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

Abstract

Reinforcement learning has achieved considerable success in recent years. One significant breakthrough was Google DeepMind's deep Q-network (DQN), which mastered a wide range of Atari 2600 games to a super-human level using only the pixels and score. However, reinforcement learning agents lack essential properties necessary for artificial general intelligence (AGI); namely they are slow to learn, unable to transfer knowledge between similar tasks and are unable to reason abstractly. The recent invention of deep symbolic reinforcement learning (DSRL), which is a marrying of deep reinforcement learning and symbolic logic, is a promising approach to overcome these drawbacks. However, this framework relies on the use of variational autoencoders to extract objects and their position in an unsupervised manner, which is currently an open problem in AI research. In this project, we pair a recent development of a scalable unsupervised method of learning generative factors in complex scenes, known as β-VAE, to fully-convolutional variational autoencoders in an attempt to preserve the spatial information necessary for DSRL.

TODO: Finish

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.

"The first principle is that you must not fool yourself - and you are the easiest person to fool."

R. P. Feynman

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 [17]. During the first period of hype from the 50s to the early 70s, Marvin Minsky made the following prediction: [9]

> "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 [24]. 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 [13].

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 [15], 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 [22, 29].

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. [27]

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. [23]

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: [13]

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 [13]. 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

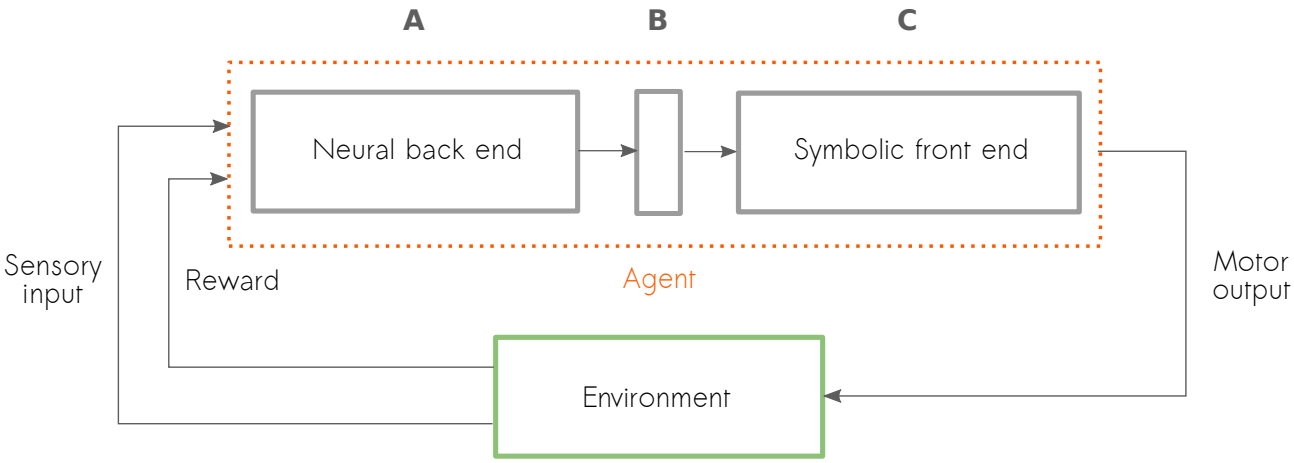


Figure 1.3: Overview of deep symbolic reinforcement learning system architecture. **A**: The neural back end maps high-dimensional raw input data to a compositionally structured symbolic representation. **B**: The compositionally structured symbolic representation. **C**: Reinforcement learning of mapping from symbolic representation to action with maximum expected reward over time. *Source: Garnelo et al.* [13].

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 ??.

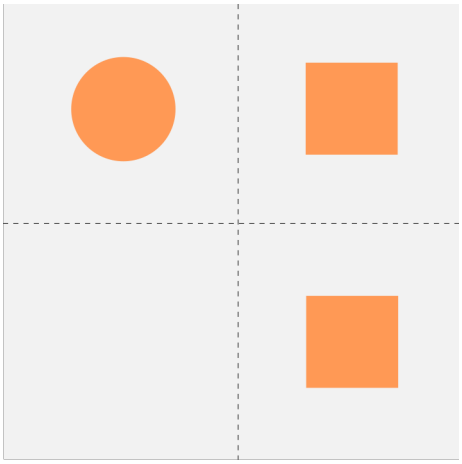


Figure 1.4: A toy example of a raw high-dimensional input.

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

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.

2.1 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.

2.1.1 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. [11]. 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.

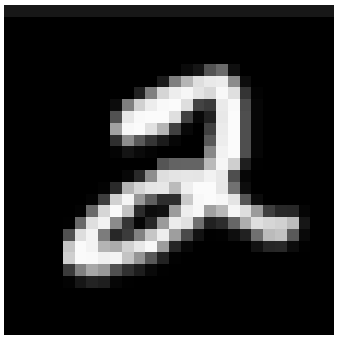


Figure 2.1

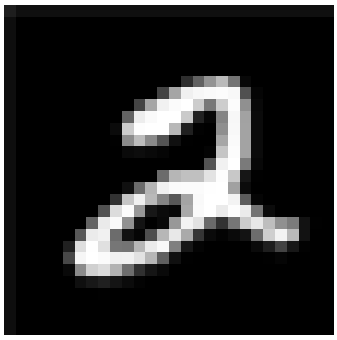


Figure 2.2

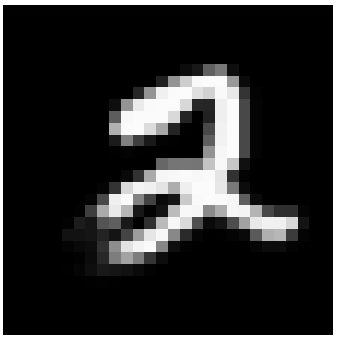


Figure 2.3

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

2.1.2 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.

2.2 Stella

Video games are becoming increasingly popular in the generation of data sets for machine learning purposes. One reason for this is that they do not inherit many operational drawbacks from real world data, such as noise, the stability of the camera or the observation of rare circumstances (the video game may just be queried for rare cases) [35, 25].

2.3 Arcade Learning Environment

[1]

2.4 Keras

Keras is a high-level deep learning library which interfaces with a Tensorflow or Theano backend. It provides a simple API which contains [6]

2.5 TensorBoard

Chapter 3

Related Work

3.1 Autoencoders

An autoencoder is a neural network that learns a compression algorithm for its input data in an unsupervised manner [21]. 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.

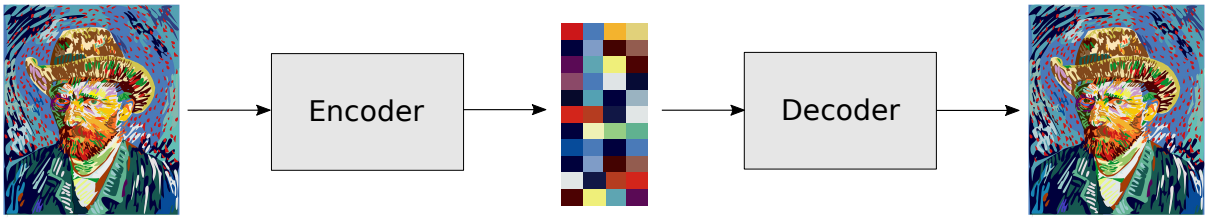


Figure 3.1: A black-box description of an autoencoder. The autoencoder learns the identity function, and in turn, the encoder and decoder learn suitable encoding and decoding algorithms respectively.

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.

3.1.1 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 3.2. Images must be flattened into vectors to be fed as input. Consequently, any spatial information is destroyed in dense feed-forward neural networks.

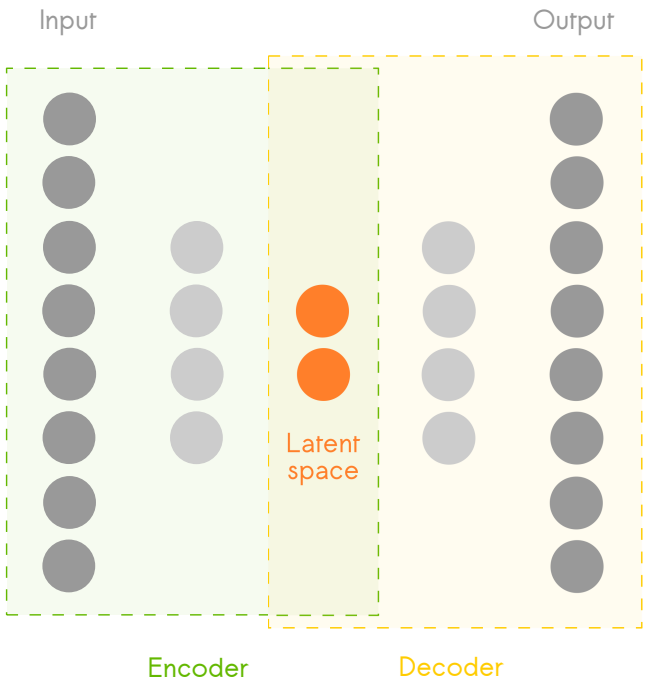


Figure 3.2: An example architecture of a fully-connected autoencoder. The latent space is constrained by having fewer neurons than the input and output layers.

An example architecture is given in Table 3.1, 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 3.3.

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

Table 3.1: A simple fully-connected autoencoder with one hidden layer. After 15 epochs, the validation score was recorded to be 71.94.

Figure 3.3: A collection of images from the MNIST data set and their respective reconstructions using the fully-connected autoencoder specified in Table 3.1. The original MNIST images are in odd columns, and their reconstructions to their immediate right.

3.1.2 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 3.4. To compare the fully-convolutional autoencoder to the fully-connected, we'll train the architecture in Table 3.2 on MNIST. As before, we'll compare the reconstructions to the originals, which can be found in Figure 3.5.

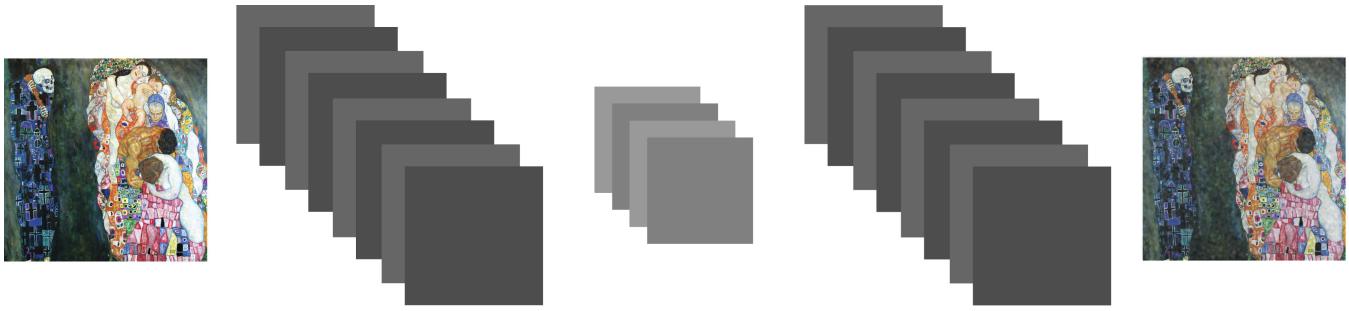


Figure 3.4: An example architecture of a fully-convolutional autoencoder. The latent space is constrained by reducing the number and/or size of the filters.

