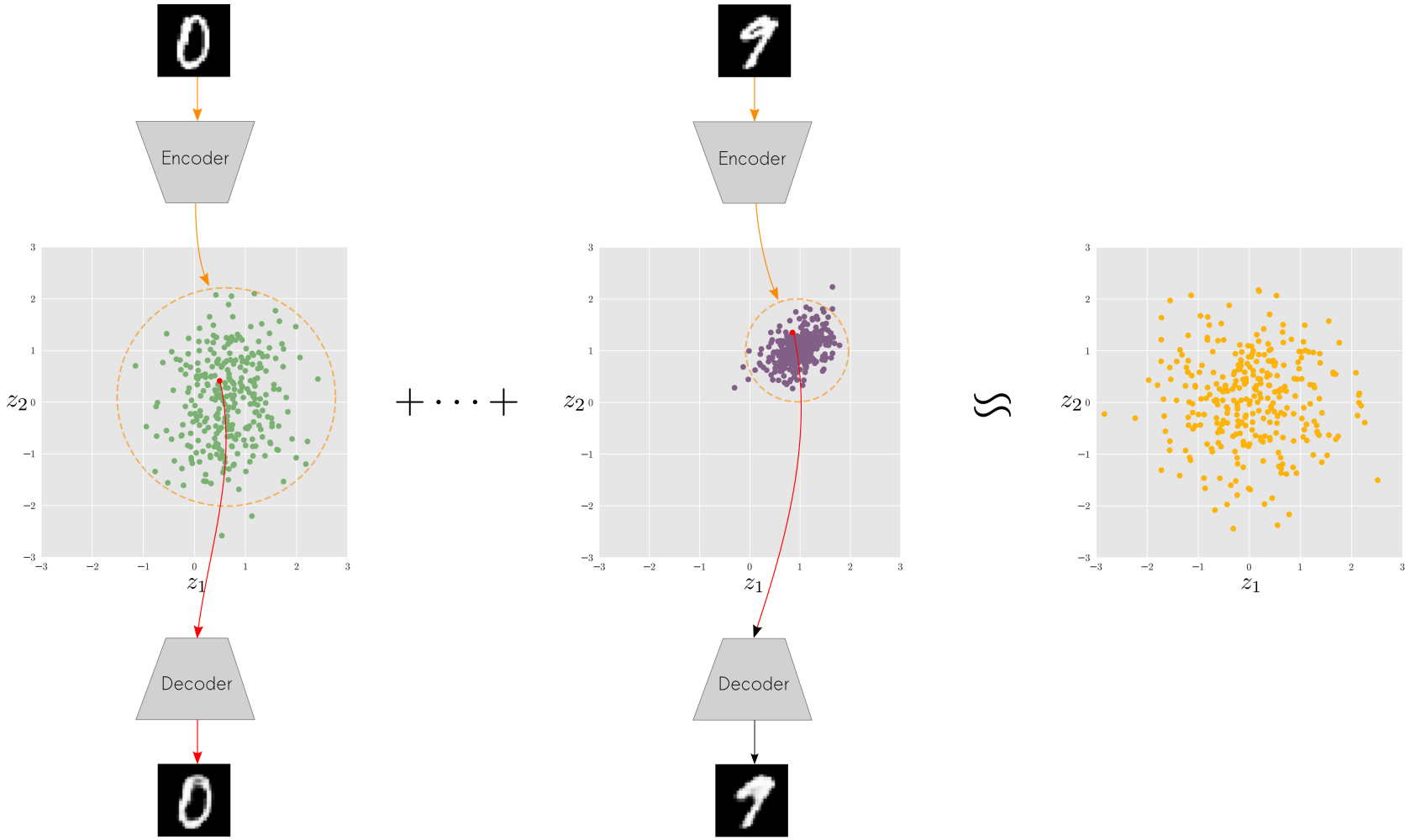

Figure 2.13: The sum over the latent space distributions of digits $0, 1, \ldots, 9$ approximates the multivariate isotropic Gaussian. [?]

Chapter 3

Conclusion

3.1 Summary of Thesis Achievements

Summary.

3.2 Applications

Applications.

3.3 Future Work

Future Work.

Bibliography

[1] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational Inference: A Review for Statisticians. jan 2016.

[2] D. M. P. U. Blei. Variational Inference, 2011.

[3] K. Burnham and D. Anderson. *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach (2nd ed)*, volume 172. 2002.

[4] B. Darrach. Meet Shakey, the First Electronic Person. *Life*, pages 58–68, 1970.

[5] C. Doersch. Tutorial on Variational Autoencoders. *arXiv*, pages 1–23, 2016.

[6] M. Garnelo, K. Arulkumaran, and M. Shanahan. Towards Deep Symbolic Reinforcement Learning. *arXiv Preprint*, pages 1–13, 2016.

[7] M. Hutter. *Universal Artificial Intelligence*, volume 1. 2005.

[8] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. *Iclr*, (Ml):1–14, 2014.

[9] W. Knight. AI Winter Isn't Coming. *MIT Technology Review*, 2016.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. *Advances In Neural Information Processing Systems*, pages 1–9, 2012.

[11] F.-F. S. U. Li, J. S. U. Johnson, and S. S. U. Yeung. Convolutional Neural Networks for Visual Recognition, 2016.

[12] C. Y. Liou, J. C. Huang, and W. C. Yang. Modeling word perception using the Elman network. In *Neurocomputing*, volume 71, pages 3150–3157, 2008.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[14] D. Ormerod. Lee Sedol plays the first move of game three against AlphaGo, 2016.

[15] E. U. o. T. Reingold. PSY371F Higher Cognitive Processes, 2001.

[16] R. Rosen. IBM's Deep Blue vs Gary Kasparov, 2012.

[17] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 07-12-June, pages 1–9, 2015.

[19] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8689 LNCS, pages 818–833, 2014.