# DENSITY RATIO ESTIMATORS FOR VARIATIONAL METHODS IN BAYESIAN NEURAL NETWORKS

Alexander Lam

Supervisor: Professor Scott Sisson

School of Mathematics and Statistics
UNSW Sydney

October 2018

Submitted in partial fulfillment of the requirements of the degree of
Bachelor of Science with Honours

## Plagiarism statement

# Acknowledgements

Scott Sisson, Edwin Bonilla, Louis Tiao

Trevor Trotman

Honours Room Inhabitants, esp. Bouldering crew

Discord Group+Kong+Brian

Sunchit

누나

Family

Lammy, 11 September 2018.

# Abstract

Never use ReLU output to estimate density ratios.
Class probability estimator = good, direct density ratio and direct log density ratio estimators = bad.
Also KL Divergence > "CPE" Divergence for formulating estimator loss function but we already know that (although everyone seems to continue to use CPE divergence).

# Contents

# Chapter 1

# Introduction

## 1.1 Problem Context

In machine learning, particularly for high dimensional applications such as image analysis, it is often desirable to build generative models, so that we can represent the data in lower dimensions via representation learning, and generate new data similar to the examples in our dataset. Assume our dataset $X = \{x^{(i)}\}_{i=1}^{N} \sim q^*(x)$ is $N$ i.i.d. samples of random variables $x$. Also assume $x$ can be generated by a stochastic process from a latent continuous random variable $z$. These models involve a posterior distribution $p(z|x)$ that maps the dataset $x$ to lower dimensional latent prior $z$ (e.g. $z \sim N(\mu, \Sigma)$) then simulating from the prior $p(z)$ to generate new data through a decoder parametrized by $\theta$ $p_\theta(x|z)$. In this particular field, there are three main problems to solve:

1. Estimation of $\theta$, so that we can actually generate new data $x$
2. Evaluation of the posterior density $p(z|x) = \frac{p(z)p_\theta(x|z)}{p(x)} = \frac{p(x|z)p(z)}{\int_z p(x,z)dz}$, so we can encode our data $x$ in an efficient representation $z$
3. Marginal inference of $x$ ie. evaluating $p(x)$, so it can be used as a prior for other tasks

This problem is analogous to a typical Bayesian inference problem, in which $z$ is the parameter we want to perform inference on, and $x$ is the dataset. We have a distribution which represents our prior beliefs $p(z)$ and a likelihood distribution $p(x|z)$, and we want to determine the posterior distribution $p(z|x)$.

# Chapter 2

# Background on Neural Networks

In this chapter we give a general overview of a common model used in deep learning: neural networks. We first explain the motivation and intuition behind the model, then we describe the structure of an individual node. We then expand to the overall neural network structure, and after describing the network initialisation method, we conclude the chapter by demonstrating gradient descent training of neural networks and how back-propagation is used to find the required partial derivatives.

## 2.1 Motivation

Originally, neural networks were an attempt to create an algorithm that mimics the human brain's method of solving problems. The first machines using a neural network structure were created in the 1950s, and they were used widely from the 1980s onwards, as computers became sufficiently powerful for network training [Goodfellow et al., 2016].

One key feature of the brain structure is the capability of the neurons to adapt to suit different purposesZilles [1992]. Neuroscientists have conducted experiments on animals where they rewired the optic nerve from the eye to the auditory cortex. They found that the auditory cortex eventually adapted to process the visual signals, and the animals were able to perform tasks requiring sight. This experiment can be repeated for almost any input sensor and the neurons will adjust accordingly to process the signals in a useful manner. They deduced that each neuron has a similar structure regardless of its location in the brain, in which electrical signal inputs are transformed in some way and outputted to other neurons. Overall, the network of neurons was able to process an arbitrary input signal to suit a given purposeZilles [1992]. These are the core principles behind neural network models.

Let $f^*$ be some function in the space of $\mathbb{R}$. The primary goal of a neural network is to approximate $f^*$ using a mapping with parameters $\boldsymbol{\Theta}$ from input $\boldsymbol{x}$ to output $\boldsymbol{y}$: $\boldsymbol{y} = \boldsymbol{f}_{\boldsymbol{\Theta}}(\boldsymbol{x})$ [Goodfellow et al., 2016]. In fact, the universal approximation theorem states that neural networks can approximate any function in a finite-dimensional space with any desired non-zero amount of error, provided they are complex enough [Cybenko, 1989; Hornik, 1991]. For example, a typical regression problem of estimating housing prices would have the network inputting the values of certain predictors such as size (continuous) and type of building (categorical), and outputting the expected price. Another example is the classification problem of recognising handwritten digits (0-9) in a black and white image [Simard et al., 2003]. There would be many inputs corresponding to the value of each pixel, and the network would have 10 outputs corresponding to the probability of each digit, and the digit with the highest probability is then selected.

## 2.2 Individual Node Structure

Before discussing the overall structure of the neural network, we describe the structure of an individual node. A typical node takes in inputs from either the external input, or the outputs from other nodes, in addition to a bias node, which has the same purpose as the intercept term in a regression problem. The nodal inputs $\boldsymbol{x}$ are multiplied by weights $\boldsymbol{\theta}$ and then passed through an activation function $g(\boldsymbol{x})$. The individual node function is therefore

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\sum_{i=0}^{n} \theta_i x_i\right)$$

where n is the number of inputs excluding the bias node, which always has constant value $x_0 = 1$ [Cheng and Titterington, 1994]. An example of this is given in Figure 2.1 on the next page.

The objective of the activation function is to map the network output, possibly non-linearly, to a given range, such as $(0, 1)$ or $\mathbb{R}$. This mapping restricts the output range and determines how much input signal is needed before the output becomes asymptotically large [Haykin, 1998]. A list of common activation functions is given in Section 2.3.

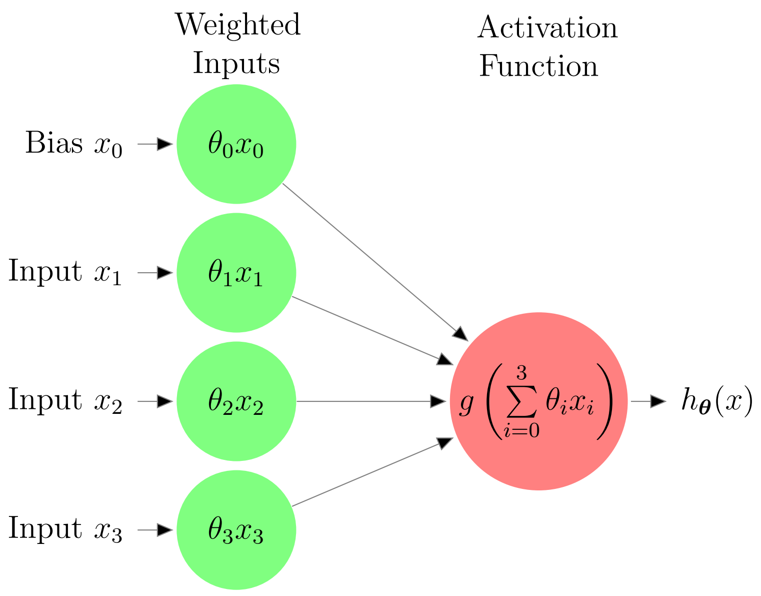


Figure 2.1: Example structure of an individual node function with 3 inputs, labelled as $\boldsymbol{x} = [x_0 \quad x_1 \quad x_2 \quad x_3]^\top$, with $x_0$ corresponding to the bias node. The weights are denoted as $\boldsymbol{\theta} = [\theta_0 \quad \theta_1 \quad \theta_2 \quad \theta_3]$.

## 2.3 Activation Functions

Some common activation functions are [Goodfellow et al., 2016]:

- The rectified linear unit or ReLU activation function output is bound in $[0, \infty)$. It has the formula $g(x) = \max\{0, x\}$ corresponding to node function $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \max\{0, \boldsymbol{\theta}^\top \boldsymbol{x}\}$.
- The sigmoid or logistic activation function outputs are restricted to $(0, 1)$, with the formula $g(x) = (1 + \exp(-x))^{-1}$ corresponding to node function $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = (1 + \exp(-\boldsymbol{\theta}^\top \boldsymbol{x}))^{-1}$.
- The hyperbolic tangent function output ranges between $(-1, 1)$, denoted as $g(x) = \tanh(x)$ corresponding to $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \tanh(\boldsymbol{\theta}^\top \boldsymbol{x})$.
- The linear activation function is used to describe nodes with no activation function, as it's formula is $g(x) = x$, corresponding to $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta}^\top \boldsymbol{x}$.

Their plots are shown in Figure 2.2 on the next page.

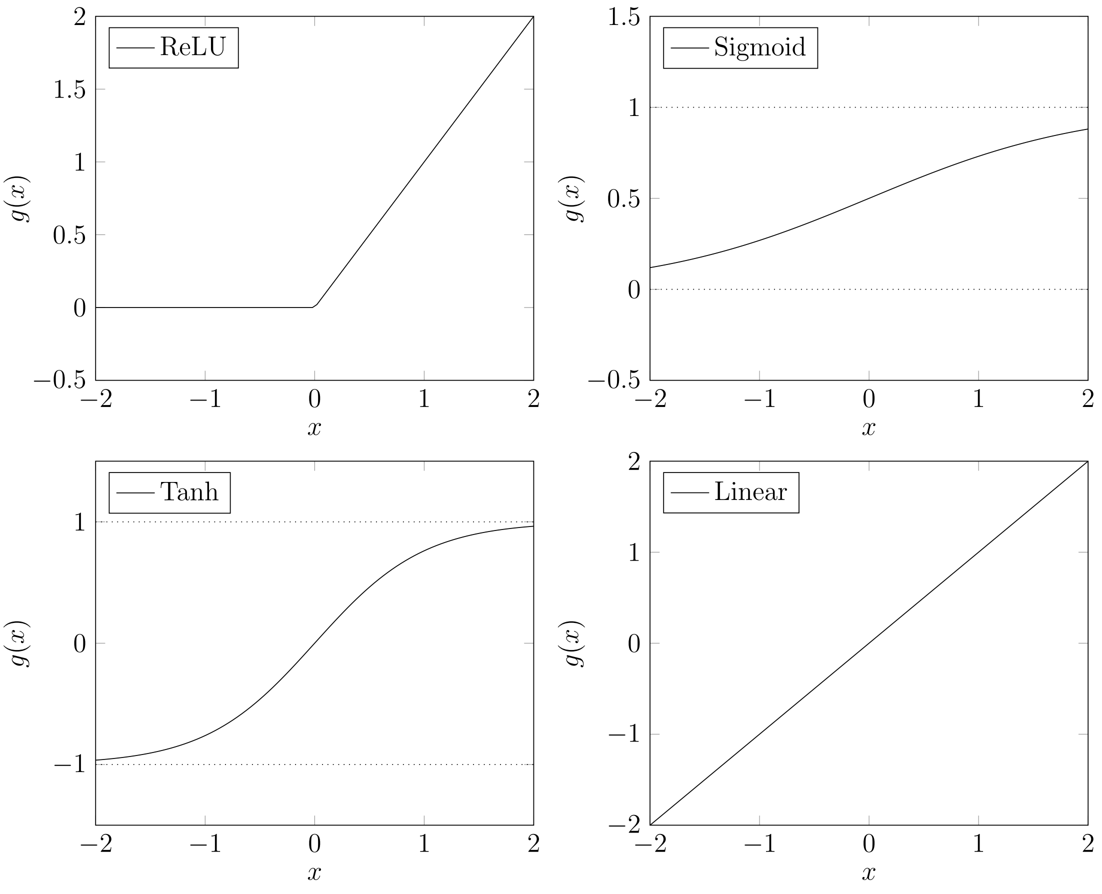


Figure 2.2: Activation Function Plots

The choice of activation function is dependent on several factors:

- The node's location in the network,
- Desired node output range,
- Continuous differentiability is ideal for gradient-based optimisation methods [Snyman, 2005],
- Monotonic activation functions ensure that the error is convex [Wu, 2009],
- In cases where the output range is restricted, the amount of input signal required for the activation function output to be asymptotically close to its limit.

For example, if the node's output is the estimate of a probability (ranging in $(0, 1)$), then the sigmoid function would be used [Cybenko, 1989]. In addition to its ideal output range, the sigmoid function is continuously differentiable ($g'(x) = g(x)(1-g(x))$) and requires significant input to output a value asymptotically close to 0 or 1. This allows the probability to be estimated with greater precision than with an activation function that approaches 0 or 1 very quickly. Note from Figure 2.2 that the tanh activation function approaches its limits much faster than the sigmoid function, so even if tanh was bound in $(0, 1)$, it would be a less suitable choice.