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 3.2: 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 [18, 34, 31]. 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 3.5: A collection of images from the MNIST data set and their respective reconstructions using the fully-convolutional autoencoder specified in Table 3.2. The original MNIST images are in odd columns, and their reconstructions to their immediate right.

3.2 Variational Autoencoders

[16]

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.

3.2.1 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 [16]:

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 $\boldsymbol{x}$

3.2.2 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:

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}) = \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})} \tag{3.1}$$

The marginal likelihood

$$p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \int p_{\boldsymbol{\theta}}(\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})d\boldsymbol{z} \tag{3.2}$$

involves an exponential-time integral over every combination of the latent variable $\boldsymbol{z}$, and is therefore computationally intractable [16]. 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.

3.2.3 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

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{3.3}$$

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) [4]. Using the KL divergence, our problem now amounts to the optimisation problem [20]:

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\theta}^*, \boldsymbol{\phi}^*} D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) \tag{3.4}$$

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

$$\begin{aligned}
D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) &= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})}\right] && (3.5)\\
&= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log_e q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\big] - \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log_e p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\big] && (3.6)\\
&= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log_e q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})\big] - \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x})}\right] && (3.7)\\
&= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}\right] + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}[\log_e p_{\boldsymbol{\theta}}(\boldsymbol{x})] && (3.8)\\
&= \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z},\boldsymbol{x})}\right] + \log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}) && (3.9)
\end{aligned}$$

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

$$f(\mathbf{E}\big[X\big]) \geq \mathbf{E}\big[f(X)\big] \tag{3.10}$$

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

$$\log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log_e \int p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) d\boldsymbol{z} \tag{3.11}$$

$$= \log_e \int p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}) \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} d\boldsymbol{z} \tag{3.12}$$

$$= \log_e \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \right] \tag{3.13}$$

$$\geq \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log_e \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \right] \tag{3.14}$$

$$:= \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) \tag{3.15}$$

Expression (3.14) is called the *ELBO* (short for expected lower bound) [2, 16].

How does the *ELBO* help us with minimising the KL divergence? First recall the alternative form of the KL divergence:

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})} \left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})} \right] + \log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{3.9}$$

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

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})) = -\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) + \log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}) \tag{3.16}$$

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* [3]. Now we may make a revision to our original optimisaiton problem (3.4). Our problem is written as [20]:

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \arg\max_{\boldsymbol{\theta}^*, \boldsymbol{\phi}^*} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) \tag{3.17}$$

3.2.4 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:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\right] \tag{3.18}$$

$$= -\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z})}\right] \tag{3.19}$$

$$= -\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p_{\boldsymbol{\theta}}(\boldsymbol{z})}\right] \tag{3.20}$$

$$= -\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log_e \frac{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}{p_{\boldsymbol{\theta}}(\boldsymbol{z})}\right] + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log_e p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \tag{3.21}$$

$$= -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \tag{3.22}$$

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

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big] \tag{3.23}$$

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:

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{3.24}$$

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,

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I}) \tag{3.25}$$

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,

$$D_{KL}(\mathcal{N}_0||\mathcal{N}_1) = \frac{1}{2}\left[\operatorname{tr}(\boldsymbol{\Sigma}_{\mathbf{1}}^{-1}\boldsymbol{\Sigma}_{\mathbf{0}}) + (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)^T\boldsymbol{\Sigma}_{\mathbf{1}}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0) - k + \ln\left(\frac{\det \boldsymbol{\Sigma}_{\mathbf{1}}}{\det \boldsymbol{\Sigma}_{\mathbf{0}}}\right)\right] \tag{3.26}$$

we may write the first term in closed-form:

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) = \frac{1}{2}\Big[\sum_{j=1}^{k}\sigma_j^{2(i)} + \sum_{j=1}^{k}\mu_j^{2(i)} - k - \ln\prod_{j=1}^{k}\sigma_j^{2(i)}\Big] \tag{3.27}$$

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

$$\mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big] \tag{3.28}$$

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) [20].

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

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \frac{1}{2}\Big[\sum_{j=1}^{k}\sigma_j^{2(i)} + \sum_{j=1}^{k}\mu_j^{2(i)} - k - \ln\prod_{j=1}^{k}\sigma_j^{2(i)}\Big] + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big] \tag{3.29}$$

3.2.5 Implementing the Variational Autoencoder

The variational autoencoder uses a neural network for the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ [16]. 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

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I}) \tag{3.25}$$

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 (3.6) [20].

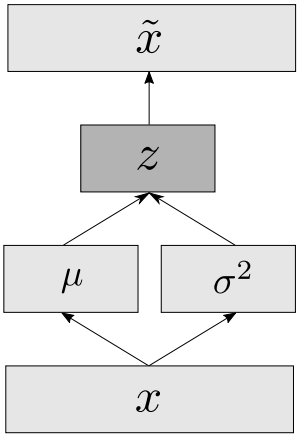


Figure 3.6: A naïve implementation of the variational autoencoder. The input $\boldsymbol{x}$ is mapped to intermediate layers taking the values of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$. The latent variable $\boldsymbol{z}$ is then sampled from the probabilistic encoder $\boldsymbol{z} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. 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 $\boldsymbol{\phi}$ [16]. 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

$$\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon} \qquad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}) \tag{3.30}$$

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" [16]. The reparameterisation trick allows backpropagation to be used, and thus completes our method of solving optimisation problem (3.17).

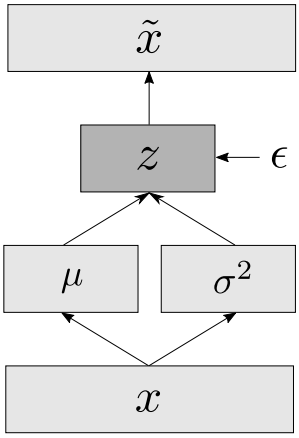


Figure 3.7: A viable implementation of the variational autoencoder. Sampling from the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is simulated by evaluating $\boldsymbol{z} = g_{\boldsymbol{\phi}}(\boldsymbol{x}, \boldsymbol{\epsilon}) = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}$.

3.2.6 Intuition Behind the Variational Autoencoder

In this section, we'll develop the intuition behind the variational autoencoder. We've been considering the data set $X = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N}$. Now we'll choose X to be the MNIST data set (the vector $\boldsymbol{x^{(i)}}$ now corresponds to a flattened MNIST image of length $28 \times 28 = 784$). For visualisation purposes, we'll also take the latent space dimension to be $k = 2$, that is, $\boldsymbol{z} = (z_1, z_2)$.

The probabilistic encoder

$$q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)}) = \mathcal{N}(\boldsymbol{\mu}^{(i)}, \boldsymbol{\sigma}^{2(i)}\boldsymbol{I}) \tag{3.25}$$

gives a normal distribution over the values of the latent variable $\boldsymbol{z}^{(i)}$ given data point $\boldsymbol{x}^{(i)}$. A sample is drawn from this normal distribution and passed through to the decoder. The decoder then reconstructs the original image from the latent representation. Since we've chosen our latent space to be two-dimensional, it's possible to visualise this process, which is done in Figure (3.8). It's then possible to compare the reconstruction to the original (using an appropriate reconstruction loss function), and therefore to train the autoencoder as a whole.

Recall that we chose the prior to be the standard multivariate Gaussian

$$p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{3.24}$$

We also derived the loss function to be

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}|\boldsymbol{z})\big] \tag{3.23}$$

which we want to maximise. Since

$$D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) \geq 0 \tag{3.31}$$

maximising the $ELBO$ is eqiuvalent to ruducing the KL divergence between the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})$ and the prior $p_{\boldsymbol{\theta}}(\boldsymbol{z})$. Therefore after training, we expect that the probabilistic encoder should map samples to the standard multivariate Gaussian. (In practice, $\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}^{(i)}) > 0 \quad \forall i$, so we therefore only expect it to be *approximately* mapped to the standard Gaussian). Decoding samples from the prior should correspond to meaningful reconstructions.

TODO: FINISH

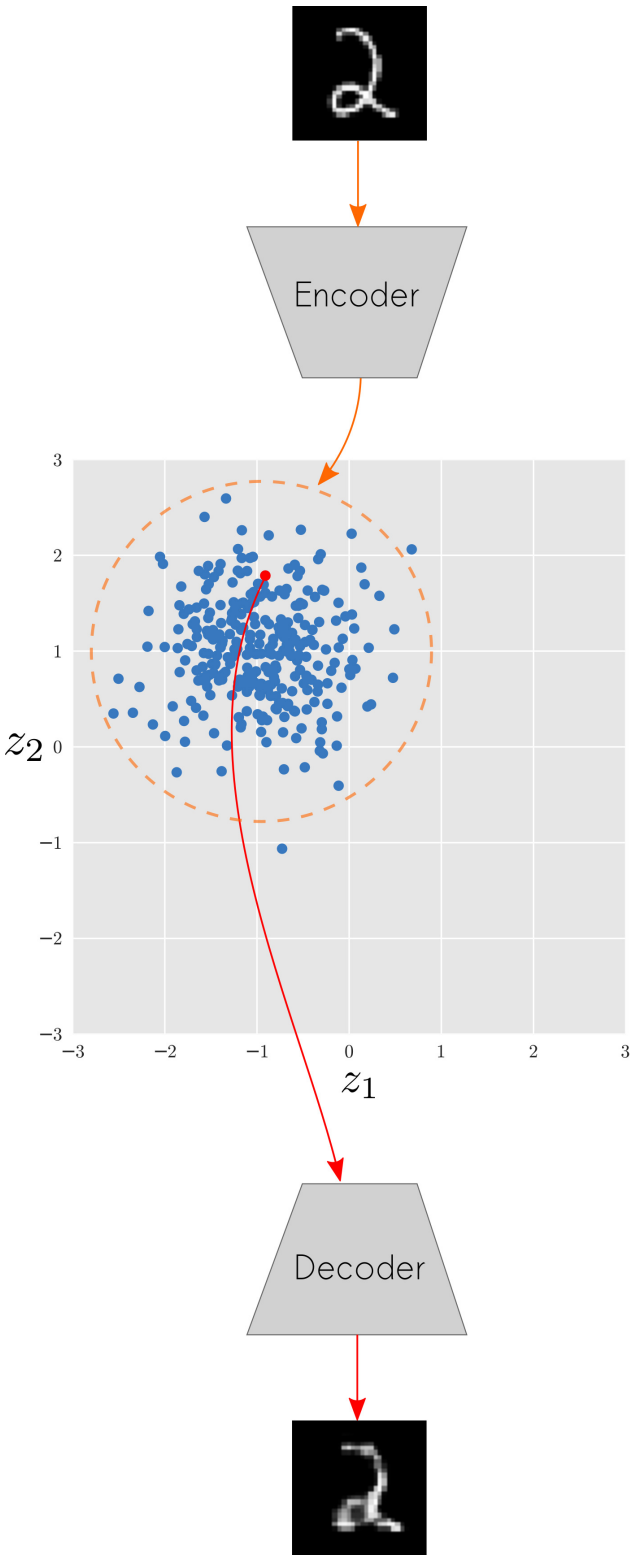


Figure 3.8: The encoder takes a data point and returns a normal distribution (orange); some samples of which are shown (blue). A sample is drawn from the normal distribution (red) and decoded. [12]

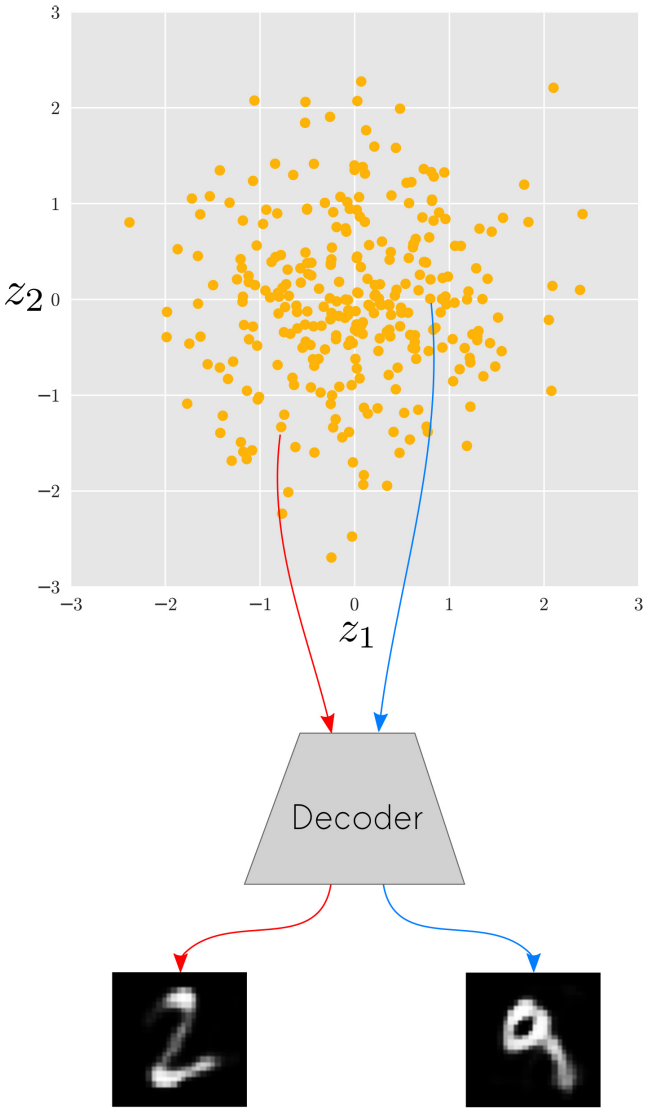


Figure 3.9: The prior distribution should approximate the standard multivariate Gaussian $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. Samples of the prior are shown (yellow); two of which are decoded (red and blue). [12]

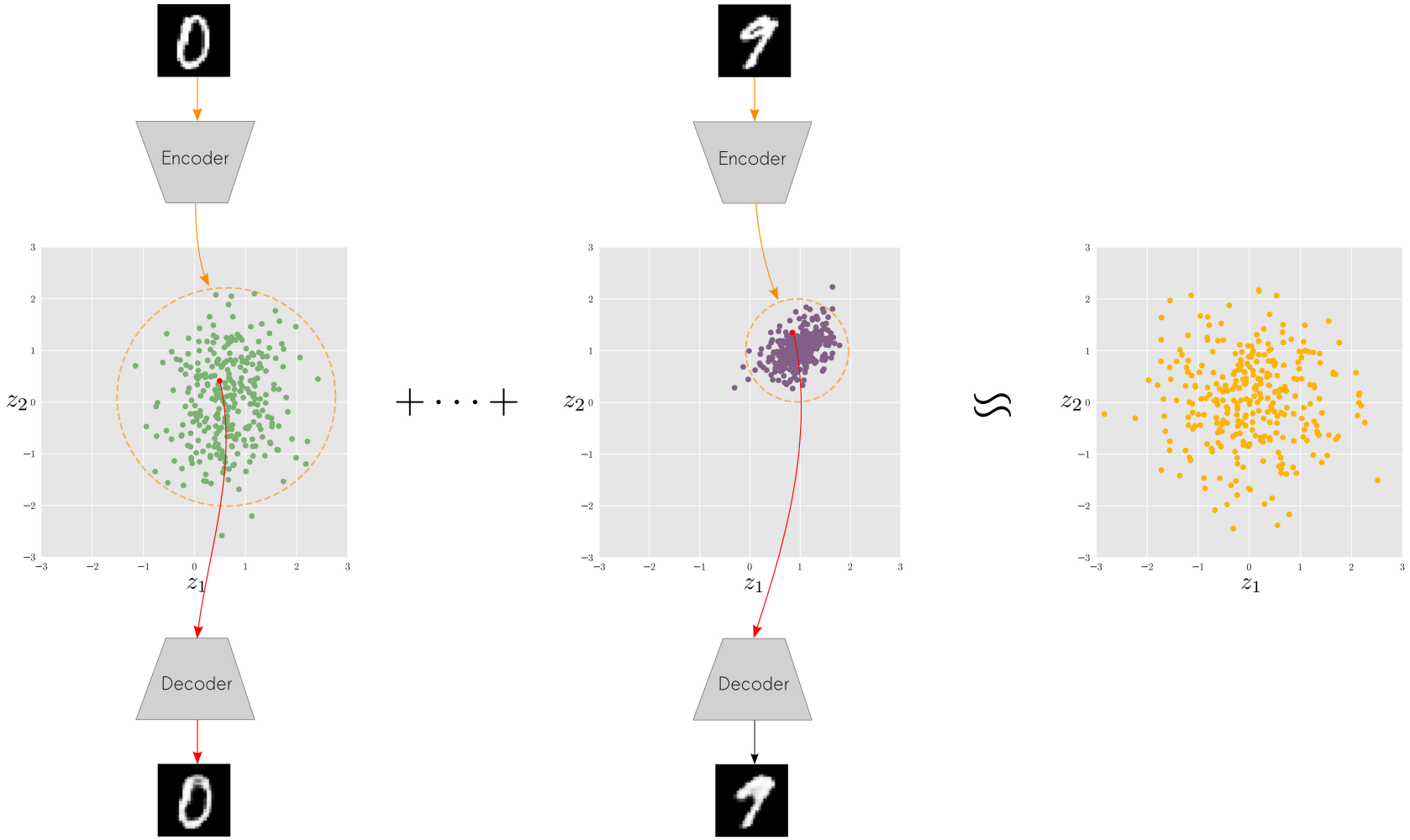


Figure 3.10: The sum over the latent space distributions of all data points $\boldsymbol{x}^{(i)}$ approximates the multivariate isotropic Gaussian. [12]

3.3 Disentangled Representations

[14, 33]

3.4 Unsupervised Learning of Generative Factors

Learning disentangled generative factors of a scene in an unsupervised manner is an open challenge in AI research. Although many attempts have been made, they have not scaled well [33, 28, 10, 32, 7]. However, two recent advancements have made significant headway: β-VAE and InfoGAN [5, 33].

3.4.1 InfoGAN

InfoGAN is an information theoretic extension of the Generative Adversarial Network, which has convincingly learnt generative factors of multiple data sets in an unsupervised manner, including MNIST, 3D faces and 3D chairs [5].

However, InfoGAN is sensitive to the choice of the prior distribution, and therefore requires a priori knowledge of the data set, as well as the number and type of generative factors [33]. Ideally, the number of generative factors would be inferred and not made explicit by the designer. On this basis, we can conclude that future work is needed.

3.4.2 β-VAE

β-VAE is the first method to overcome what InfoGAN could not: to learn the (unspecified amount of) generative factors of a data set in an unsupervised manner. First a formal derivation of β-VAE will be proposed, then a number of experimental results.

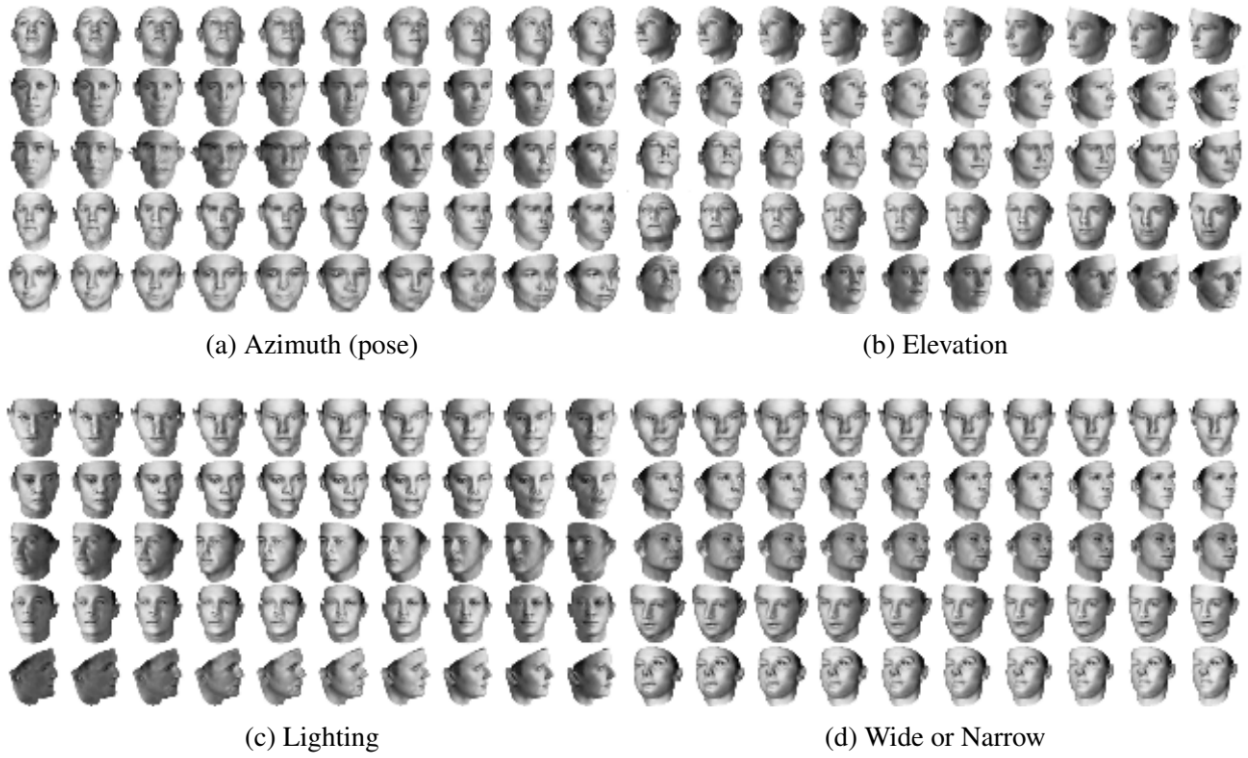


Figure 3.11: InfoGAN convincingly learns the underlying generative factors in the 3D face data set. Rows correspond to a data point and columns the value of the latent variable (varied from -1 to 1). Each section (a), (b), (c) and (d) consider a different latent variable. [5]

Derivation

Recall that for the variational autoencoder, we assumed the data point $\boldsymbol{x}$ was generated by a process involving a latent variable $\boldsymbol{z}$ and conditional $p(\boldsymbol{x}|\boldsymbol{z})$. For β-VAE we make a slightly different assumption: data point $\boldsymbol{x} \in \mathbb{R}^N$ was generated by conditionally independent factors $\boldsymbol{v} \in \mathbb{R}^K$ s.t. $p(\boldsymbol{v}|\boldsymbol{x}) = \prod_k p(v_k|\boldsymbol{x})$, conditionally dependent factors $\boldsymbol{w} \in \mathbb{R}^H$ and the conditional $p(\boldsymbol{x}|\boldsymbol{v}, \boldsymbol{w})$. In short, this is written as

$$p(\boldsymbol{x}|\boldsymbol{v}, \boldsymbol{w}) = Sim(\boldsymbol{v}, \boldsymbol{w}) \tag{3.32}$$

where Sim is short for a true world simulator. β-VAE seeks to represent these generative factors in the latent variable $\boldsymbol{z} \in \mathbb{R}^M$, such that

$$p(\boldsymbol{x}|\boldsymbol{z}) \approx p(\boldsymbol{x}|\boldsymbol{v}, \boldsymbol{w}) = Sim(\boldsymbol{v}, \boldsymbol{w}) \tag{3.33}$$

Note that, unlike InfoGAN, β-VAE does not specify the number independent generative factors K; instead it is inferred. However, in order for $\boldsymbol{z} \in \mathbb{R}^M$ to learn the conditionally independent factors $\boldsymbol{v} \in \mathbb{R}^K$, we must assert that $M \geq K$.