On the other hand, if the node's output was the expectation of a non-negative quantity, such as price or time, then the ReLU activation function would be used, as it is bound in $[0, \infty)$ and its linearity makes the overall node operation similar to linear regression.

Before discussing this further, we first describe the overall neural network structure, as the choice of activation function is dependent on the node's relative location on the network.

## 2.4 Neural Network Structure

A typical neural network is made up of layers of interconnected nodes [Cheng and Titterington, 1994]. The first layer, called the input layer, does not have an activation function or weights, rather it simply acts as an input interface for the network. The outputs from the nodes can only be sent to other nodes in succeeding layers, with the exception of the final output layer; it's result is simply the output of the network. The layers of nodes between the input and output layer are called the "hidden" layers, as their outputs are generally not interpreted by the user. Hidden layers can have an arbitrary number of nodes, whilst the nodes in the input and output layers are restricted to the number of inputs and outputs the program has. Example 2.4.1 on the next page explains the arithmetic operations within a neural network, and is illustrated in Figure 2.3.

The choice of activation function for a node in a neural network typically depends on the layer. Rectified linear units are the default choice for the hidden layers for their many advantages [Goodfellow et al., 2016]:

- Sparsity: since negative ReLU inputs result in a zero output, not all of the units are "active" (non-zero output) during the network's runtime. Sparsity is preferred in neural networks as it reduces overfitting and makes the model more robust to insignificant input changes [Glorot et al., 2011].
- Faster computation: a $\max\{0, x\}$ function is computed much faster than a function that uses exp or tanh.
- Better gradient propagation: weight training in a neural network (discussed in Sections 2.6 and 2.7) involves back-propagating a loss value through the network to calculate the partial derivatives of the loss function with respect to the weights. The weights receive a change proportional to their partial derivative. Back-propagation uses the chain rule, so activation functions such as sigmoid

or tanh that have a low gradient near their asymptotes may experience the 'vanishing gradient problem', in which the calculated partial derivatives become increasingly small as the loss value propagates through the network [Kolen and Kremer, 2001]. This causes the front layers to train very slowly. The ReLU activation function does not experience this issue as it its gradient is either linear or 0.

Since the input layer has no activation function, it can be described as having a linear activation function. The types of activation function used in the output layer has been explained in section 2.3.

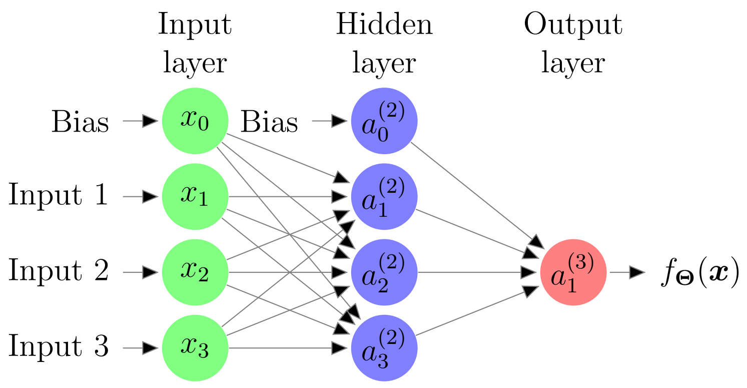


Figure 2.3: Example Neural Network Structure with 3 external inputs, 1 hidden layer with 3 nodes, 1 bias node per non-output layer and 1 output node. The variable inside each node denotes it's output as calculated in Section 2.2: the output of node $i$ in layer $j$ is denoted as $a_i^{(j)}$. $\boldsymbol{\Theta}$ denotes the weights of the network.

**Example 2.4.1.** In this example, we follow Figure 2.3, describing a neural network with 3 inputs, 1 hidden layer with 3 nodes and 1 output node. We denote the activation function as $g$, the output of node $i$ in layer $j$ as $a_i^{(j)}$, and the matrix of weights from layer $j$ to $j+1$ as $\Theta^{(j)}$. The activation function may vary with each node, but typically each layer uses the same activation function for each node. We also use the subscript $\Theta_{m,n}^{(j)}$ where $m$ is the row of the matrix corresponding to the node $m$ in layer $j+1$, and $n$ is the column of the matrix relating to node $n$ in layer $j$.

Denoting the weights outputting to unit $i$ in layer $j+1$ as $\boldsymbol{\theta}_i^{(j)}$, we have the following relation between the node weights as described in Section 2.2 and these weight matrices: $\boldsymbol{\theta}_i^{(j)} = [\Theta_{i,0}^{(j)} \quad \Theta_{i,1}^{(j)} \cdots \Theta_{i,k}^{(j)}]^\top$, where $k+1$ is the number of inputs.

Individually, the outputs in the hidden nodes and the output node are:

$$x_0 = 1, \qquad a_0^{(2)} = 1$$

$$\begin{aligned}
a_1^{(2)} &= g(\Theta_{1,0}^{(1)}x_0 + \Theta_{1,1}^{(1)}x_1 + \Theta_{1,2}^{(1)}x_2 + \Theta_{1,3}^{(1)}x_3) = g((\boldsymbol{\theta}_1^{(1)})^\intercal \boldsymbol{x}) \\
a_2^{(2)} &= g(\Theta_{2,0}^{(1)}x_0 + \Theta_{2,1}^{(1)}x_1 + \Theta_{2,2}^{(1)}x_2 + \Theta_{2,3}^{(1)}x_3) = g((\boldsymbol{\theta}_2^{(1)})^\top \boldsymbol{x}) \\
a_3^{(2)} &= g(\Theta_{3,0}^{(1)}x_0 + \Theta_{3,1}^{(1)}x_1 + \Theta_{3,2}^{(1)}x_2 + \Theta_{3,3}^{(1)}x_3) = g((\boldsymbol{\theta}_3^{(1)})^\top \boldsymbol{x}) \\
f_\Theta(\boldsymbol{x}) = a_1^{(3)} &= g(\Theta_{1,0}^{(2)}a_0^{(2)} + \Theta_{1,1}^{(2)}a_1^{(2)} + \Theta_{1,2}^{(2)}a_2^{(2)} + \Theta_{1,3}^{(2)}a_3^{(2)}) = g((\boldsymbol{\theta}_1^{(2)})^\top \boldsymbol{a}^{(2)}),
\end{aligned}$$

where $\boldsymbol{a}^{(2)} = [a_0^{(2)} \quad a_1^{(2)} \quad a_2^{(2)} \quad a_3^{(3)}]^\top$.

## 2.5 Weight Initialisation

Proper initialisation of the weights $\mathbf{\Theta}_i$ is ideal to improve network training (discussed in Sections 2.6 and 2.7), as if the weights are too low, then the nodal outputs will continually decrease through the layers and become very small, resulting in a significant loss value which requires many iterations of training to fix. A similar scenario occurs when the initial weights are too high [Bishop, 1995]. In this section we discuss Xavier Initialization [Glorot and Bengio, 2010], a common initialisation method used in deep learning which aims to keep the signal variance constant throughout the network. To derive the initialization algorithm, first consider a single node with $n+1$ inputs, and let $z$ denote the weighted sum of the inputs $\boldsymbol{\theta}^\top \boldsymbol{x}$ before it is passed through the activation function. This is written as

$$z = \theta_0 + \sum_{i=1}^{n} \theta_i x_i.$$

Here, $\theta_0$ is constant with respect to the external input, so $\mathrm{Var}(\theta_0) = 0$. Now without any prior knowledge of the inputs and weights, we assume that they are independent and have 0 mean. We can then find the variance of the other terms by using the formula for the product of independent variables[Goodman, 1960]:

$$\begin{aligned} Var(\theta_i x_i) &= \mathbb{E}[x_i]^2 \mathrm{Var}(\theta_i) + \mathbb{E}[\theta_i]^2 \mathrm{Var}(x_i) + \mathrm{Var}(\theta_i)\mathrm{Var}(x_i) \\ &= \mathrm{Var}(\theta_i)\mathrm{Var}(x_i). \end{aligned}$$

Assuming that the weights and inputs are also identically distributed, we have

$$\mathrm{Var}(z) = n\mathrm{Var}(\theta_i)\mathrm{Var}(x_i).$$

Since we want constant variance of the signals throughout the network, we set $\mathrm{Var}(z) = \mathrm{Var}(x_i)$ and the result follows:

$$\mathrm{Var}(\theta_i) = \frac{1}{n}.$$

However, this result only considers forward propagation of the signal. A variation of this result accounts for back propagation by averaging the number of input and output nodes:

$$\mathrm{Var}(\theta_i) = \frac{2}{n_{in} + n_{out}}.$$

Thus, to enforce constant signal variance throughout the network, the ideal initialization of weights is to sample from a distribution, typically uniform or Gaussian, with 0 mean and $\frac{2}{n_{in}+n_{out}}$ variance:

$$\theta_i \sim U\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right)$$

or

$$\theta_i \sim N\left(0, \frac{2}{n_{in}+n_{out}}\right).$$

## 2.6 Optimisation

The goal of optimising the network is to train the weights of the network such that a loss function, which we will denote as $L$, is minimized. A common loss function is the squared error between the batch of network outputs and the actual results, so our objective function would be:

$$\min_{\Theta} L(\Theta) = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{f}_{\Theta}(\boldsymbol{x}))^{\top}(\boldsymbol{y} - \boldsymbol{f}_{\Theta}(\boldsymbol{x})).$$

The $\frac{1}{2}$ factor is included to eliminate the factor of 2 in the derivative, simplifying the derivations. The derivative is multiplied by an arbitrary training rate during optimization so there is no significant impact of including that term [Goodfellow et al., 2016].

Back-propagation (Section 2.8) is used to calculate the partial derivative of the loss function with respect to each individual weight. These partial derivatives are used in the gradient-based optimisation of the weights. There are many variations of neural network optimisation algorithms, but they are mostly based off gradient descent, which we will cover in this section [Ruder, 2016].

**Definition 2.6.1.** At point $\boldsymbol{x}^{(n)}$, $\boldsymbol{s}^{(n)}$ is a descent direction if $\nabla f\left(\boldsymbol{x}^{(n)}\right)^{\top} \boldsymbol{s}^{(n)} < 0$.

Gradient descent [Boyd and Vandenberghe, 2004] is an algorithm used to find the minimizer $\boldsymbol{x}^{*}$ of a function $f$ by iterating on an arbitrary point $\boldsymbol{x}^{(n)}$, taking steps proportional to a descent direction $\boldsymbol{s}^{(n)}$:

$$\boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + \alpha \boldsymbol{s}^{(n)}, \qquad \alpha > 0.$$

**Proposition 2.6.2.** *If $\boldsymbol{s}^{(n)}$ is a descent direction for $x^{(n)}$, then for small $\alpha > 0$,*

$$f(\boldsymbol{x}^{(n)} + \alpha \boldsymbol{s}^{(n)}) < f(\boldsymbol{x}^{(n)}).$$

*Proof.* First we show that

$$\frac{d}{d\alpha} f(\boldsymbol{x}^{(n)} + \alpha \boldsymbol{s}^{(n)}) = \nabla f(\boldsymbol{x}^{(n)} + \alpha \boldsymbol{s}^{(n)})^\top \boldsymbol{s}^{(n)},$$

where $\boldsymbol{x}^{(n)} = [x_1^{(n)}, \ldots, x_k^{(n)}]^\top$ and $\boldsymbol{s}^{(k)} = [s_1^{(n)}, \ldots, s_k^{(n)}]^\top$.
Let

$$x_i^{(n)}(\alpha) = x_i^{(n)} + \alpha s_i^{(n)}, \quad i = 1, \ldots, n,$$

so that $\boldsymbol{x}^{(n)} + \alpha \boldsymbol{s}^{(n)} = [x_1^{(n)}(\alpha), \ldots, x_k^{(n)}(\alpha)]^\top$. We have

$$\begin{aligned}
\frac{d}{d\alpha} f(\boldsymbol{x}^{(n)} + \alpha \boldsymbol{s}^{(n)}) &= \frac{d}{d\alpha} f(x_1^{(n)}, \ldots, x_k^{(n)}(\alpha)) \\
&= \sum_{i=1}^{k} \frac{\partial f(\boldsymbol{x})}{\partial x_i}\Big|_{\boldsymbol{x}=\boldsymbol{x}^{(n)}+\alpha \boldsymbol{s}^{(n)}} \frac{d(x_i^{(n)}(\alpha))}{d\alpha} \\
&= \sum_{i=1}^{k} \frac{\partial f(\boldsymbol{x})}{\partial x_i}\Big|_{\boldsymbol{x}=\boldsymbol{x}^{(n)}+\alpha \boldsymbol{s}^{(n)}} \boldsymbol{s}_i^{(n)} \\
&= \nabla f(\boldsymbol{x}^{(n)} + \alpha \boldsymbol{s}^{(n)})^\top \boldsymbol{s}^{(n)}
\end{aligned}$$

Setting $\alpha = 0$ and using Definition 2.7.1,

$$\frac{d}{d\alpha} f(\boldsymbol{x}^{(n)} + \alpha \boldsymbol{s}^{(n)})|_{\alpha=0} = \nabla f(\boldsymbol{x}^{(n)})^\top \boldsymbol{s}^{(n)} < 0.$$

Therefore for small $\alpha > 0$,

$$f(x^{(n)} + \alpha s^{(n)}) < f(x^{(n)}).$$

□

A common choice of descent direction is the negative of the gradient, that is, $-\nabla f(\boldsymbol{x}^{(n)})$, leading to the method of steepest descent. It is clearly a descent direction as $-\nabla f(\boldsymbol{x}^{(n)})^\top \nabla f(\boldsymbol{x}^{(n)}) = -\|\nabla f(\boldsymbol{x}^{(n)})\|^2 < 0$, where $\|\cdot\|$ denotes the Euclidean norm.

By nature, gradient descent is guaranteed to converge to a local minimum, which is problematic if the function has local minima which differ from the global minima. This is not an issue in this thesis, as all the loss functions we use are convex, so any local minima are also global minima. Those interested in global optimization can refer to "Deterministic Global Optimization" by Floudas [2005]. When training a neural network on a non-convex loss function there are currently no commonly used methods of guaranteeing a global minimum, but the path may escape from a local minimum if randomness is introduced to the training process. This can be accomplished by stochastic gradient descent.

Typically, the entire batch of data is used in each iteration to calculate the loss function and gradient values required for gradient descent. This method of batch gradient descent is very slow for large datasets. Stochastic gradient descent is defined by the use of only one observation per iteration, so the latent randomness associated with each observation effectively leads to noise added to each step, but the optimization is much faster. In practice, a compromise between stochastic and batch gradient descent is typically used; mini-batch gradient descent involves using several observations per iteration, leading to a reduction in the gradient variance [Li et al., 2014; Ruder, 2016]. The size of the batch depends on the size and nature of the data set. For small data sets, the batch would typically be the entire data set as this would computationally feasible. On the other hand, online data is stochastic in nature and the data set is generally very large, so a small batch size would be used [Bengio, 2012].

Gradient descent convergence can be improved by using an adaptive learning rate, adjusting $\alpha$ over the iterations, as a low learning rate in the process will make convergence slow, whilst a high learning rate can cause the algorithm to oscillate around the minima [Ruder, 2016].

In this thesis, we use the Adam algorithm, which incorporates these two concepts to form an effective optimization algorithm that is commonly applied to neural networks. It uses the first and second moments of the gradient decay rate to adapt the weight training rate. More details can be found in Adam: A Method for Stochastic Optimization by Kingma, Ba [Kingma and Ba, 2014].

## 2.7 Back-Propagation

In the back-propagation algorithm, the goal is to find the partial derivative of the loss function with respect to the individual weights

$$\frac{\partial}{\partial \Theta_{m,n}^{(j)}} L(\Theta),$$

so that gradient descent can be performed to optimize the weights [E. Rumelhart et al., 1986]. For each training sample $(\boldsymbol{x}^{(I)}, \boldsymbol{y}^{(I)})$, $I = 1, \ldots, N$, the input signal is propagated forward throughout the network to calculate $\boldsymbol{a}^{(j)}$ for $j = 2, \ldots, J$, where $J$ is the total number of layers. The difference between the network output and the ideal result is calculated with

$$\boldsymbol{\delta}^{(J)} = \boldsymbol{a}^{(J)} - \boldsymbol{y}^{(I)},$$

and this error is propagated backwards through the network to find $\boldsymbol{\delta}^{(J-1)}, \dots, \boldsymbol{\delta}^{(2)}$ by using the formula

$$\boldsymbol{\delta}^{(j)} = ((\Theta^{(j)})^\top \boldsymbol{\delta}^{(j+1)}). * g'(\Theta^{(j)^\top} \boldsymbol{a}^{(j)}),$$

where $.*$ denotes element-wise multiplication and $g'$ is the derivative of the activation function. In this case, $g'$ takes in the sum of its weighted inputs, and as an example, the sigmoid activation function has the derivative $g'(\Theta^{(j)^\top}\boldsymbol{a}^{(j)}) = \boldsymbol{a}^{(j)}. * (1 - \boldsymbol{a}^{(j)})$. Note that $\boldsymbol{\delta}^{(1)}$ does not need to be calculated as the input layer is not weighted.
The errors for each layer are multiplied by each of the preceding layer's activation outputs to form the estimated partial derivative for the training sample. This result is added to an accumulator matrix, so that the average partial derivative from all the training samples can be computed:

$$\Delta^{(j)}_{m,n} := \Delta^{(j)}_{m,n} + a^{(j)}_n \delta^{(j+1)}_m$$

or in matrix-vector form.

$$\Delta^{(j)} := \Delta^{(j)} + \boldsymbol{\delta}^{(j+1)}(\boldsymbol{a}^{(j)})^\top.$$

Finally, we divide the accumulator matrix entries by the number of training samples to find the average partial derivative of the cost function with respect to the weights:

$$\frac{\partial}{\partial \Theta^{(j)}_{m,n}} L(\Theta) = \frac{1}{N}\Delta^{(j)}_{m,n}.$$

When $n \neq 0$ (i.e. not considering the bias node), we can optionally add a regularizer term $\lambda > 0$ which decreases the magnitude of the weights, reducing overfitting [Goodfellow et al., 2016]

$$\frac{\partial}{\partial \Theta^{(j)}_{m,n}} L(\Theta) = \frac{1}{N}(\Delta^{(j)}_{m,n} + \lambda \Theta^{(j)}_{m,n}).$$

There is no significant change when the bias node is regularized.

Pseudocode for back-propagation is shown in Algorithm 1 below.

```
Data: Training Data $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})\}$, Regularizer Term $\lambda$
Result: Cost Function Partial Derivatives $\frac{\partial}{\partial \Theta^{(j)}_{m,n}} L(\Theta)$
begin
    Initialize weights $\Theta$ using Xavier Initialisation;
    Set $\Delta^{(j)}_{m,n} = 0 \quad \forall j, m, n$;
    for $I = 1$ to $N$ do
        Set $\boldsymbol{a}^{(1)} = \boldsymbol{x}^{(I)}$;
        for $j = 2$ to $J$ do
            Set $\boldsymbol{a}^{(j)} = {\Theta^{(j-1)}}^{\top} \boldsymbol{a}^{(j-1)}$;
        end
        Set $\boldsymbol{\delta}^{(J)} = \boldsymbol{a}^{(J)} - \boldsymbol{y}^{(I)}$;
        for $j = J - 1$ to $2$ do
            Set $\boldsymbol{\delta}^{(j)} = ((\Theta^{(j)})^{\top} \boldsymbol{\delta}^{(j+1)}) .* g'({\Theta^{(j)}}^{\top} \boldsymbol{a}^{(j)})$;
        end
        for $j = 1$ to $J - 1$ do
            Set $\Delta^{(j)} = \Delta^{(j)} + \boldsymbol{\delta}^{(j+1)} (\boldsymbol{a}^{(j)})^{\top}$;
        end
    end
    for all $j, m, n$ do
        if $n = 0$ then
            Set $\frac{\partial}{\partial \Theta^{(j)}_{m,n}} L(\Theta) = \frac{1}{N} \Delta^{(j)}_{m,n}$;
        else
            Set $\frac{\partial}{\partial \Theta^{(j)}_{m,n}} L(\Theta) = \frac{1}{N} (\Delta^{(j)}_{m,n} + \lambda \Theta^{(j)}_{m,n})$;
        end
    end
end
```

**Algorithm 1:** Back-Propagation Algorithm

# Chapter 3

# Variational Inference

In this chapter, we explain variational inference, a method that uses a specific functional form to approximate posterior distributions in the context of Bayesian statistics. These are denoted as 'variational distributions', from the use of variational calculus to derive certain expressions. We first describe the Bayesian framework and problems associated with computational intractability. We then explain, with examples, two types of variational inference: mean-field variational inference and amortized inference. Finally, the chapter is concluded with a description of issues that arise when one or more of the prior or likelihood distributions are implicit, that is, samples can be readily drawn from them but the density function is difficult to numerically evaluate.

## 3.1 Context

A fundamental problem in Bayesian statistics is to evaluate, or estimate posterior densities to perform analysis on unknown parameters [Gelman et al., 2004]. Consider the set of latent and known variables $\boldsymbol{z} = (z_1, \ldots, z_M) \in \mathbb{R}^M$ and $\boldsymbol{x} = (x_1, \ldots, x_N) \in \mathbb{R}^N$, respectively, with joint density $p(\boldsymbol{z}, \boldsymbol{x})$. The posterior density $p(\boldsymbol{z}|\boldsymbol{x})$ is the distribution of the latent parameters $z_1, \ldots, z_M$ conditioned on the known variables $\boldsymbol{x}$. Applying Bayes' theorem, it can be written as:

$$p(\boldsymbol{z}|\boldsymbol{x}) = \frac{p(\boldsymbol{z}, \boldsymbol{x})}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})}{\int_{\mathbb{R}} p(\boldsymbol{z}, \boldsymbol{x}) d\boldsymbol{z}},$$

where

- $p(\boldsymbol{z})$ is the prior distribution: the initial distribution of $\boldsymbol{z}$ before the data $\boldsymbol{x}$ is observed. This can be initialised to represent our subjective beliefs, or it can be an uninformative prior that implies objectivity.
- $p(\boldsymbol{x}|\boldsymbol{z})$ is the likelihood: the distribution of data $\boldsymbol{x}$ conditioned on the parameters $\boldsymbol{z}$.

- $p(\boldsymbol{x}) = \int_{\boldsymbol{z}} p(\boldsymbol{z}, \boldsymbol{x}) d\boldsymbol{z}$ is the marginal likelihood, or the evidence: the density of the data averaged across all possible parameter values.

In simple cases, the posterior can typically be calculated algebraically by using the proportionality $p(\boldsymbol{z}|\boldsymbol{x}) \propto p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})$ and normalising over the constant $p(\boldsymbol{x})$. As the model becomes more complex, the calculations required can be extremely difficult, so traditional MCMC (Markov Chain Monte Carlo) methods overcome this obstacle by sampling from a Markov chain that converges to the stationary distribution $p(\boldsymbol{z}|\boldsymbol{x})$. However, these methods tend to have slow convergence for large datasets or high dimensional data. When faced with these issues or when desiring a faster computation, one may instead apply variational inference, an alternative approach to density estimation. Variational inference methods can be much faster than MCMC as they replace sampling with optimisation, but they are known to underestimate the true posterior variance [Blei et al., 2017].

## 3.2 The KL Divergence

We first define the f-divergence: a measure of how much two probability distributions differ.

**Definition 3.2.1.** The f-divergence of continuous probability distribution $Q$ from $P$ is

$$D_f(P\|Q) = \mathbb{E}_{p(u)}\left[f\left(\frac{q(u)}{p(u)}\right)\right],$$

where $f$ is a convex function such that $f(1) = 0$.

When $f(u) = -\log u$, we have the KL (Kullback-Leibler) divergence [Kullback, 1959], a type of f-divergence that is commonly used in variational inference [Blei et al., 2017].

**Definition 3.2.2.** The KL divergence is the expected logarithmic difference between two distributions $P$ and $Q$ with respect to $P$:

$$KL(p(x)\|q(x)) = \int_{-\infty}^{\infty} p(x)\log\left(\frac{p(x)}{q(x)}\right) dx = \mathbb{E}_{p(x)}\left[\log\left(\frac{p(x)}{q(x)}\right)\right].$$

**Remark 3.2.3.** *The KL divergence is not symmetric:*

$$KL(p(x)||q(x)) \neq KL(q(x)||p(x)) \text{ for } p(x) \neq q(x).$$

In variational inference, $KL(p(x)\|q(x))$ is known as the forward KL divergence, whilst $KL(q(x)\|p(x))$ is the reverse KL divergence.