As before, the posterior $p_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ is intractable, which motivates the introduction of the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. A reasonable objective is to find the most likely parameters of the model over all latent variables that produced the observed data. This is summarised more precisely as the optimisation problem

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^* = \max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \tag{3.34}$$

A constraint is added to control the capacity of information in the latent space and encourage the factors $\boldsymbol{v}$ to be learnt in a disentangled manner. The probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ is pressured to be close to an isotropic Gaussian, implicitly applying an independence pressure. By using the KL divergence, we form the optimisation problem

$$\begin{aligned} \boldsymbol{\theta}^*, \boldsymbol{\phi}^* &= \max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \\ &\text{s.t. } D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})) < \epsilon \end{aligned} \tag{3.35}$$

The Lagrangian of optimisation problem (3.35) is

$$\mathcal{F}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] - \beta(D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})) - \epsilon) \tag{3.36}$$

Since $\beta, \epsilon > 0$,

$$\mathcal{F}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) \geq \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] - \beta D_{KL} q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z})) \tag{3.37}$$

By applying pressure for $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ to be close to the prior $p(\boldsymbol{z})$, the latent space learns the conditionally independent factors $\boldsymbol{v}$ in a different subset of $\boldsymbol{z}$ than the conditionally dependent factors $\boldsymbol{w}$. β controls the degree of this pressure. Also note that $\beta = 1$ yields the $ELBO$ formulated earlier (3.22), and $\beta = 0$ yields the maximum likelihood case. It's hypothesised

that disentangled representations are learnt for $\beta > 1$.

As the prior needs to be isotropic, it's often chosen to be the standard multivariate Gaussian:

$$p(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{3.38}$$

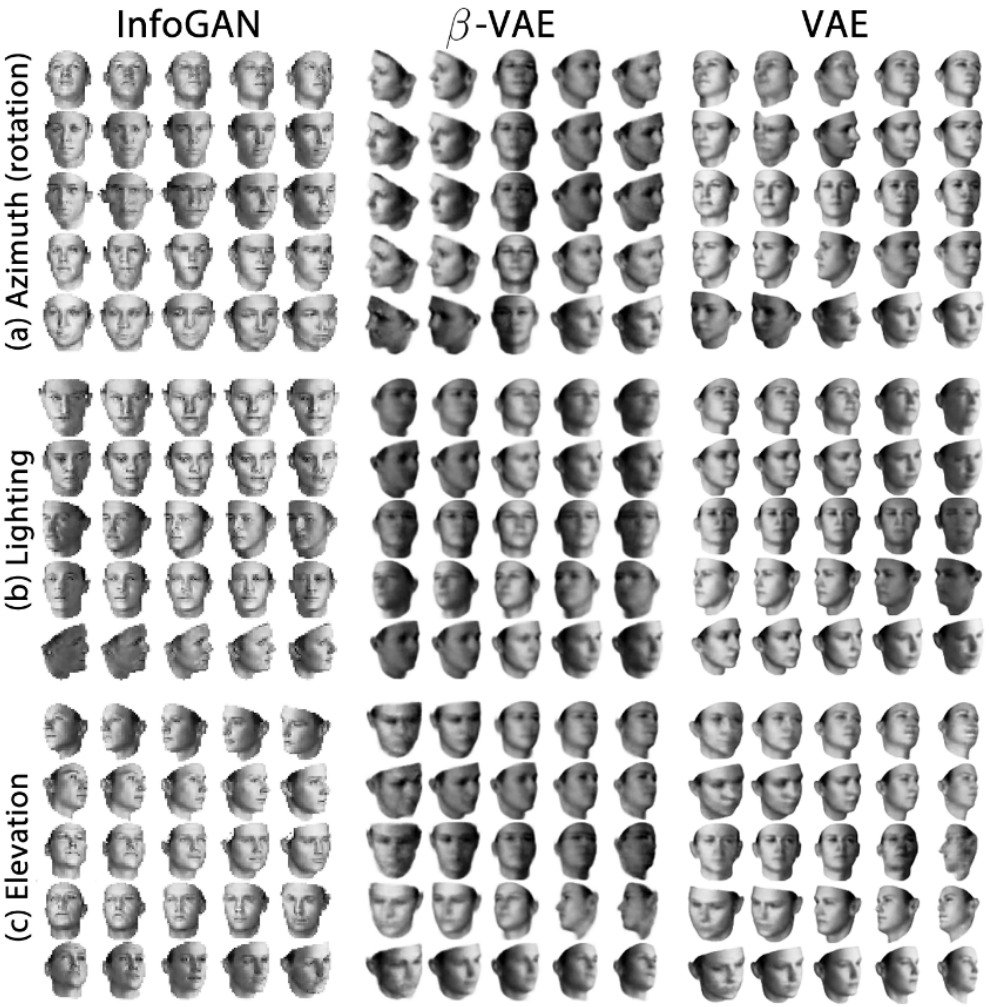


Figure 3.12: A comparison of InfoGAN, β-VAE ($\beta = 20$) and VAE on the 3D face data set. Different latent variables are varied for sections (a), (b) and (c). All models learnt lighting and elevation, but only InfoGAN and β-VAE managed to continuously vary the azimuth. [33]

3.5 Improving Sampling from Generative Autoencoders with Markov Chains

A generative autoencoder may be defined as an autoencoder that pressures its latent distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ to match a given prior $p(\boldsymbol{z})$ [8]. As discussed earlier, the KL divergence term in the $ELBO$ applies this pressure:

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = -D_{KL}(q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})||p_{\boldsymbol{\theta}}(\boldsymbol{z})) + \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] \tag{3.23}$$

In Section (3.2) it was assumed that the data set X was generated by

$$\boldsymbol{z}^{(i)} \sim p(\boldsymbol{z}) \qquad \boldsymbol{x}^{(i)} \sim p_{\boldsymbol{\theta}^*}(\boldsymbol{x}|\boldsymbol{z}^{(i)}) \tag{3.39}$$

where $p(\boldsymbol{z})$ is a parameterless prior distribution and $p_{\boldsymbol{\theta}^*}(\boldsymbol{x}|\boldsymbol{z}^{(i)})$ is the true conditional. As the true set of parameters $\boldsymbol{\theta}^*$ is unknown to us, we may attempt to generate samples with learnt parameters $\boldsymbol{\theta}$:

$$\boldsymbol{z}^{(i)} \sim p(\boldsymbol{z}) \qquad \boldsymbol{x}^{(i)} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}^{(i)}) \tag{3.40}$$

Now suppose

$$\int q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} = \hat{p}(\boldsymbol{z}) \tag{3.41}$$

where $p(\boldsymbol{x})$ is the distribution that generated data set X. The generative procedure (3.40) makes the erroneous assumption that $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ perfectly matches the prior $p(\boldsymbol{z})$, that is, $p(\boldsymbol{z}) = \hat{p}(\boldsymbol{z})$ [8]. Since only pressure was applied to match the two, this is clearly not true in general. That is, in general,

$$p(\boldsymbol{z}) \neq \hat{p}(\boldsymbol{z}) \tag{3.42}$$

Therefore we have

$$\int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\hat{p}(\boldsymbol{z})d\boldsymbol{z} = p(\boldsymbol{x}) \neq \int p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})d\boldsymbol{z} \tag{3.43}$$

which suggests that the generative procedure (3.40) does not produce samples from $p(\boldsymbol{x})$ exactly. To sample from $p(\boldsymbol{x})$, a new generative procedure is proposed [8]:

$$\boldsymbol{z}^{(i)} \sim \hat{p}(\boldsymbol{z}) \qquad \boldsymbol{x}^{(i)} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}^{(i)}) \tag{3.44}$$

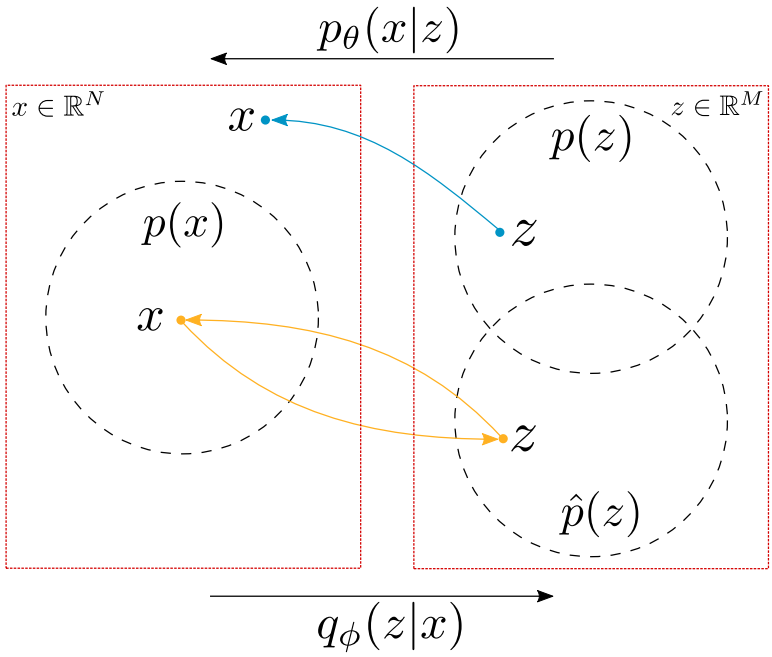
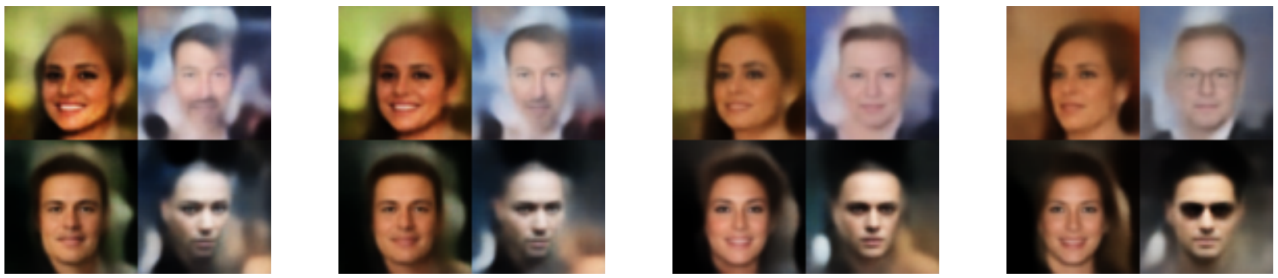


Figure 3.13: The probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ maps a given data point $\boldsymbol{x}$ to the unknown distribution $\hat{p}(\boldsymbol{z})$. The probabilistic decoder $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ is trained to map samples from $\hat{p}(\boldsymbol{z})$ back to $p(\boldsymbol{x})$, since its inputs are drawn from the probabilistic encoder $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$. A sample from the prior $p(\boldsymbol{z})$ will not be mapped back by $p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$ to $p(\boldsymbol{x})$ exactly if $p(\boldsymbol{z}) \neq \hat{p}(\boldsymbol{z})$. [8]

As $\hat{p}(\boldsymbol{z})$ is unknown, it is not obvious how to sample from generative autoencoders using procedure (3.44). However, it is possible to formulate a Markov chain Monte Carlo (MCMC) chain that starts with an arbitrary latent variable $\boldsymbol{z}_{t=0}$ and converges with $\boldsymbol{z}_{t\to\infty} \sim \hat{p}(\boldsymbol{z})$ [8]. This is done by successively encoding and decoding the same sample as follows:

$$\boldsymbol{x}_{t=k+1} \sim p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}_{t=k}) \qquad \boldsymbol{z}_{t=k+1} \sim q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x}_{t=k+1}) \tag{3.45}$$

where $\boldsymbol{z}_{t=0}$ is arbitrarily chosen.