**Lemma 3.2.4.** *The reverse KL divergence is formulated when $f(u) = u \log u$ in an f-divergence.*

*Proof.*

$$\begin{aligned}
D_{RKL}(P\|Q) &= \mathbb{E}_{p(u)}\left[\frac{q(u)}{p(u)}\log\left(\frac{q(u)}{p(u)}\right)\right] \\
&= \int p(u)\frac{q(u)}{p(u)}\log\left(\frac{q(u)}{p(u)}\right) du \\
&= \int q(u)\log\left(\frac{q(u)}{p(u)}\right) du \\
&= \mathbb{E}_q\left[\log\left(\frac{q(u)}{p(u)}\right)\right] \\
&= KL[q(u)\|p(u)].
\end{aligned}$$

□

Note that the forward and reverse KL divergences mainly differ in the distribution that is used to take the expectation.

**Lemma 3.2.5.** *The KL divergence is non-negative, and it is equal to zero if and only if $p(x)$ and $q(x)$ are equivalent:*

$$KL(q(x)||p(x)) \geq 0.$$

*We prove this in the case where $P$ and $Q$ are continuous distributions: a similar proof holds when they are discrete.*

*Proof.*

$$\begin{aligned}
KL(p(x)\|q(x)) &= \int_{-\infty}^{\infty} p(x)\log\left(\frac{p(x)}{q(x)}\right) dx \\
&= -\int_{-\infty}^{\infty} p(x)\log\left(\frac{q(x)}{p(x)}\right) dx \\
&\geq -\int_{-\infty}^{\infty} p(x)\left(\frac{q(x)}{p(x)} - 1\right) dx \\
&= -\int_{-\infty}^{\infty} q(x)dx + \int_{-\infty}^{\infty} p(x)dx \\
&= 0
\end{aligned}$$

In the third line we use $-\log x \geq -(x-1)$ for all $x > 0$ with equality if and only if $x = 1$, therefore $KL(q(x)\|p(x)) = 0$ if and only if $q(x) = p(x)$. The last line is due to $p(x)$ and $q(x)$ being probability densities. $\square$

## 3.3 Introduction to Variational Inference

Variational inference approximates the true posterior distribution $p(\boldsymbol{z}|\boldsymbol{x})$ with a different distribution $q(\boldsymbol{z})$, taken from a tractable family of approximate distributions $\mathcal{Q}$, and then minimizes the f-divergence between the two distributions in an optimization problem:

$$q^*(\boldsymbol{z}) = \underset{q(\boldsymbol{z})\in\mathcal{Q}}{\arg\min}\, D_f(q(\boldsymbol{z})\|p(\boldsymbol{z}|\boldsymbol{x})), \tag{3.3.1}$$

where $D_f$ denotes an f-divergence [Blei et al., 2017]. This produces an analytic approximation to the posterior density. The most common f-divergence used in variational inference is the reverse KL divergence, used instead of the forward KL divergence as we are unable to sample from our true posterior $p(z|x)$, and because it leads to an expectation maximization algorithm as opposed to an expectation propagation algorithm. Equation 3.3.1 can therefore be written as:

$$q^*(\boldsymbol{z}) = \underset{q(\boldsymbol{z})\in\mathcal{Q}}{\arg\min}\, KL(q(\boldsymbol{z})\|p(\boldsymbol{z}|\boldsymbol{x})). \tag{3.3.2}$$

From Lemma 3.2.5, it is evident that $KL(q(\boldsymbol{z})\|p(\boldsymbol{z}|\boldsymbol{x}))$ attains a minimal value of 0 when $q(\boldsymbol{z}) = p(\boldsymbol{z}|\boldsymbol{x})$.
There is an issue with solving equation (3.3.2) directly: we cannot evaluate the reverse KL divergence as $p(\boldsymbol{z}|\boldsymbol{x})$ is unknown. Instead, we rearrange the terms of equation 3.3.2 to formulate a tractable expression that can be optimized.

## 3.4 Derivation of the ELBO

In this section, we formulate the evidence lower bound (ELBO) of our posterior inference problem. Maximisation of this term is equivalent to solving equation (3.3.2).

We begin by applying Bayes' law to the problem and expanding the terms:

$$\begin{aligned}
q^*(\boldsymbol{z}) &= \underset{q(\boldsymbol{z})\in\mathcal{Q}}{\arg\min}\, KL(q(\boldsymbol{z})\|p(\boldsymbol{z}|\boldsymbol{x})) \\
&= \underset{q(\boldsymbol{z})\in\mathcal{Q}}{\arg\min}\, \mathbb{E}_{q(\boldsymbol{z})}[\log q(\boldsymbol{z}) - \log p(\boldsymbol{z}|\boldsymbol{x})] \\
&= \underset{q(\boldsymbol{z})\in\mathcal{Q}}{\arg\min}\, \mathbb{E}_{q(\boldsymbol{z})}\left[\log q(\boldsymbol{z}) - \log \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})}\right] \\
&= \underset{q(\boldsymbol{z})\in\mathcal{Q}}{\arg\min}\, \left(\mathbb{E}_{q(\boldsymbol{z})}[\log q(\boldsymbol{z}) - \log p(\boldsymbol{x}|\boldsymbol{z}) - \log p(\boldsymbol{z})] + \log p(\boldsymbol{x})\right).
\end{aligned}$$

Note in the last line $\mathbb{E}_{q(\boldsymbol{z})}[p(\boldsymbol{x})] = p(\boldsymbol{x})$ as it is not dependent on $q(\boldsymbol{z})$. Since our issues with equation (3.3.2) result from the intractability of $p(\boldsymbol{x})$, we rearrange the KL divergence expression as follows:

$$\begin{aligned}
KL(q(\boldsymbol{z})\|p(\boldsymbol{z}|\boldsymbol{x})) &= \mathbb{E}_{q(\boldsymbol{z})}[\log q(\boldsymbol{z}) - \log p(\boldsymbol{x}|\boldsymbol{z}) - \log p(\boldsymbol{z})] + \log p(\boldsymbol{x}) \\
\log p(\boldsymbol{x}) - KL(q(\boldsymbol{z})\|p(\boldsymbol{z}|\boldsymbol{x})) &= -\mathbb{E}_{q(\boldsymbol{z})}[\log q(\boldsymbol{z}) - \log p(\boldsymbol{x}|\boldsymbol{z}) - \log p(\boldsymbol{z})] \\
&= \mathbb{E}_{q(\boldsymbol{z})}[\log p(\boldsymbol{x}|\boldsymbol{z})] - \mathbb{E}_{q(\boldsymbol{z})}[\log q(\boldsymbol{z}) - \log p(\boldsymbol{z})] \\
&= \mathbb{E}_{q(\boldsymbol{z})}[\log p(\boldsymbol{x}|\boldsymbol{z})] - KL(q(\boldsymbol{z})\|p(\boldsymbol{z})) \qquad (3.4.1) \\
\log p(\boldsymbol{x}) &\geq \mathbb{E}_{q(\boldsymbol{z})}[\log p(\boldsymbol{x}|\boldsymbol{z})] - KL(q(\boldsymbol{z})\|p(\boldsymbol{z})). \qquad (3.4.2)
\end{aligned}$$

We refer to $\log p(\boldsymbol{x}) - KL(q(\boldsymbol{z})\|p(\boldsymbol{z}|\boldsymbol{x}))$ as $ELBO(q)$, as it is equal to the marginal probability of the data subtracted by a constant 'error term'. Now our problem of minimizing the KL divergence between $q(\boldsymbol{z})$ and $p(\boldsymbol{z}|\boldsymbol{x})$ is equivalent to maximizing $ELBO(q)$, which is equal to the tractable expression on line (3.4.1). We can therefore rewrite our optimization problem as:

$$\begin{aligned}
q^*(\boldsymbol{z}) &= \underset{q(\boldsymbol{z})\in\mathcal{Q}}{\arg\min}\, KL(q(\boldsymbol{z})\|p(\boldsymbol{z}|\mathbf{x})) \\
&= \underset{q(\boldsymbol{z})\in\mathcal{Q}}{\arg\max}\, ELBO(q) \\
&= \underset{q(\boldsymbol{z})\in\mathcal{Q}}{\arg\max}\, \left(\mathbb{E}_{q(\boldsymbol{z})}[\log p(\boldsymbol{x}|\boldsymbol{z})] - KL(q(\boldsymbol{z})\|p(\boldsymbol{z}))\right). \qquad (3.4.3)
\end{aligned}$$

Note from line (3.4.2) that the expression on line (3.4.3) attains a maximum at the marginal likelihood of the dataset $\log p(\boldsymbol{x})$, hence we may use the ELBO to construct a model selection criterion[Bishop, 2006]. However, this criterion may not be reliable as it is a lower bound.

## 3.5 Mean-Field Variational Family

The family of variational distributions $\mathcal{Q}$ is typically a 'mean-field variational family', in which the distribution $q(\boldsymbol{z})$ factorizes over the latent variables $\{z_i\}_{i=1}^M$, each with an individual set of parameters $\{\phi_i\}_{i=1}^M$ [Blei et al., 2017]:

$$q(\boldsymbol{z}) = \prod_{i=1}^{M} q_{\phi_i}(z_i). \tag{3.5.1}$$

The individual factors $q_{\phi_i}(z_i)$ can take any form, but they are assumed to be independent, which simplifies derivations but is less accurate when the true latent variables exhibit dependence. Fixing the forms of the individual factors, we want to choose the parameters $\phi_i$ so that $ELBO(q)$ is maximized. To derive an expression for the optimal factor $q_i^*(z_i)$, we substitute equation (3.5.1) into the $ELBO$, factor out a specific $q_j(z_j)$ and equate the functional derivative of the resulting Lagrangian equation with 0.
Firstly, we express $ELBO(q)$ in an integral form as follows:

$$\begin{aligned}
ELBO(q) &= \mathbb{E}_{q(z)}[\log p(\boldsymbol{x}|\boldsymbol{z})] - KL(q(\boldsymbol{z})\|p(\boldsymbol{z})) \\
&= \mathbb{E}_{q(x)}[\log p(\boldsymbol{x}|\boldsymbol{z}) + \log p(\boldsymbol{z}) - \log q(\boldsymbol{z})] \\
&= \mathbb{E}_{q(z)}[\log p(\boldsymbol{x}, \boldsymbol{z}) - \log q(\boldsymbol{z})] \\
&= \int_{\mathbb{R}^M} q(\boldsymbol{z})(\log p(\boldsymbol{x}, \boldsymbol{z}) - \log q(\boldsymbol{z}))d\boldsymbol{z}.
\end{aligned}$$

Substituting $q(\boldsymbol{z}) = \prod_{i=1}^M q_i(z_i)$ and factoring out $q_j(z_j)$ yields:

$$\begin{aligned}
ELBO(q) &= \int_{\mathbb{R}^M} \left[\prod_{i=1}^{M} q_i(z_i)\right] \left(\log p(\boldsymbol{x}, \boldsymbol{z}) - \sum_{i=1}^{M} \log q_i(z_i)\right) d\boldsymbol{z} \\
&= \int_{\mathbb{R}} q_j(z_j) \left(\int_{\mathbb{R}^{M-1}} \log p(\boldsymbol{x}, \boldsymbol{z}) \prod_{i \neq j} q_i(z_i) d\boldsymbol{z}_{-j}\right) dz_j \\
&\quad - \int_{\mathbb{R}} q_j(z_j) \left(\int_{\mathbb{R}^{M-1}} \left[\prod_{i \neq j} q_i(z_i)\right] \sum_{i=1}^{M} \log q_i(z_i) d\boldsymbol{z}_{-j}\right) dz_j \\
&= \int_{\mathbb{R}} q_j(z_j) \mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(\boldsymbol{x}, \boldsymbol{z})] dz_j \\
&\quad - \int_{\mathbb{R}} q_j(z_j) \log q_j(z_j) \left(\int_{\mathbb{R}^{M-1}} \prod_{i \neq j} q_i(z_i) dz_{-j}\right) dz_j
\end{aligned}$$

$$\begin{aligned}
&- \int_{\mathbb{R}} q_j(z_j) \left( \int_{\mathbb{R}^{M-1}} \left[ \prod_{i \neq j} q_i(z_i) \right] \sum_{i \neq j} \log q_i(z_i) d\boldsymbol{z}_{-j} \right) dz_j \\
&= \int_{\mathbb{R}} q_j(z_j) \mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(\boldsymbol{x}, \boldsymbol{z})] dz_j - \int_{\mathbb{R}} q_j(z_j) \log q_j(z_j) dz_j \\
&- \int_{\mathbb{R}^{M-1}} \left[ \prod_{i \neq j} \log q_i(z_i) \right] \sum_{i \neq j} \log q_i(z_i) d\boldsymbol{z}_{-j} \qquad (3.5.2) \\
&= \int_{\mathbb{R}} q_j(z_j) \left( \mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(\boldsymbol{x}, \boldsymbol{z})] - \log q_j(z_j) \right) dz_j + \text{const.} \qquad (3.5.3)
\end{aligned}$$

The term in line (3.5.2) is a constant with respect to $q_j(z_j)$. We want to maximize $ELBO(q)$, so we formulate the Lagrangian equation with the constraint that $q_i(z_i)$ are probability density functions:

$$ELBO(q) - \sum_{i=1}^{M} \lambda_i \int_{\mathbb{R}} q_i(z_i) dz_i = 0,$$

or using our expression for $ELBO(q)$ in line (3.5.3),

$$\int_{\mathbb{R}} q_j(z_j) \left( \mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(\boldsymbol{x}, \boldsymbol{z})] - \log q_j(z_j) \right) dz_j - \sum_{i=1}^{M} \lambda_i \int_{\mathbb{R}} q_i(z_i) dz_i + \text{const} = 0. \qquad (3.5.4)$$

We then take the functional derivative of equation (3.5.4) with respect to $q_j(z_j)$:

$$\begin{aligned}
\frac{\partial ELBO(q)}{\partial q_j(z_j)} &= \frac{\partial}{\partial q_j(z_j)} \left[ q_j(z_j) \left( \mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(\boldsymbol{x}, \boldsymbol{z})] - \log q_j(z_j) \right) - \lambda_j q_j(z_j) \right] \\
&= \mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(\boldsymbol{x}, \boldsymbol{z})] - \log q_j(z_j) - 1 - \lambda_j. \qquad (3.5.5)
\end{aligned}$$

Equating expression (3.5.5) to 0 and observing that $1 + \lambda_j$ is constant with respect to $z$, we have:

$$\begin{aligned}
\log q_j^*(z_j) &= \mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(\boldsymbol{x}, \boldsymbol{z})] - \text{const} \\
q_j^*(z_j) &= \frac{e^{\mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(\boldsymbol{x}, \boldsymbol{z})]}}{\exp(\text{const})} \\
&= \frac{e^{\mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(\boldsymbol{x}, \boldsymbol{z})]}}{\int e^{\mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(\boldsymbol{x}, \boldsymbol{z})]} dz_j}. \qquad (3.5.6)
\end{aligned}$$

The normalization constant on the denominator of (3.5.6) is derived by observing $q_j^*(z_j)$ as a density. Finally, we derive a simpler expression of $q_j^*(z_j)$ by observing that terms independent of $z_j$ can be treated as a constant:

$$\begin{aligned} q_j^*(z_j) &\propto \exp\left(\mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(\boldsymbol{x}, \boldsymbol{z})]\right) \\ &\propto \exp\left(\mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(z_j | \boldsymbol{z}_{-j}, \boldsymbol{x})]\right). \end{aligned} \tag{3.5.7}$$

This expression can be used in an expectation-maximization algorithm, in which the $q_j^*(z_j)$ is evaluated and iterated from $j = 1 \dots M$ until $ELBO(q)$ converges. We can say this occurs when there is little variation in $ELBO(q)$ over the iterations. This particular algorithm is called coordinate ascent variational inference (CAVI) (Algorithm 2):

```
Data: Dataset $\boldsymbol{x}$ and Bayesian Model p($\boldsymbol{x}, \boldsymbol{z}$)
Result: Variational density $q(\boldsymbol{z}) = \prod_{i=1}^{M} q_i(z_i)$
begin
    Initialize random variational factors $q_j(z_j)$;
    while ELBO(q) has not converged do
        for $j = 1$ to $m$ do
            Set $q_j(z_j) \propto \exp(\mathbb{E}_{\boldsymbol{z}_{-j}}[\log p(z_j|\boldsymbol{z}_{-j}, \boldsymbol{x})])$;
        end
        Calculate $ELBO(q) = \mathbb{E}_{q(\boldsymbol{z})}[\log p(\boldsymbol{z}, \boldsymbol{x})] - \mathbb{E}_{q(\boldsymbol{z})}[\log q(\boldsymbol{z})]$;
    end
    Return $q(\boldsymbol{z})$;
end
```

**Algorithm 2:** Coordinate Ascent Variational Inference (CAVI)

## 3.6 Example: Bayesian Mixture of Gaussians

To illustrate the mean-field variational inference approach, we closely follow the "Bayesian mixture of Gaussians" example from "Variational Inference: A Review for Statisticians" by Blei et al. [2017].

Consider the hierarchical model

$$\begin{aligned} \mu_k &\sim N(0, \sigma^2), && k = 1, \dots, K, \\ c_i &\sim \text{Categorical}\left(1; \frac{1}{K}, \dots, \frac{1}{K}\right), && i = 1, \dots, n, \\ x_i | c_i, \boldsymbol{\mu} &\sim N(c_i^\top \boldsymbol{\mu}, 1), && i = 1, \dots, n, \end{aligned}$$

where $\boldsymbol{\mu} = (\mu_1, \dots, \mu_K)^\top$.

This is a Bayesian mixture of univariate Gaussian random variables with unit variance. In this model, we draw $K$ $\mu_k$ variables from a prior Gaussian distribution $N(0, \sigma^2)$ ($\sigma^2$ is a fixed hyperparameter), forming the vector $\boldsymbol{\mu}$. We then generate an indicator vector $c_i$ of length $K$ from a prior categorical distribution. This vector has zeros for every element except for one element, where it is a 1. Each element has equal probability $1/K$ of being the element that is equal to 1. The transpose of this $c_i$ is then multiplied by $\boldsymbol{\mu}$, essentially choosing one of the $\boldsymbol{\mu}$ elements at random. We then draw $x_i$ from the resulting $N(c_i^\top \boldsymbol{\mu}, 1)$.

Defining $\boldsymbol{c} = (c_1, \dots, c_n)^\top$, our latent variables are $\boldsymbol{z} = \{\boldsymbol{c}, \boldsymbol{\mu}\}$. Assuming $n$ samples, our joint density is

$$p(\boldsymbol{\mu}, \boldsymbol{c}, \boldsymbol{x}) = p(\boldsymbol{\mu}) \prod_{i=1}^{n} p(c_i) p(x_i | c_i, \boldsymbol{\mu}). \tag{3.6.1}$$

To evaluate the posterior distribution over the latent variables $p(\boldsymbol{\mu}, \boldsymbol{c} | \boldsymbol{x})$, we apply variational inference, approximating it with a variational distribution $q(\boldsymbol{\mu}, \boldsymbol{c})$. We will assume this distribution follows the mean-field variational family:

$$q(\boldsymbol{\mu}, \boldsymbol{c}) = \prod_{k=1}^{K} q(\mu_k; m_k, s_k^2) \prod_{i=1}^{n} q(c_i; \boldsymbol{\phi_i}).$$

In this distribution, we have $K$ Gaussian factors with mean $m_k$ and variance $s_k^2$, and $n$ categorical factors with index probabilities defined by the vector $\boldsymbol{\phi_i}$, such that

$$\begin{aligned} \mu_k &\sim N(m_k, s_k^2), && k = 1, \dots, K, \\ c_i &\sim \text{Categorical}(\boldsymbol{\phi_i}), && i = 1, \dots, n. \end{aligned}$$

Using this and equation (3.6.1), we can derive the evidence lower bound as a function of the variational parameters $\boldsymbol{m} = (m_1, \dots, m_k)^\top$, $\boldsymbol{s}^2 = (s_1^2, \dots, s_k^2)^\top$ and $\boldsymbol{\phi} = [\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_n]^\top$:

$$\begin{aligned} ELBO(\boldsymbol{m}, \boldsymbol{s}^2, \boldsymbol{\phi}) &= \mathbb{E}_{p(\mathbf{z},\boldsymbol{x})}[\log p(\mathbf{z}, \boldsymbol{x})] - \mathbb{E}_{q(\boldsymbol{z})}[\log q(\boldsymbol{z})] \\ &= \mathbb{E}_{p(\boldsymbol{\mu},\boldsymbol{c},\mathbf{x})}[\log p(\boldsymbol{\mu}, \boldsymbol{c}, \mathbf{x})] - \mathbb{E}_{q(\boldsymbol{\mu},\boldsymbol{c})}[\log q(\boldsymbol{\mu}, \boldsymbol{c})] \\ &= \sum_{i=1}^{K} \mathbb{E}_{p(\mu_k)}[\log p(\mu_k); m_k, s_k^2] \\ &\quad + \sum_{i=1}^{n} \left( \mathbb{E}_{p(c_i)}[\log p(c_i); \boldsymbol{\phi}_i] + \mathbb{E}_{p(x_i|c_i,\boldsymbol{\mu})}[\log p(x_i|c_i, \boldsymbol{\mu}); \boldsymbol{\phi}_i, \boldsymbol{m}, \boldsymbol{s}^2] \right) \\ &\quad - \sum_{k=1}^{K} \mathbb{E}_{q(\mu_k;m_k,s_k^2)}[\log q(\mu_k; m_k, s_k^2)] - \sum_{i=1}^{n} \mathbb{E}_{q(c_i;\boldsymbol{\phi}_i)}[\log q(c_i; \boldsymbol{\phi}_i)]. \end{aligned}$$

From equation (3.5.7), we derive the optimal categorical factor by only considering terms from the true distribution $p(\cdot)$ dependent on $c_i$:

$$q^*(c_i; \boldsymbol{\phi}_i) \propto \exp\left(\log p(c_i) + \mathbb{E}_{p(x_i|c_i,\boldsymbol{\mu})}[\log p(x_i|c_i, \boldsymbol{\mu}); \boldsymbol{m}, \boldsymbol{s}^2]\right). \tag{3.6.2}$$

Now since $c_i = (c_{i1}, \dots, c_{iK})^\top$ is an indicator vector,

$$p(x_i|c_i, \boldsymbol{\mu}) = \prod_{k=1}^{K} p(x_i|\mu_k)^{c_{ik}}.$$

We can now evaluate the second term of equation (3.6.2):

$$\begin{aligned} \mathbb{E}_{p(x_i|c_i,\boldsymbol{\mu})}\left([\log p(x_i|c_i, \boldsymbol{\mu}); \boldsymbol{m}, \boldsymbol{s}^2]\right) &= \sum_{k=1}^{K} c_{ik} \mathbb{E}_{p(x_i|\mu_k)}[\log p(x_i|\mu_k); m_k, s_k^2] \\ &= \sum_{k=1}^{K} c_{ik} \mathbb{E}_{x_i}[-(x_i - \mu_k)^2/2; m_k, s_k^2] + \text{const} \end{aligned}$$

$$= \sum_{k=1}^{K} c_{ik} \left( \mathbb{E}_{\mu_k}[\mu_k; m_k, s_k^2] x_i - \mathbb{E}_{\mu_k^2}[\mu_k^2; m_k, s_k^2]/2 \right) + \text{const.}$$

In each line, terms constant with respect to $c_{ik}$ have been absorbed into the constant. Our optimal categorical factor becomes

$$q^*(c_i; \boldsymbol{\phi}_i) \propto \exp\left( \log p(c_i) + \sum_{k=1}^{K} c_{ik} \left( \mathbb{E}_{\mu_k}[\mu_k; m_k, s_k^2] x_i - \mathbb{E}_{\mu_k^2}[\mu_k^2; m_k, s_k^2]/2 \right) \right).$$

By proportionality, we then have the variational update

$$\phi_{ik} \propto \exp\left( \mathbb{E}_{\mu_k}[\mu_k; m_k, s_k^2] x_i - \mathbb{E}_{\mu_k^2}[\mu_k^2; m_k, s_k^2]/2 \right).$$