(a) VAE (initial) (b) VAE (1 step) (c) VAE (5 steps) (d) VAE (10 steps)

Figure 3.14: Samples from a variational autoencoder trained on the CelebA data set after $t = 0, 1, 5$ and 10 steps of the generative procedure (3.45). The MCMC chain was initialised with a sample from the prior $\boldsymbol{z}_{t=0} \sim p(\boldsymbol{z})$, which often improves the quality of the samples. [8]

3.6 Regularizing CNNs with Locally Constrained Decorrelations

[26]

3.7 Batch Normalisation

...

Chapter 4

Implementation

4.1 Dimensionality Reduction

Machine learning with high-dimensional data, such as images, is often computationally intensive. Atari 2600 frames are RGB images (3 channels) of size 210×160, expressed in shorthand as $(3, 210, 160)$. (This data is much higher in dimensionality than MNIST, which has dimensions $(1, 28, 28)$). Considering the training sets considered in this project are of hundreds of thousands of images, data points of dimension $(3, 210, 160)$ are too computationally intensive with the best hardware available ($2 \times$ NVIDIA Tesla K80 GPU Accelerators). It is therefore necessary to reduce the dimensionality of our data set. Google DeepMind's *Human-level Control Through Deep Reinforcement Learning* [22] made use of Stella, as we have, and have convincingly struck a reasonable balance between resolution and dimensionality reduction. This section will be a short but necessary mentioning of the pre-processing pipeline used to generate the data sets in later chapters.

4.1.1 Pre-processing Pipeline

Ensuring Object Persistence

Atari 2600 games can only store a limited number of sprites per frame due to the limitations in hardware during its development [22]. This is an issue as some objects that appear in one frame fail to appear in the next. To solve this lack of object persistence, even and odd frames are combined by taking the maximum over each channel (RGB). By taking the maximum, we ensure that any object present in one frame is also present in the other.

Extracting Luminance and Cropping

The luminance Y is then extracted from the RGB image [30]:

$$Y = 0.2126 \times R + 0.7152 \times G + 0.0722 \times B \tag{4.1}$$

The resultant greyscale image of shape $(1, 210, 160)$ is cropped to $(1, 84, 84)$.

File Formats

Image formats were discovered to be more important than originally thought. Empirically, PNG or GIF formats were reasonable formats, but using JPEG often resulted in distortions near the perimeter of objects. An example of this effect is shown in Figure (4.2).

4.2 Qualitative Assessment Using GUIs

- To qualitatively assess the significance of a latent neuron in the final reconstruction, we change its value over a given range and inspect the reconstructions

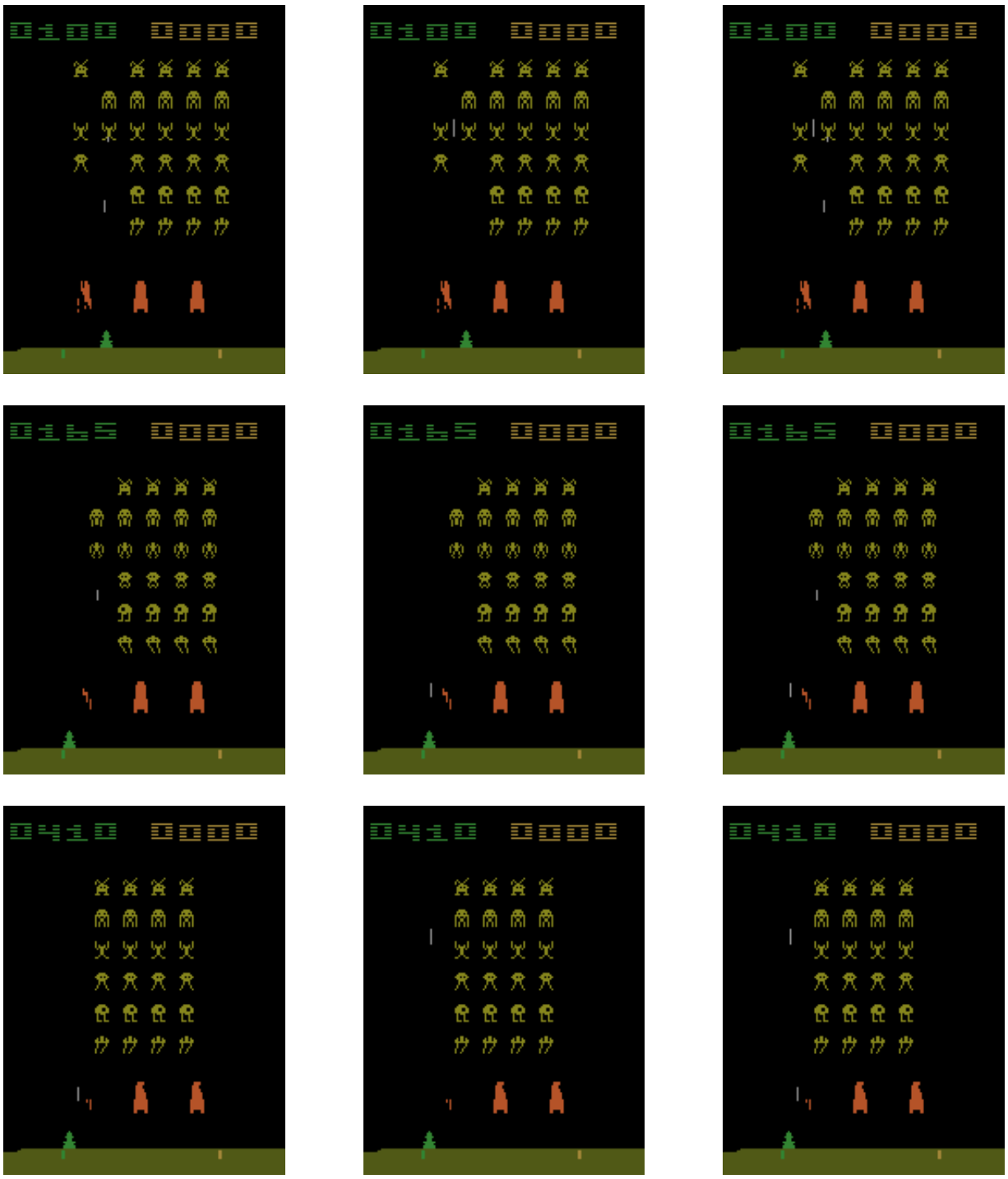


Figure 4.1: A collection of frames captured from Space Invaders emulated on Stella. **Left column:** an even frame. **Middle column:** the (odd) frame following. **Right column:** Combining the even and odd frames by taking the maximal value over each channel (RGB). Clearly the bullets visible in one frame fail to persist in the next. As mentioned, this is due to the limited number of sprites Atari 2600 can load in a single frame.

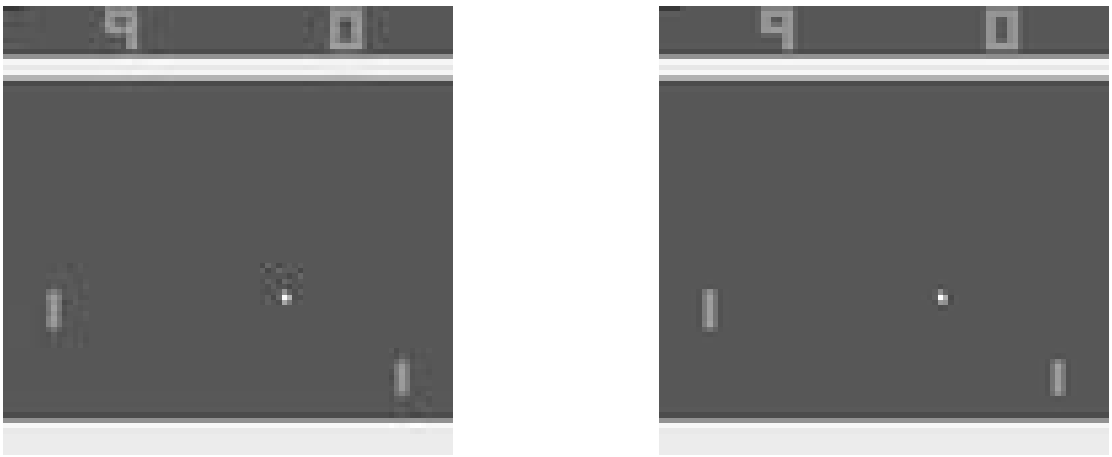


Figure 4.2: Pre-processed frames captured from Pong emulated on Stella. These frames were originally 84×84, but are printed here as 168×168 to emphasise distortions. **Left:** The JPEG format distorts the ball, paddle and score sprites. **Right:** The PNG format displays the frame without such distortions.

- In order to speed up this process, Tkinter was used to develop a GUI with a sliding bar corresponding to the latent neuron's value
- This real-time reconstruction allows for much quicker feedback

4.3 Training and Validation Data Generators

- Using generators from directories

4.4 Ensuring Numerical Stability in the Latent Space

- Use log variance instead of variance to spread values evenly, which helps with backpropagation, as it's easier to learn with an approximately equal step size between values rather than those that are either clumped or sparse
- Since log is monotonic, the value ordering is invariant, which allows back propagation to implicitly optimise the variance σ^2
- State implemented ELBO

4.5 Activation Functions in the Latent Space

- Conventionally activation functions are used in the hidden layers, such as sigmoid or, more commonly, ReLU.
- However, we must be careful not to mindlessly follow this trend.
- It makes no sense to use a ReLU activation function on the mean layer of the variational autoencoder. The mean $\mu \in \mathbf{R}$ is unconstrained, and hence a linear activation function is appropriate.
- Stuff about log variance - also use linear activation function

4.6 Keras Callbacks

- LR annealing
- Checkpointing
- Reduce LR on plateau
- Early stopping

Chapter 5

Methods

> "It is not unscientific to make a guess, although many people who are not in science think it is."
>
> *R. P. Feynman*

- Need to preserve spatial information for symbolic front-end
- Need disentangled latent space for symbolic front-end
- This may be summarised as needing to be able to deduce the object type and its location solely from the latent space representation
- Explain degree of freedom here (disentangled filters, or disentangled neurons, or something else?)
- This is unexplored territory - we decide what the reasonable architecture should be, then evaluate it in the results section
- As far as we know, this section contains the first considerations of a convolutional variational latent space