Now we find the variational density of the $k$th mixture component, again using equation 3.5.7 with the ELBO and ignoring terms independent of $p(\cdot)$ and $\mu_k$:

$$q(\mu_k; m_k, s_k^2) \propto \exp\left( \log p(\mu_k) + \sum_{i=1}^{n} \mathbb{E}_{p(x_i|c_i,\boldsymbol{\mu})}[\log p(x_i|c_i, \boldsymbol{\mu}); \boldsymbol{\phi}_i, \boldsymbol{m}_{-k}, \boldsymbol{s}_{-k}^2] \right).$$

The log of this density is

$$\begin{aligned}
\log q(\mu_k) &= \log p(\mu_k) + \sum_{i=1}^{n} \mathbb{E}_{p(x_i|c_i,\boldsymbol{\mu})}[\log p(x_i|c_i, \boldsymbol{\mu}); \boldsymbol{\phi}_i, \boldsymbol{m}_{-k}, \boldsymbol{s}_{-k}^2] + \text{const} \\
&= \log p(\mu_k) + \sum_{i=1}^{n} \mathbb{E}_{c_{ik}, p(x_i|,\mu_k)}[c_{ik} \log p(x_i|\mu_k); \boldsymbol{\phi}_i] + \text{const} \\
&= -\frac{\mu_k^2}{2\sigma^2} + \sum_{i=1}^{n} \mathbb{E}_{c_{ik}}[c_{ik}; \boldsymbol{\phi}_i] \log p(x_i|\mu_k) + \text{const} \\
&= -\frac{\mu_k^2}{2\sigma^2} + \sum_{i=1}^{n} \phi_{ik} \frac{-(x_i - \mu_k)^2}{2} + \text{const} \\
&= -\frac{\mu_k^2}{2\sigma^2} + \sum_{i=1}^{n} \phi_{ik} x_i \mu_k - \frac{\phi_{ik} \mu_k^2}{2} + \text{const} \\
&= \mu_k \left( \sum_{i=1}^{n} \phi_{ik} x_i \right) - \mu_k^2 \left( \frac{1}{2\sigma^2} + \frac{\sum_{i=1}^{n} \phi_{ik}}{2} \right) + \text{const} \\
&= -\frac{1}{2} \left( \frac{1}{\sigma^2} + \sum_{i=1}^{n} \phi_{ik} \right) \left( \mu_k^2 - \frac{2 \sum_{i=1}^{n} \phi_{ik} x_i}{1/\sigma^2 + \sum_{i=1}^{n} \phi_{ik}} \mu_k \right) + \text{const.}
\end{aligned}$$

The density is therefore

$$q(\mu_k) \propto \sqrt{\frac{1/\sigma^2 + \sum_{i=1}^n \phi_{ik}}{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{1}{\sigma^2} + \sum_{i=1}^n \phi_{ik}\right)\left(\mu_k - \frac{\sum_{i=1}^n \phi_{ik} x_i}{1/\sigma^2 + \sum_{i=1}^n \phi_{ik}}\right)^2\right).$$

It can be seen that $q(\mu_k)$ is a Gaussian distribution, so our variational updates for $m_k$ and $s_k^2$ are its mean and variance:

$$m_k = \frac{\sum_{i=1}^n \phi_{ik} x_i}{1/\sigma^2 + \sum_{i=1}^n \phi_{ik}}, \qquad s_k^2 = \frac{1}{1/\sigma^2 + \sum_{i=1}^n \phi_{ik}}.$$

We can now formulate the CAVI algorithm (Algorithm 3), which simply iterates the cluster assignment probabilities $\phi_{ik}$ and the variational density parameters $m_k$ and $s_k^2$ until the ELBO converges.

```
Data: Data $\boldsymbol{x}$, Number of Gaussian components $K$, Hyperparameter value
      $\sigma^2$
Result: Optimal variational factors $q(\mu_k; m_k, s_k^2)$ and $q(c_i; \boldsymbol{\phi_i})$
begin
    Randomly initialize parameters $\boldsymbol{m}, \boldsymbol{s}^2$ and $\boldsymbol{\phi}$;
    while ELBO has not converged do
        for $i = 1$ to $n$ do
            Set $\phi_{ik} \propto \exp\left(\mathbb{E}_{\mu_k}[\mu_k; m_k, s_k^2] x_i - \mathbb{E}_{\mu_k^2}[\mu_k^2; m_k, s_k^2]/2\right)$;
        end
        for $k = 1$ to $K$ do
            Set $m_k = \frac{\sum_i \phi_{ik} x_i}{1/\sigma^2 + \sum_i \phi_{ik}}$;
            Set $s_k^2 = \frac{1}{1/\sigma^2 + \sum_i \phi_{ik}}$;
        end
        Compute $ELBO(\boldsymbol{m}, \boldsymbol{s}^2, \boldsymbol{\phi})$;
    end
    Return $q(\boldsymbol{m}, \boldsymbol{s}^2, \boldsymbol{\phi})$;
end
```

**Algorithm 3:** CAVI Algorithm for Bayesian mixture of Gaussians

## 3.7 Amortized Inference

Now consider the case where we have $K$ data points, each with dimensionality $N$. We denote the set of data points as $\boldsymbol{X} = (\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(K)})^\top$, where $\boldsymbol{x}^{(i)} = (x_1^{(i)}, \dots, x_N^{(i)}) \in \mathbb{R}^N$, $i = 1, \dots, K$. One disadvantage of mean field variational inference is that a specific set of variational parameters needs to be derived and optimized for each of these data points. This can be computationally expensive for large datasets, and is because the parametrisation of the posterior $p(\boldsymbol{z}|\boldsymbol{x})$ changes as the data point $\boldsymbol{x}$ changes. We therefore denote our set of latent variable points as $\boldsymbol{Z} = (\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(K)})^\top$, where $\boldsymbol{z}^{(i)} = (z_1^{(i)}, \dots, z_M^{(i)}) \in \mathbb{R}^M$, $i = 1, \dots, K$. In mean-field variational inference, each latent variable $z_j^{(i)}$ has its own individual set of parameters $\phi_j^{(i)}$, so overall there would be $M \times N$ sets of parameters.

Amortized inference resolves this issue by using a single, constant set of parameters for all data points, adding the data point itself as an input to the variational distribution [Zhang et al., 2017]. Our variational distribution therefore conditions on the observation, taking the form

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}).$$

Figures 3.1 and 3.2 highlight the difference between mean-field and amortized variational inference.

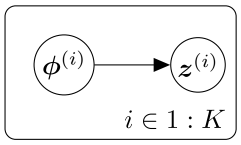


Figure 3.1: This is a DAG representing Mean-Field Variational Inference. For each of the $K$ datasets, a set of parameter sets $\boldsymbol{\phi}^{(i)}$ corresponding to each latent variable point $\boldsymbol{z}^{(i)}$ has to be found. $\boldsymbol{\phi}^{(i)}$ is $M$-dimensional, so there is a total of $M \times K$ sets of parameters.

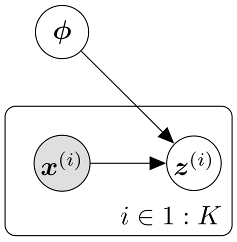


Figure 3.2: This DAG represents Amortized Variational Inference. Here, there is only one set of variational parameters $\boldsymbol{\phi}$, and each data point $\boldsymbol{x}^{(i)}$ is used as an input in the variational posterior $q_{\boldsymbol{\phi}}(\boldsymbol{z}|\boldsymbol{x})$ to find $\boldsymbol{z}^{(i)}$.

Clearly, a very complex variational distribution is required to model such a structure, so it often takes the form of a neural network with $\boldsymbol{x}$ as an input. This method is often used in deep learning due to the significant amount of data required to train such a network.

Now recall that $p(\boldsymbol{x})$ represents the marginal likelihood, or 'true' distribution of the data. This is typically never available, we often instead represent it with the distribution of a sample dataset. Denoting the sample dataset density function as $q^*(\boldsymbol{x})$, we want to optimize $\phi$ across the observations from our dataset, so our objective now is to choose parameters $\phi$ such that the expected KL divergence with respect to $q^*(\boldsymbol{x})$ is minimized:

$$\begin{aligned}
\phi &= \arg\min_{\phi} \mathbb{E}_{q^*(\boldsymbol{x})} KL(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z}|\boldsymbol{x})) \\
&= \arg\min_{\phi} \mathbb{E}_{q^*(\boldsymbol{x})q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) - \log p(\boldsymbol{z}|\boldsymbol{x})\right] \\
&= \arg\min_{\phi} \mathbb{E}_{q^*(\boldsymbol{x})q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) - \log \frac{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}{p(\boldsymbol{x})}\right] \\
&= \arg\min_{\phi} \left(\mathbb{E}_{q^*(\boldsymbol{x})q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) - \log p(\boldsymbol{x}|\boldsymbol{z}) - \log p(\boldsymbol{z})\right] + \log p(\boldsymbol{x})\right).
\end{aligned}$$

Again, we cannot evaluate this expression as $\log p(\boldsymbol{x})$ is intractable, so we rearrange the terms to form the evidence lower bound, which we want to maximise to minimise the KL divergence.

$$\begin{aligned}
ELBO(q) &= \mathbb{E}_{q^*(\boldsymbol{x})}[\log p(\boldsymbol{x}) - KL(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z}|\boldsymbol{x}))] \\
&= -\mathbb{E}_{q^*(\boldsymbol{x})q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[\log q_\phi(\boldsymbol{z}|\boldsymbol{x}) - \log p(\boldsymbol{x}|\boldsymbol{z}) - \log p(\boldsymbol{z})\right] \\
&= \mathbb{E}_{q^*(\boldsymbol{x})q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[\log p(\boldsymbol{x}|\boldsymbol{z}) + \log p(\boldsymbol{z}) - \log q_\phi(\boldsymbol{z}|\boldsymbol{x})\right] \\
&= \mathbb{E}_{q^*(\boldsymbol{x})} \left[\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p(\boldsymbol{x}|\boldsymbol{z})] - KL(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z}))\right].
\end{aligned}$$

We take the negative to form the minimization problem of the negative evidence lower bound $NELBO(q)$:

$$\min_{\phi} NELBO(q) = \mathbb{E}_{q^*(\boldsymbol{x})} \left[-\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p(\boldsymbol{x}|\boldsymbol{z})] + KL(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z}))\right]. \quad (3.7.1)$$

Since $\phi$ represents the parameters of a neural network, it is inefficient to find the specific weight values during back-propagation: we are more interested in training the network to optimality, which occurs when we minimize the objective function.

Hence, it is more appropriate to write the optimization problem with min than with arg min.

In deep learning, the likelihood term $p(\boldsymbol{x}|\boldsymbol{z})$ is often represented as a neural network parametrized by $\theta$: $p_\theta(\boldsymbol{x}|\boldsymbol{z})$. This network is optimized alongside the variational distribution, in a formation known as the variational autoencoder.

## 3.8 Example: Variational Autoencoder

A variational autoencoder (Figure 3.3) is a model consisting of two simultaneously trained neural networks: an encoder representing the posterior distribution $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ that "compresses" a data point $\boldsymbol{x} = (x_1, \ldots, x_N)^\top$ into a lower dimensional latent representation $\boldsymbol{z}$, and a decoder representing the likelihood distribution $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ that "reconstructs" the data point from the latent variable [Kingma and Welling, 2013]. It has two main purposes: lower-dimensional representation learning of data and data generation. Typically, the prior $p(\boldsymbol{z})$ is simply a standard multivariate normal distribution with dimensionality equal to that of the latent variable point $\boldsymbol{z} = (z_1, \ldots, z_M)^\top$: $\mathcal{N}(0, I_{M\times M})$. We now reiterate our optimization problem in equation (3.7.1), this time including the optimization of the decoder parameters $\theta$:

$$\min_{\phi,\theta} \mathbb{E}_{q^*(\boldsymbol{x})}\left[-\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] + KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z}))\right]. \tag{3.8.1}$$

The first term is the negative likelihood, which is analogous to the reconstruction error which we want to minimize. The KL divergence between the variational posterior and the true prior distribution acts as a regularizer term, encouraging the output of the posterior to be similar to $p(\boldsymbol{z})$, often a standard multivariate normal distribution. Without it, the encoder would learn to segregate distinct data types in separate regions of the Euclidean plane, which runs contrary to the randomness of a probability distribution. Due to this regularizer term, we can generate new data $\boldsymbol{x}$ by sampling $\boldsymbol{z} \sim p(\boldsymbol{z})$ and feeding it through the decoder [Doersch, 2016]. This is illustrated in Figure 3.4.

At our current formulation, we are trying to represent the variational posterior distribution with a deterministic neural network, so for any given data point $\boldsymbol{x}$, the encoder will always output the same $\boldsymbol{z}$. The solution is to add a noise distribution to the model to make it probabilistic. Instead of the encoder outputting posterior sample $\boldsymbol{z}$ directly, we configure it to output a mean vector $\boldsymbol{\mu} = (\mu_1, \mu_2, \cdots, \mu_M)^\top$ and variance vector $\boldsymbol{\sigma}^2 = (\sigma_1^2, \sigma_2^2, \cdots, \sigma_M^2)^\top$, each with dimensions equal to that of latent variable $\boldsymbol{z}$. We then define $\boldsymbol{z}$ as an output from a multivariate normal distribution with means and variances as specified by the encoder output:

$$q_\phi(\boldsymbol{z}|\boldsymbol{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 I_{M\times M}).$$

Often in practice, this is achieved by sampling random standard normal noise $\epsilon$, multiplying it by the variance and adding the result to the mean:

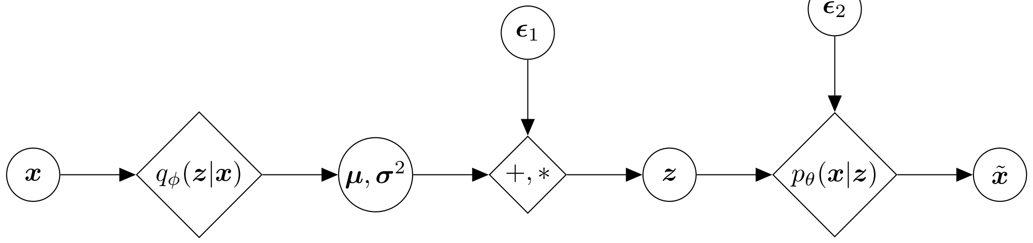
$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, I_{M\times M}), \qquad \boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\epsilon}\cdot\boldsymbol{\sigma}^2.$$

A similar process of adding random noise is used for the decoder network representing the likelihood distribution $p_\theta(\boldsymbol{x}|\boldsymbol{z})$, but the parametrisation of the distribution is chosen depending on the nature of the data. For example, for most continuous data, we can use a multivariate normal parametrisation similar to our variational posterior. For binary data, a sigmoid output layer is specified in the neural network and the likelihood distribution is expressed as a Bernoulli distribution with probabilities given by the network output.

Since we have the explicit form of the multivariate normal $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ and $p(\boldsymbol{z})$ densities, the KL divergence term can be calculated through the equation:

$$KL(q(\boldsymbol{z}|\boldsymbol{x})\|p(\boldsymbol{z})) = \frac{1}{2}\sum_{i=1}^{k}\left(\sigma_i^2 + \mu_i^2 - \log(\sigma_i^2) - 1\right).$$

We can now evaluate and minimise the $NELBO$ in equation (3.8.1) as we have an explicit parametrisation of all of the terms.



$\boldsymbol{x}$
$q_\phi(\boldsymbol{z}|\boldsymbol{x})$
$\boldsymbol{\mu}, \boldsymbol{\sigma}^2$
$\boldsymbol{\epsilon}_1$
$+, *$
$\boldsymbol{z}$
$\boldsymbol{\epsilon}_2$
$p_\theta(\boldsymbol{x}|\boldsymbol{z})$
$\tilde{\boldsymbol{x}}$

Figure 3.3: A simple DAG representing a Variational Autoencoder. An arbitrary data point $\boldsymbol{x}$ is passed through the encoder $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ to produce a mean vector $\boldsymbol{\mu}$ and a variance vector $\boldsymbol{\sigma}^2$. Random noise $\boldsymbol{\epsilon}_1$ is sampled from $\mathcal{N}(0, I_{M\times M})$ and transformed to generate $\boldsymbol{z}$: $\boldsymbol{z} = \boldsymbol{\mu}+\boldsymbol{\epsilon}\cdot\boldsymbol{\sigma}^2$. This latent variable $\boldsymbol{z}$ is passed through the decoder $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ to reconstruct the data point as $\tilde{\boldsymbol{x}}$.

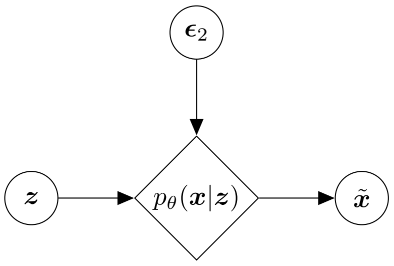


Figure 3.4: To generate new data $\tilde{\boldsymbol{x}}$ similar to existing data $\boldsymbol{x}$, we sample latent variable $\boldsymbol{z}$ from the prior distribution $p(\boldsymbol{z})$ and pass it through the decoder $p_\theta(\boldsymbol{x}|\boldsymbol{z})$.

## 3.9 Problems with Implicit Distributions

Now we return to our original Bayesian context in which the likelihood distribution is not represented by a neural network: $p(\boldsymbol{x}|\boldsymbol{z})$. The above sections assume that the prior $p(\boldsymbol{z})$, likelihood $p(\boldsymbol{x}|\boldsymbol{z})$ and variational posterior $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ have an explicit form. However, there are two main scenarios in which at least one of these distribution are implicit, that is, the parametrisation of their density is difficult to numerically evaluate but we are able to easily generate samples from their distributions. This poses problems with optimization as the objective function becomes difficult to compute.

### *3.9.1 Implicit Prior and/or Variational Posterior*

In the first scenario, at least one of the prior $p(\boldsymbol{z})$ or variational posterior $q_\phi(\boldsymbol{z}|bmx)$ distributions is implicit, but the likelihood is known. An implicit prior is rare but may occur in posterior inference. The implicit variational posterior is more common (in both posterior inference and data generation), detailed in "Adversarial Variational Bayes" by Mescheder et al. [2017].

Typically in amortized inference and variational autoencoders, the variational posterior sample $\boldsymbol{z}$ is sampled from a multivariate normal distribution with mean and variance defined by the variational network. The problem with this representation is the lack of dependencies between the latent variables and the inability to model multi-modal or flexible densities. In his paper, Mescheder adds random noise $\epsilon \sim \pi(\epsilon)$ as additional inputs to the variational network, training the network to directly output $\boldsymbol{z}$. $\pi(\epsilon)$ is a typical noise distribution, such as $\mathscr{N}(0, I_{p\times p})$ where $p$ is the desired number of noise inputs. This added noise allows the probabilistic nature of a probability density to be represented by a deterministic neural network. However, note that the neural network does not output the probability density $q_\phi(\boldsymbol{z}|\boldsymbol{x})$, rather it outputs distribution sample $\boldsymbol{z} \sim q_\phi(\boldsymbol{z}|\boldsymbol{x})$. Retaining the network parameters $\phi$, we denote the generator of posterior samples as $\mathscr{G}_\phi(\epsilon; \boldsymbol{x})$. A diagram illustrating this is shown in Figure 3.5 below.

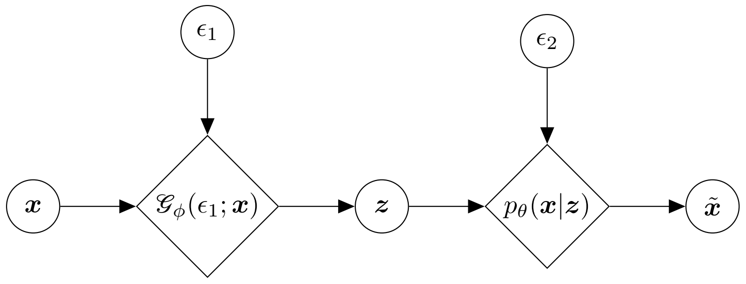

$\epsilon_1$
$\epsilon_2$
$\boldsymbol{x}$
$\mathscr{G}_\phi(\epsilon_1; \boldsymbol{x})$
$\boldsymbol{z}$
$p_\theta(\boldsymbol{x}|\boldsymbol{z})$
$\tilde{\boldsymbol{x}}$

Figure 3.5: This diagram depicts the "Adversarial Variational Bayes" formulation of the variational autoencoder. Rather than transforming random noise $\epsilon_1$ according to the mean vector $\boldsymbol{\mu}$ and variance vector $\boldsymbol{\sigma}$ output of the variational posterior $q_\phi(z|x)$, we add the noise to the encoder network $\mathscr{G}_\phi(\epsilon_1; \boldsymbol{x})$ directly as an additional input. The likelihood distribution $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ has the same explicit representation as in Figures 3.3 and 3.4.

Due to the complex nature of the neural network, it is difficult to numerically compute the explicit form of $q_\phi(\boldsymbol{z}|\boldsymbol{x})$ in this representation, but we are able to easily generate samples by feeding data and noise through the network, hence its implicit nature.

Now again recall our optimization problem from line 3.7.1:

$$\min_{\phi,\theta} \mathbb{E}_{q^*(\boldsymbol{x})} \left[ -\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})] + KL(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})) \right].$$

When either the prior or variational posterior is implicit, this expression is difficult to evaluate as we are unable to calculate $KL(q_\phi(\boldsymbol{z}|\boldsymbol{x}) \| p(\boldsymbol{z})) = \mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})} \left[ \log \frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z})} \right]$. We therefore resort to density ratio estimation techniques (Chapter 4) to approximate $\frac{q_\phi(\boldsymbol{z}|\boldsymbol{x})}{p(\boldsymbol{z})}$, using only samples from the two densities.

### *3.9.2 Implicit Likelihood*

When we apply amortized variational inference to an implicit likelihood distribution, the optimization problem is intractable even with density ratio estimation, as it is difficult to evaluate the term $-\mathbb{E}_{q_\phi(\boldsymbol{z}|\boldsymbol{x})}[\log p_\theta(\boldsymbol{x}|\boldsymbol{z})]$. This can occur in posterior inference or in some data generation algorithms where an implicit generative function $G(\epsilon; \boldsymbol{x})$ is used to represent the likelihood distribution $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ [Dumoulin et al., 2016]. An example of the latter case is illustrated in Figure 3.6 on the next page.

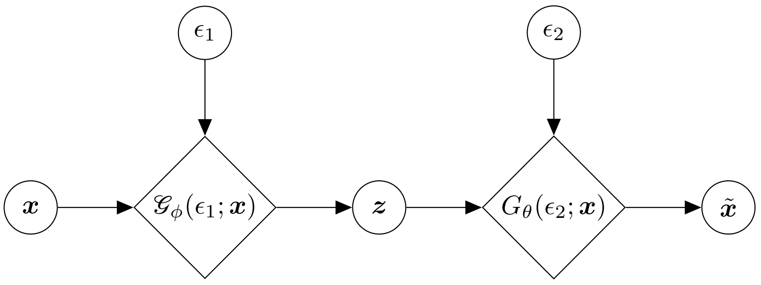

$\epsilon_1$
$\epsilon_2$
$\boldsymbol{x}$
$\mathscr{G}_\phi(\epsilon_1;\boldsymbol{x})$
$\boldsymbol{z}$
$G_\theta(\epsilon_2;\boldsymbol{x})$
$\tilde{\boldsymbol{x}}$

Figure 3.6: This diagram depicts a variation of "Adversarial Variational Bayes", in which noise is additionally added to the input of the generator of likelihood distribution samples $G_\theta(\epsilon_2;\boldsymbol{x})$. The likelihood distribution $p_\theta(\boldsymbol{x}|\boldsymbol{z})$ is therefore implicit and consequently, its corresponding term in the optimisation problem cannot be evaluated.