5.1 Single Latent Filter

- As mentioned, the type and position of an object must be preserved in the latent space
- This may be achieved by using a single latent filter, with the object type corresponding to the value of the weighted sum of the neuron.
- Recall that no activation function is used for these layers, and therefore the object type is represented by an unbounded real number
- Although it's suspected that this may be too much to ask from the network, we still grant it the possibility to surprise us

5.1.1 Architecture

- The input image is passed through convolutional layers to build increasingly meaningful hidden representations
- The convolutional mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$, both of shape $(1, m, n)$, are sampled using ... cite reparameterisation trick here... (so we may use backpropagation) to give the latent space of shape $(1, m, n)$.
- The corresponding deconvolutional layers are applied to reconstruct the original image
- The proposed architecture is shown in Figure (5.1)

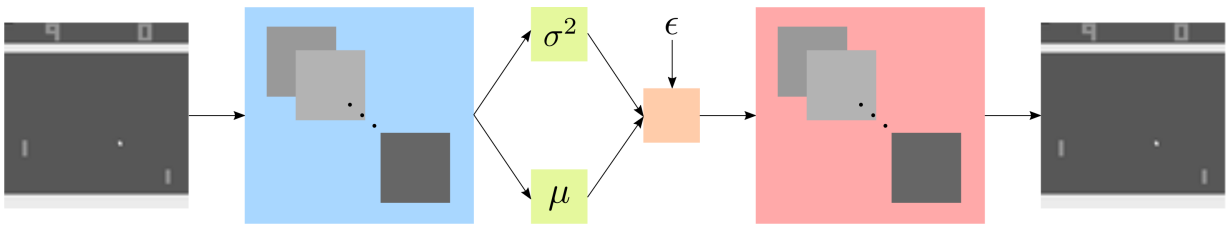


Figure 5.1: The fully-convolutional single latent filter architecture. **Blue:** An arbitrary amount of convolutional layers. **Green:** The latent mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$, which are both single filters of shape $(1, m, n)$. **Orange:** A single latent filter of shape $(1, m, n)$ sampled component-wise from $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$. **Red:** The corresponding deconvolutional layers.

5.1.2 Neuron-Level Decoupling

- As with the variational autoencoders seen earlier, we have a reconstruction loss term in the loss function

- Learning the identity function ensures a meaningful lower-dimensional representation is learnt in the latent space

- However, as we only want values of high magnitude where there are objects, we need to reduce the reundancy in the latent space

- Therefore we need to decide on the appropriate term in the loss function to reduce redundancy

- The latent filter of shape $(1, m, n)$ is flattened to a $m \times n$ vector

- This flattened representation of the single latent filter is used then pressured to be close to an isotropic Gaussian with KL divergence ...equation here...

- This pressuring reduces redundancy in the latent space as if neighbouring pixels

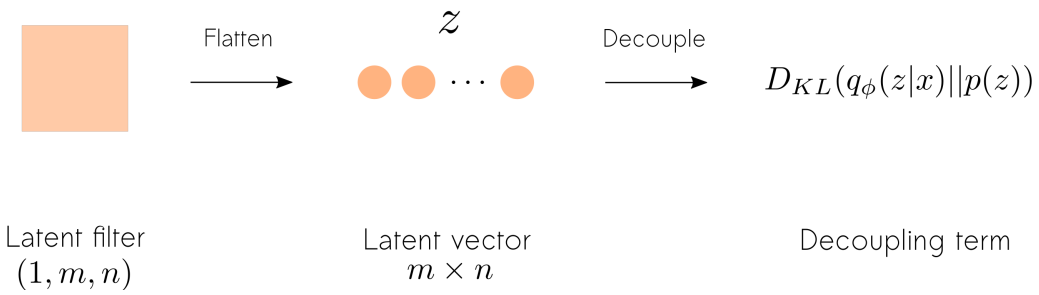


Figure 5.2: Caption.

5.2 Multiple Latent Filters

- As mentioned, a single latent filter seems unlikely to be able to represent a types of objects in the original scene
- This leads to the idea of pressuring filters to learn at most one object type
- For scenes with more than one object type, the latent space must have multiple features

5.2.1 Architecture

TODO: Finish subsection

- The architecture is the same as the fully-convolutional single lantent filter architecture, but with one difference in the latent space.

- The convolutional mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$, both of shape (k, m, n), are sampled using ... cite reparameterisation trick here... (so we may use backpropagation) to give the latent space of shape (k, m, n).
- The proposed architecture is shown in Figure (5.3)

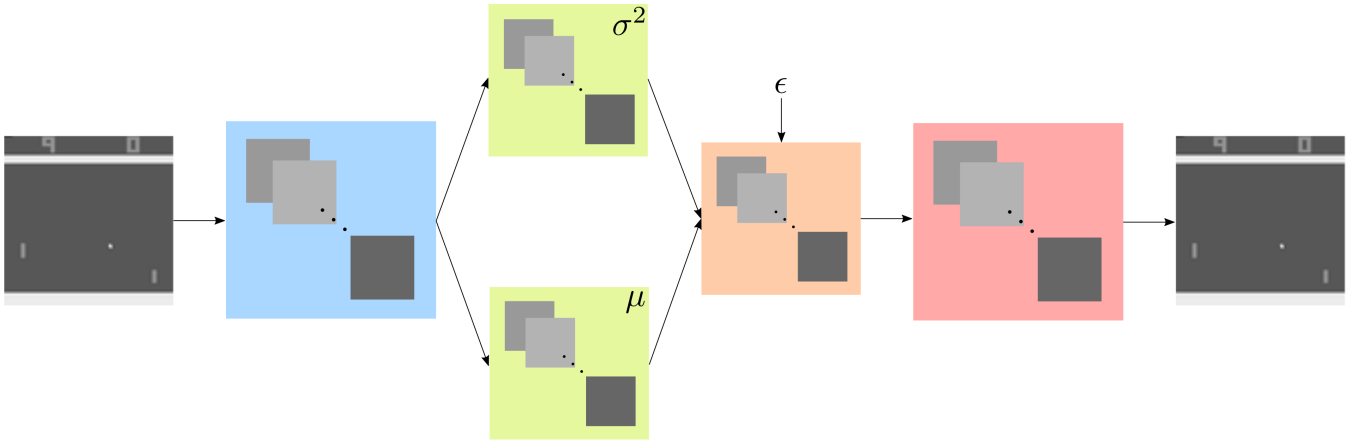


Figure 5.3: The fully-convolutional multiple latent filter architecture. **Blue:** Unchanged. **Green:** The latent mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma}^2$, which are both single filters of shape (k, m, n). **Orange:** A single latent filter of shape (k, m, n) sampled component-wise from $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$. **Red:** Unchanged.

- By the same reasoning for the previous architecture, we keep the reconstruction loss term.
- Need to decide on the appropriate term in the loss function to reduce redundancy
- Here some creativity is needed on how to reduce redundancy over the filters
- We present three methods of decoupling the latent volume: ...list...

5.2.2 Neuron-Level Decoupling

- The latent filter of shape (k, m, n) is flattened to a $k \times m \times n$ vector
- This flattened representation of the single latent filter is used then pressured to be close to an isotropic Gaussian with KL divergence ...equation here...
- This pressuring reduces redundancy among all pixels in the latent volume

- The loss function can therefore be written as ...loss function...

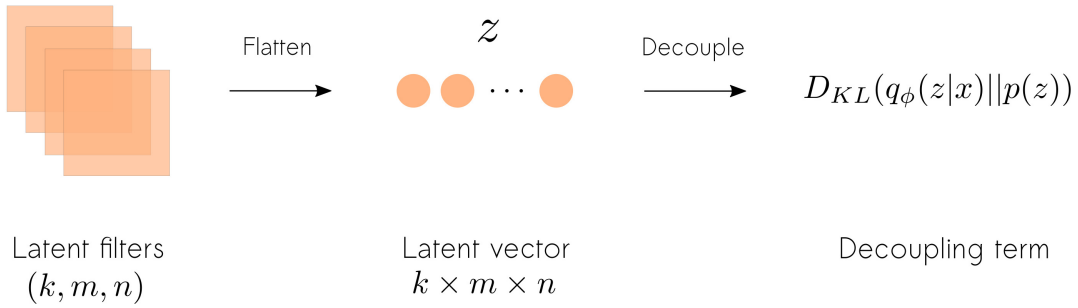


Figure 5.4: Caption.

5.2.3 Naïve Filter-Level Decoupling

- The k filters of the (k, m, n)-dimensional latent space are averaged across their activations and stored in a vector $\bar{\boldsymbol{z}}$
- We've reduced our problem to one solved in β-VAE!
- To disentangle the average activations over filters, we pressure the vector $\bar{\boldsymbol{z}}$ to be close to an isotropic Gaussian
- This pressuring reduces redundancy the average activations among filters
- This achieves the desired outcome because: - if two filters that have learnt the same object type, one may be pressured to match the prior (or "forgotten"), as the same information about the original scene is encoded in the latent space

- The loss function then can be written as ...loss function...

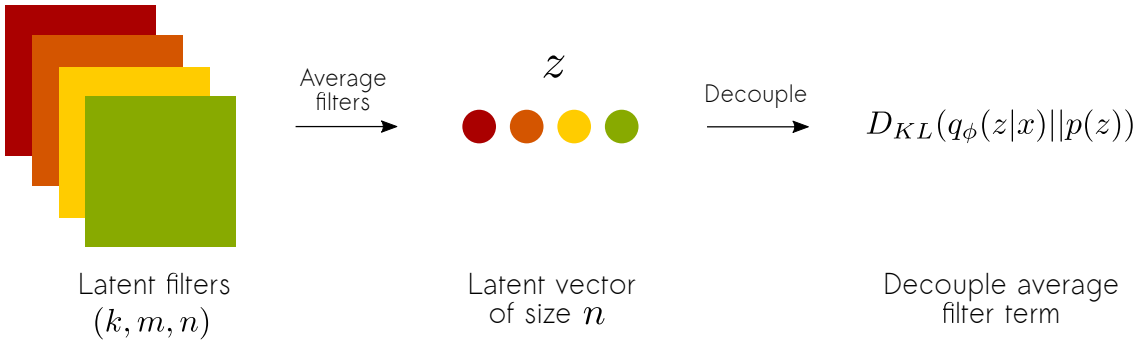


Figure 5.5: Caption.

5.2.4 Weighted Filter-Level Decoupling

- Before we have taken the sum of the pixel-wise loss using binary cross entropy and the sum of the average activations over latent filters

- This may be a bad design choice, as the KL loss is extremely small compared to the reconstruction loss

- For example, suppose we use an input image of 28×28 and 16 latent filters. Further assume that, on average, the reconstruction loss of one pixel is 0.5 and the average activation is 4 (both are reasonable for the MNIST data set). Then the reconstruction loss is 28 * 28 * 0.5 = 392 compared to the KL loss of 16 * 2 = 32 - approximately an order of magnitude difference.

- Instead, if we weight the average binary cross entropy and average filter activations by the number of pixels taking in the respective averages, we may get a more balanced loss function

- Therefore our final loss function is ...

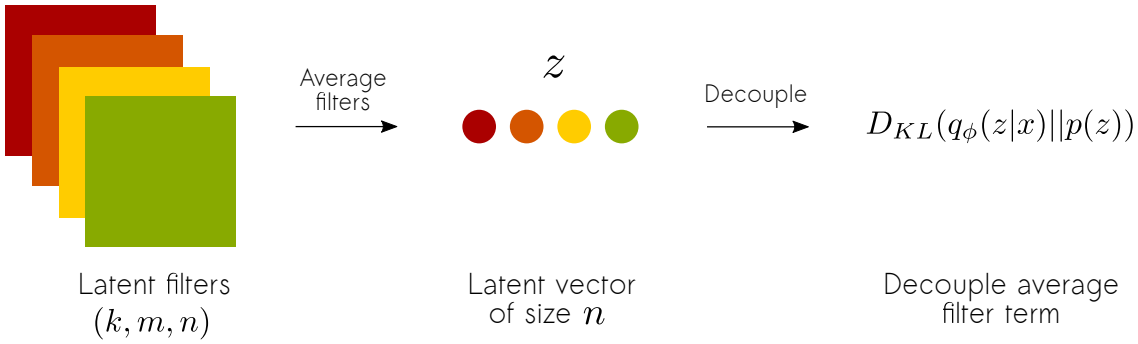


Figure 5.6: Caption.

5.3 Separating Colour Spaces

- As the task of unsupervised object recognition in fully-convolutional variational autoencoders is an open problem, it's reasonable to start by solving easy recognition tasks
- Since sprites are of different colours in Atari games, maybe we could leverage this property by including the colour channels of the frame as input to make the separation of objects easier
- An example for a frame of Space Invaders is shown in Figure (5.12)
- This technique may be applied to all architectures listed here.

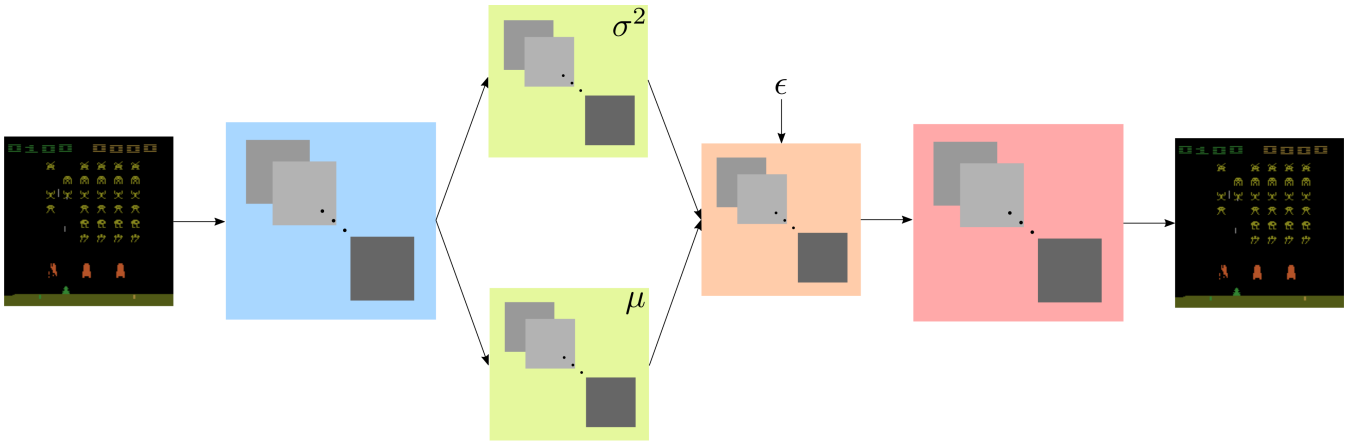


Figure 5.7: Caption.

- Even if we saw perfect separation of sprites in colour spaces, we'd only be able to exclusively separate at most three object types

Figure 5.8: Original

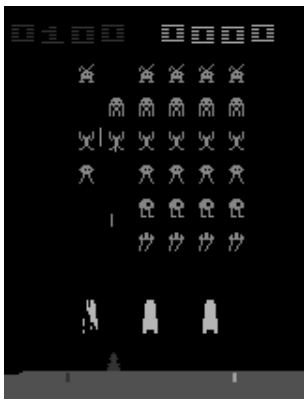


Figure 5.9: Red

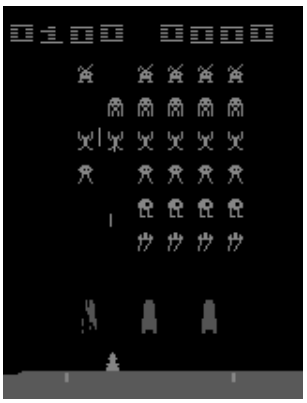


Figure 5.10: Green

Figure 5.11: Blue

Figure 5.12: A comparison of a 210×160 RGB frame from Space Invaders and its red, green and blue channels. The bullet is clearly separated from other sprites in the blue channel. The red and green channels separate collections of sprites. The red channel excludes the gunship and players score, while the green partially excludes the barriers.

5.4 Winner Takes All

- As mentioned, it seems likely that objects may be classified by which latent filter has a high activation in its position
- Assuming the latent space takes this structure, we would expect no more than one filter to be activated in any given position.
- In other words, objects must have distinct types: a sprite cannot be a gunship and a number at the same time.
- This leads to the Winner Takes All method, where we apply a position-wise pressure to the latent space so that no more than one filter has a high activation in any given position.
- Naturally this applies to fully-convolutional multiple latent filter architectures.

5.4.1 Derivation

We wish to include a term in our loss function that ensures no more than one filter has a high activation in any given position.

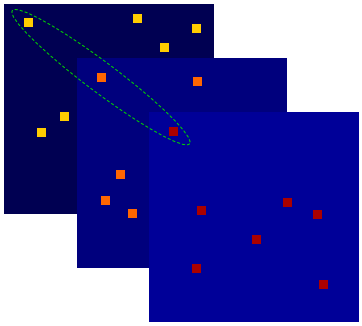


Figure 5.13: Multiple types of objects recognised in same position.

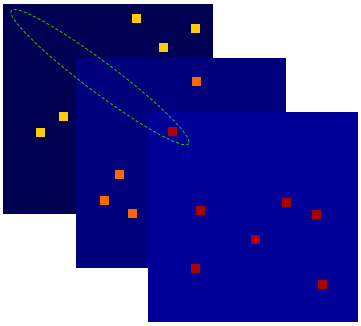


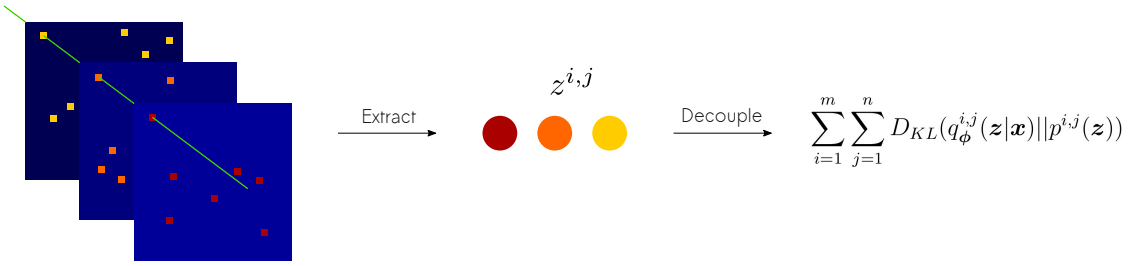
Figure 5.14: At most one object recognised in a given position.

Figure 5.15: Caption.

Consider a convolutional latent space $\boldsymbol{z}$ of shape (k, m, n). Let $\boldsymbol{z}^{i,j} \in \mathbb{R}^k$ be the vector storing the activations over the k filters at position (i, j). Naturally, we also let $q_{\boldsymbol{\phi}}^{i,j}(\boldsymbol{z}|\boldsymbol{x})$ represent the probabilistic encoder for $\boldsymbol{z}^{i,j} \in \mathbb{R}^k$. By matching the probabilistic encoder to an isotropic Gaussian we may reduce the redundancy, as argued previously. We choose to include the following term in our loss function:

$$\sum_{i=1}^{m}\sum_{j=1}^{n} D_{KL}(q_{\boldsymbol{\phi}}^{i,j}(\boldsymbol{z}|\boldsymbol{x})||p^{i,j}(\boldsymbol{z})) \quad \text{where} \quad p^{i,j}(\boldsymbol{z}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{5.1}$$

As shown with β-VAE, the multiplicative β controls the redundancy pressure. We will include this also and see if we observe a similar effect later.



Therefore the final loss function is given as:

$$\mathcal{L}_{WTA}(\boldsymbol{\theta}, \boldsymbol{\phi}; \boldsymbol{x}) = \mathbf{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})}\big[\log p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})\big] - \beta \sum_{i=1}^{m} \sum_{j=1}^{n} D_{KL}(q_{\boldsymbol{\phi}}^{i,j}(\boldsymbol{z}|\boldsymbol{x})||p^{i,j}(\boldsymbol{z})) \tag{5.2}$$

5.5 Orthogonal Convolutions

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

5.5.1 Architecture

TODO: Finish subsection

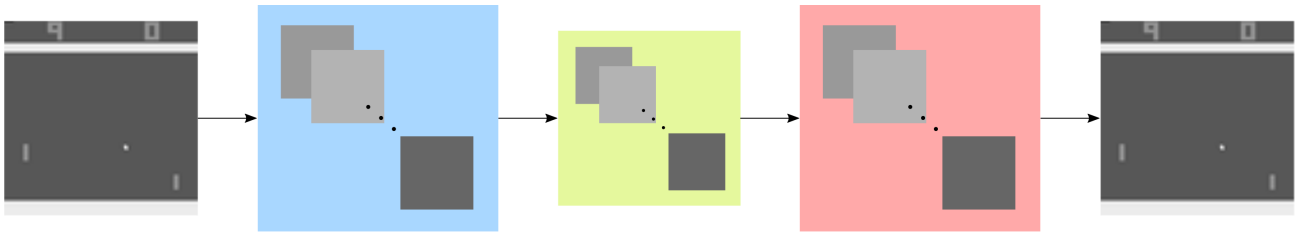


Figure 5.16: Caption.

5.5.2 Derivation

TODO: Finish subsection

Chapter 6

Results

6.1 Learning Generative Factors in Dense Latent Spaces

- Reconstruction
- Sampling
- MNIST, Frey, Pong, Space Invaders and Breakout
- We wish to be able to achieve a similar result with altering activations of specific filters

6.2 Single Latent Filter

- Consider two architectures: shallow and deep
- Consider Pong, Space Invaders (in progress) and Breakout
- Plot reconstruction loss and KL divergence
- Show reconstructions and convolutional layers of each
- Show mean activations of filters
- Alter latent space variable and show reconstruction

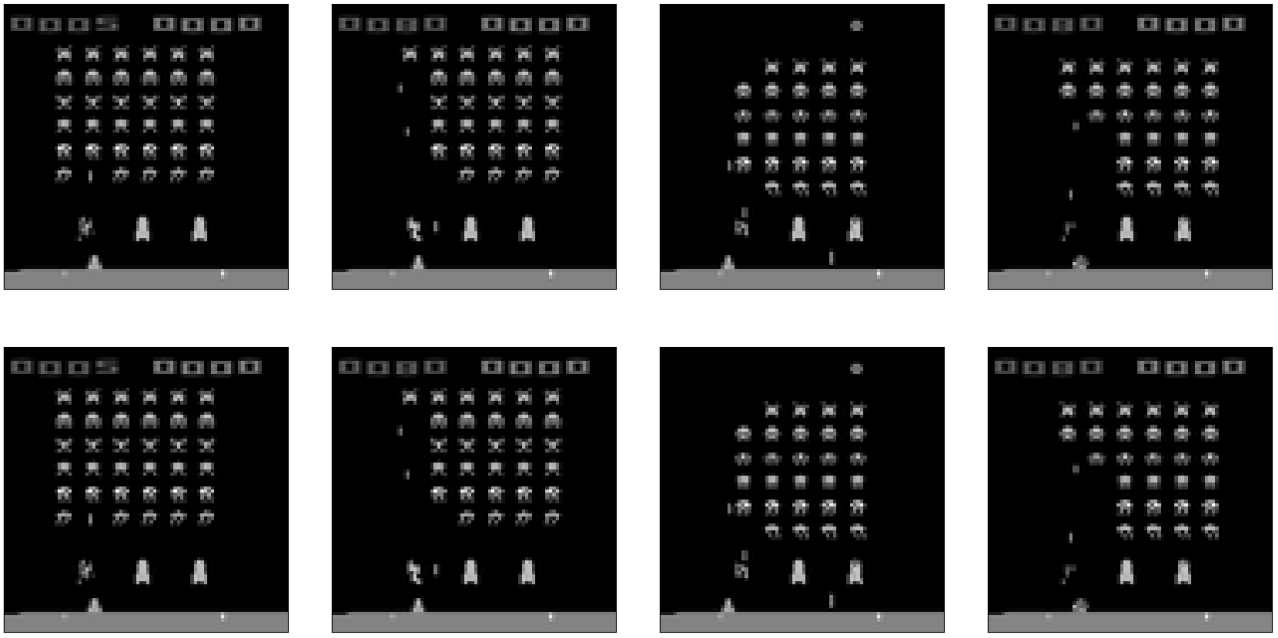


Figure 6.1: **Top row:** Original frames from Space Invaders. **Bottom row:** Corresponding reconstructions.

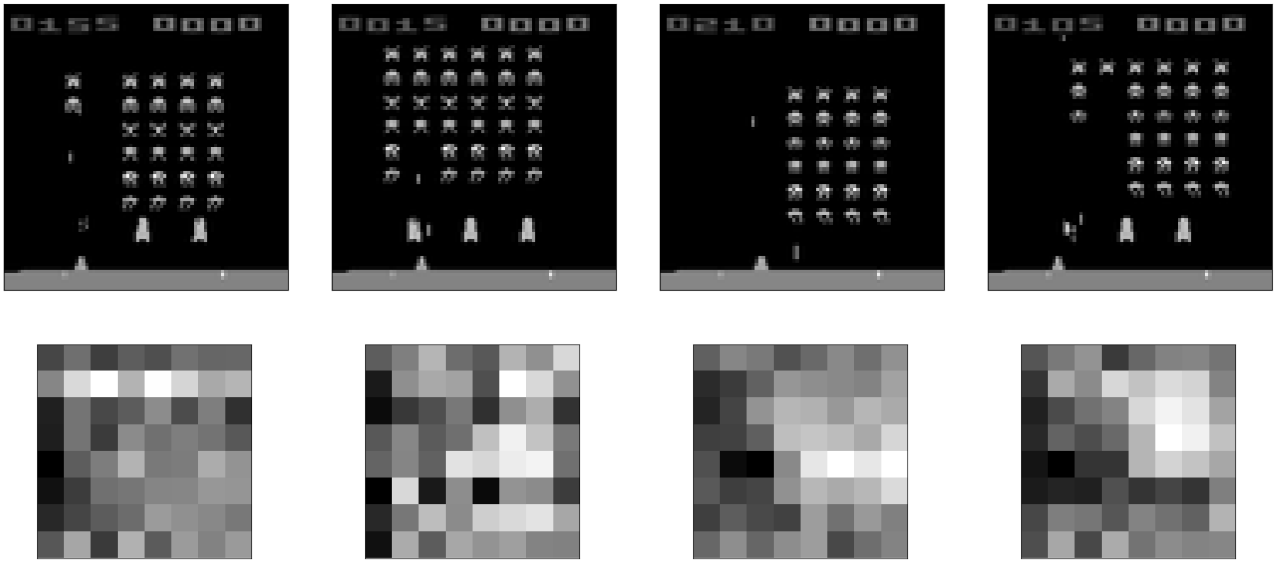


Figure 6.2: **Top row:** Original frames from Space Invaders. **Bottom row:** Corresponding latent images.

6.3 Disentangling Latent Neurons

- Consider two architectures: shallow and deep
- Consider Pong, Space Invaders (in progress) and Breakout

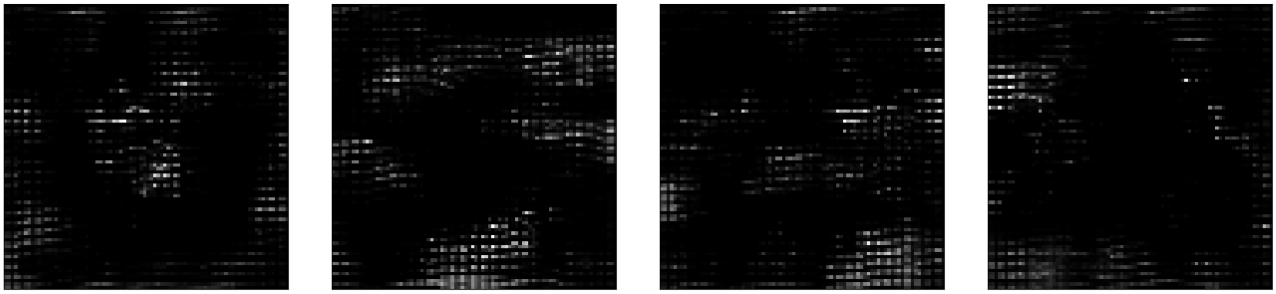


Figure 6.3: Samples from prior.

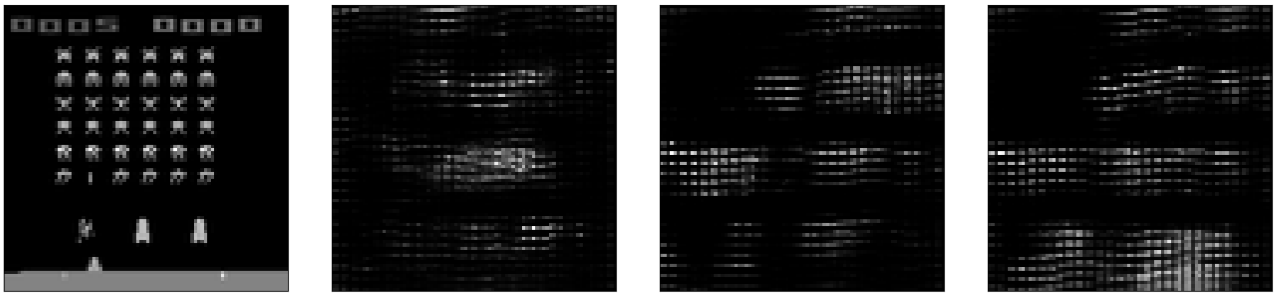


Figure 6.4: Samples from posterior.

- Plot reconstruction loss and KL divergence
- Show reconstructions and convolutional layers of each
- Show mean activations of filters
- Alter latent space variable and show reconstruction
- Add noise to filters

6.4 Disentangling Latent Filters Using Averages

- Consider two architectures: shallow and deep
- Consider Pong, Space Invaders (done) and Breakout
- Plot reconstruction loss and KL divergence
- Show reconstructions and convolutional layers of each

- Show mean activations of filters
- Alter latent space variable and show reconstruction
- Add noise to filters

6.5 Decoupling Latent Filters Using Weighted-Averages

- Consider two architectures: shallow and deep
- Consider Pong, Space Invaders (done) and Breakout
- Plot reconstruction loss and KL divergence
- Show reconstructions and convolutional layers of each
- Show mean activations of filters
- Alter latent space variable and show reconstruction
- Add noise to filters

6.6 Separating Colour Spaces

- Consider two architectures: shallow and deep
- Consider Pong, Space Invaders (in progress) and Breakout
- Plot reconstruction loss and KL divergence
- Show reconstructions and convolutional layers of each
- Show mean activations of filters
- Alter latent space variable and show reconstruction
- Add noise to filters

6.7 Orthogonal Convolutions

- Consider two architectures: shallow and deep
- Consider Pong, Space Invaders and Breakout
- Plot reconstruction loss and KL divergence
- Show reconstructions and convolutional layers of each
- Alter latent space variable and show reconstruction
- Add noise to filters

6.8 Winner Takes All

- Consider two architectures: shallow and deep
- Consider Pong, Space Invaders and Breakout
- Plot reconstruction loss and KL divergence
- Show reconstructions and convolutional layers of each
- Show mean activations of filters
- Alter latent space variable and show reconstruction
- Show samples from prior and posterior
- Add noise to filters

6.9 The No Free Lunch Relationship Between Reconstruction Loss and KL Divergence

- Vary β and plot BCE and KL

- Show example images for different β
- Consider Space Invaders, Pong and Breakout

6.10 Using Batch Normalisation With Convolutional Variational Latent Spaces

- Consider single architecture
- Batch norm before, after, both, and none

Chapter 7

Conclusion

7.1 Summary of Thesis Achievements

Summary.

7.2 Applications

Applications.

7.3 Future Work

Future Work.

[26] [19]

Bibliography

[1] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. In *IJCAI International Joint Conference on Artificial Intelligence*, volume 2015-Janua, pages 4148–4152, 2015.

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

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

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

[5] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets. *arXiv:1606.03657 [cs.LG]*, pages 1–14, 2016.

[6] F. Chollet. Keras: Deep Learning library for Theano and TensorFlow, 2015.

[7] T. S. Cohen and M. Welling. Transformation Properties of Learned Visual Representations. dec 2014.

[8] A. Creswell, K. Arulkumaran, and A. A. Bharath. Improving Sampling from Generative Autoencoders with Markov Chains. oct 2016.

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

[10] G. Desjardins, A. Courville, and Y. Bengio. Disentangling Factors of Variation via Generative Entangling. oct 2012.

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

[12] I. Dykeman. Conditional Variational Autoencoders, 2016.

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

[14] I. Higgins, L. Matthey, X. Glorot, A. Pal, B. Uria, C. Blundell, S. Mohamed, and A. Lerchner. Early Visual Concept Learning with Unsupervised Deep Learning. *arXiv*, 2016.

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

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

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

[18] 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.

[19] Q. V. Le, M. A. Ranzato, M. Devin, G. S. Corrado, and A. Y. Ng. Building High-level Features Using Large Scale Unsupervised Learning. *Arxiv*, 28(4):61–76, 2012.

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

[21] 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.

[22] 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.

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

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

[25] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. Playing for data: Ground truth from computer games. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9906 LNCS, pages 102–118, 2016.

[26] P. Rodríguez, J. Gonzàlez, G. Cucurull, J. M. Gonfaus, and X. Roca. Regularizing CNNs with Locally Constrained Decorrelations. nov 2016.

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

[28] J. Schmidhuber. Learning Factorial Codes by Predictability Minimization. *Neural Computation*, 4(6):863–879, 1992.

[29] 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.

[30] M. Stokes, M. Anderson, S. Chandrasekar, and R. Motta. A Standard Default Color Space for the Internet - sRGB, 1996.

[31] 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.

[32] Y. Tang and G. Hinton. Tensor Analyzers. *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 28, 2013.

[33] B. G. Thiagarajan, A. Member, and G. Z. Voyiadjis. beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. *ICLR'17 submission*, (July):1–13, 2016.

[34] 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.

[35] K. Zsolnai-Fehér. Computer Games Empower Deep Learning Research, 2016.