We rephrase the optimisation problem, instead minimising the reverse KL divergence between the two joint distributions: $KL(q(\boldsymbol{z},\boldsymbol{x})\|p(\boldsymbol{z},\boldsymbol{x})) = \mathbb{E}_{q(\boldsymbol{z},\boldsymbol{x})}\left[\log\frac{q(\boldsymbol{z},\boldsymbol{x})}{p(\boldsymbol{z},\boldsymbol{x})}\right]$ [Tran et al., 2017]. We begin deriving this by restating the original objective of minimizing the KL divergence between the true and variational posterior distributions, but this time we apply Bayes' theorem to formulate the joint densities:

$$\begin{aligned}\min_q \mathbb{E}_{q^*(\boldsymbol{x})}KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})\|p_\theta(\boldsymbol{z}|\boldsymbol{x})) &= \min_\phi \mathbb{E}_{q^*(\boldsymbol{x})q(\boldsymbol{z}|\boldsymbol{x})}\log\frac{q(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{x})}{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})}\\ &= \min_\phi \mathbb{E}_{q^*(\boldsymbol{x})q(\boldsymbol{z}|\boldsymbol{x})}\log\frac{q(\boldsymbol{z}|\boldsymbol{x})q^*(\boldsymbol{x})}{p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{z})} + \mathbb{E}_{q^*(\boldsymbol{x})}\log\frac{p(\boldsymbol{x})}{q^*(\boldsymbol{x})}\\ &= \min_\phi \mathbb{E}_{q^*(\boldsymbol{x})q(\boldsymbol{z}|\boldsymbol{x})}\log\frac{q(\boldsymbol{z},\boldsymbol{x})}{p(\boldsymbol{z},\boldsymbol{x})} - KL(q^*(\boldsymbol{x})\|p(\boldsymbol{x})).\end{aligned}$$

Recall that $p(\boldsymbol{x})$ is typically unavailable, so we are unable to evaluate $KL(q^*(\boldsymbol{x})\|p(\boldsymbol{x}))$ even with density ratio estimation techniques. Since it is constant, we add it to both sides of the equation, leading to the following expression for $NELBO(q)$:

$$\begin{aligned}NELBO(q) &= KL(q^*(\boldsymbol{x})\|p(\boldsymbol{x})) + \mathbb{E}_{q^*(\boldsymbol{x})}KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})\|p_\theta(\boldsymbol{z}|\boldsymbol{x}))\\ &= \mathbb{E}_{q^*(\boldsymbol{x})q_\phi(\boldsymbol{z}|\boldsymbol{x})}\log\frac{q(\boldsymbol{z},\boldsymbol{x})}{p(\boldsymbol{z},\boldsymbol{x})}.\end{aligned}$$

This expression attains a minimum at $KL(q^*(\boldsymbol{x})\|p(\boldsymbol{x}))$ when $\mathbb{E}_{q^*(\boldsymbol{x})}KL(q_\phi(\boldsymbol{z}|\boldsymbol{x})\|p_\theta(\boldsymbol{z}|\boldsymbol{x}))$, that is, the true and variational posteriors are equivalent with respect to the distribution of the data set. We therefore have the following lower bound:

$$KL(q^*(\boldsymbol{x})\|p(\boldsymbol{x})) \leq \mathbb{E}_{q^*(\boldsymbol{x})q_\phi(\boldsymbol{z}|\boldsymbol{x})}\log\frac{q(\boldsymbol{z},\boldsymbol{x})}{p(\boldsymbol{z},\boldsymbol{x})}.$$

Our objective now is to minimize this NELBO with respect to our variational parameters $\phi$, which can be approximated with density ratio estimation techniques:

$$\min_{\phi} \mathbb{E}_{q^*(\boldsymbol{x})q_\phi(\boldsymbol{z}|\boldsymbol{x})} \log \frac{q(\boldsymbol{z},\boldsymbol{x})}{p(\boldsymbol{z},\boldsymbol{x})}. \tag{3.9.1}$$

For consistency, we have retained the denotation of our variational posterior loss function as $NELBO(q)$. Note that the likelihood parameters no longer appear in the objective function, hence this expression on its own is only sufficient for posterior inference. The objective problem no longer has an explicit likelihood term, so it can be used with an implicit likelihood. In fact, since density ratio estimation only requires samples from the two distributions, both the prior and variational posterior densities can also be implicit, hence the "Adversarial Variational Bayes" representation (Figure 3.5) of the variational network can be used.

In data generation (Figure 3.6), we would additionally have to maximize the forward KL divergence with respect to the likelihood parameters [Tiao et al., 2018]:

$$\min_{\theta} \mathbb{E}_{p(\boldsymbol{z})p_\theta(\boldsymbol{x}|\boldsymbol{z})} \log \frac{p(\boldsymbol{z},\boldsymbol{x})}{q(\boldsymbol{z},\boldsymbol{x})}. \tag{3.9.2}$$

In this scenario, separate density ratio estimators would be used for equations (3.9.1) and (3.9.2), as we have taken the expectation of the density ratios with respect to different distributions.

# Chapter 4

# Density Ratio Estimation

In this chapter, we derive two common methods of density ratio estimation used with implicit models [Mohamed and Lakshminarayanan, 2016; Sugiyama et al., 2012]: class probability estimation and divergence minimisation, applying them to both the implicit prior/posterior and implicit likelihood objective functions to formulate bi-level optimization problems. In this thesis, we use Huszar's terminology, referring to the former objective as "prior-contrastive" and the latter as "joint-contrastive" [Huszár, 2017].

## 4.1 Class Probability Estimation

### *4.1.1 Derivation*

Firstly, consider the problem of estimating the density ratio between two arbitrary distributions $q(u)$ and $p(u)$: $\frac{q(u)}{p(u)}$. We take $m$ samples from $p(u)$: $U_p = \{u_1^{(p)}, \ldots, u_m^{(p)}\}$ and label them with $y = 0$, then we take $n$ samples from $q(u)$: $U_q = \{u_1^{(q)}, \ldots, u_n^{(q)}\}$ and label them with $y = 1$. Therefore, $p(u) = P(u|y = 0)$ and $q(u) = P(u|y = 1)$. By applying Bayes' theorem, we derive an expression for the density ratio:

$$\begin{aligned}
\frac{q(u)}{p(u)} &= \frac{P(u|y=1)}{P(u|y=0)} \\
&= \frac{P(y=1|u)P(u)}{P(y=1)} \Bigg/ \frac{P(y=0|u)P(u)}{P(y=0)} \\
&= \frac{P(y=1|u)}{P(y=0|u)} \times \frac{P(y=0)}{P(y=1)} \\
&= \frac{P(y=1|u)}{P(y=0|u)} \times \frac{n}{m}.
\end{aligned}$$

Often in practice, $m = n$, so the density ratio simplifies to:

$$\frac{q(u)}{p(u)} = \frac{P(y=1|u)}{P(y=0|u)}$$

which is the ratio of the probability that an arbitrary sample $u$ was taken from the distribution $q(u)$ to the probability that is was taken from $p(u)$. If we define a discriminator function $D(u) \simeq P(y = 1|u)$ that estimates these probabilities, then our density ratio can be expressed in terms of this discriminator function:

$$\frac{q(u)}{p(u)} \simeq \frac{D(u)}{1 - D(u)}.$$

The discriminator function typically takes the form of a neural network with parameters $\alpha$ trained with Bernoulli loss [Sugiyama et al., 2012]:

$$\min_{\alpha} L_D = -\mathbb{E}_{q(u)}[\log D_\alpha(u)] - \mathbb{E}_{p(u)}[\log(1 - D_\alpha(u))].$$

**Lemma 4.1.1.** *The discriminator reaches optimality at* $D^*_\alpha(u) = \frac{q(u)}{q(u)+p(u)}$, *minimizing its Bernoulli loss[Goodfellow et al., 2014].*

*Proof.* First we write the discriminator loss function in integral form:

$$\min_{\alpha} L_D = -\int q(u) \log D_\alpha(u) du - \int p(u) \log(1 - D_\alpha(u)) du.$$

Now we take the functional derivative of the expression and equate it to 0:

$$\frac{\partial L_D}{\partial D_\alpha(u)} = 0$$

$$\begin{aligned}
-\frac{q(u)}{D_\alpha(u)} + \frac{p(u)}{1 - D_\alpha(u)} &= 0 \\
D_\alpha(u)(q(u) + p(u)) &= q(u) \\
D_\alpha(u) &= \frac{q(u)}{q(u) + p(u)}.
\end{aligned}$$

Observing that $q(u)$ and $p(u)$ are densities, this expression is a minimum as

$$\begin{aligned}
\frac{\partial^2 L_D}{\partial D^2_\alpha(u)} &= \frac{q(u)}{D^2_\alpha(u)} + \frac{p(u)}{(1 - D_\alpha(u))^2} \\
&> 0.
\end{aligned}$$

□

**Corollary 4.1.2.** $D^*_\alpha(u)$ *is bound in* $(0, 1)$, *so a sigmoid activation function is ideal for its output layer.*

**Remark 4.1.3.** *When $q(u) = p(u)$, $D^*_\alpha(u) = \frac{1}{2}$ and $L_{D^*} = \log 4$.*

The optimal discriminative loss can be used to check posterior convergence.

### *4.1.2 Prior-Contrastive Algorithm*

We now turn to our problem in line (3.7.1) of minimizing the negative ELBO:

$$\begin{aligned}\min_q NELBO(q) &= \min_\phi \mathbb{E}_{q^*(x)} \left[-\mathbb{E}_{q_\phi(z|x)}[\log p(x|z)] + KL(q_\phi(z|x)\|p(z))\right] \\ &= \min_\phi -\mathbb{E}_{q^*(x)q_\phi(z|x)}[\log p(x|z)] + \mathbb{E}_{q^*(x)q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p(z)}\right],\end{aligned} \tag{4.1.1}$$

which requires us to find the density ratio between $q_\phi(z|x)$ and $p_\theta(z)$.
Again, if we label samples from $q_\phi(z|x)$ with $y = 1$ and samples from $p(z)$ with $y = 0$, we have the density ratio expression:

$$\frac{q_\phi(z|x)}{p(z)} = \frac{P(y = 1|z)}{P(y = 0|z)}.$$

We now define a discriminator function that calculates the probability that an arbitrary sample $z$ belongs to the variational posterior $q_\phi(z|x)$. Since the posterior changes depending on the observation $x$, we also amortize the discriminator by taking an additional input from the dataset $x$, as opposed to training multiple discriminators for each observation $x_i$ :

$$D_\alpha(z, x) \simeq P(y = 1|z).$$

As a binary classifier, this function can be trained by inputting an equal number of samples from both distributions and minimizing its Bernoulli loss:

$$\min_\alpha -\mathbb{E}_{q^*(x)q_\phi(z|x)}[\log D_\alpha(z, x)] - \mathbb{E}_{q^*(x)p_\theta(z)}[\log(1 - D_\alpha(z, x))]. \tag{4.1.2}$$

We can now express the expected log ratio in terms of the probability that a posterior sample is correctly classified by the discriminator:

$$\begin{aligned}\mathbb{E}_{q^*(x)q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p(z)}\right] &= \mathbb{E}_{q^*(x)q_\phi(z|x)}\left[\log \frac{P(y = 1|z)}{P(y = 0|z)}\right] \\ &\simeq \mathbb{E}_{q^*(x)q_\phi(z|x)}\left[\log \frac{D_\alpha(z, x)}{1 - D_\alpha(z, x)}\right].\end{aligned}$$

Our NELBO optimization objective in line 4.1.1 can now be written as:

$$\min_{\phi} -\mathbb{E}_{q^*(x)q_\phi(z|x)}[\log p(x|z)] + \mathbb{E}_{q^*(x)q_\phi(z|x)}\left[\log \frac{D_\alpha(z,x)}{1-D_\alpha(z,x)}\right]. \tag{4.1.3}$$

In practice, the algorithm cycles between optimizing the discriminator until convergence (with the generator parameters fixed) and taking optimization steps of the negative evidence lower bound (with the discriminator parameters fixed).
Since our variational posterior distribution is in the form of a generative neural network function, our optimization objectives 4.1.2 and 4.1.3 take the expressions:

$$\min_{\alpha} -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log D_\alpha(\mathscr{G}_\phi(\epsilon;x),x) - \mathbb{E}_{p_\theta(z)q^*(x)}[\log(1-D_\alpha(z,x))]$$

$$\min_{\phi} -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log p(x|\mathscr{G}_\phi(\epsilon;x))] + \mathbb{E}_{q^*(x)\pi(\epsilon)}\left[\log \frac{D_\alpha(\mathscr{G}_\phi(\epsilon;x),x)}{1-D_\alpha(\mathscr{G}_\phi(\epsilon;x),x)}\right].$$

Recall that if we apply this algorithm to a data generation problem, ie. the likelihood function also needs to be optimized, then in the posterior optimization function, we would additionally parametrize the likelihood with $\theta$ and optimize the same function with respect to those parameters. Also note that from equation (3.4.1) that the NELBO attains an optimal value equal to $-p(x)$, which is unknown. It is therefore inadequate to assess convergence using the value of the NELBO, we would additionally check that the discriminative loss is equal to $\log 4$, using remark 4.1.3. Pseudocode for this algorithm is shown in Algorithm 4 on the next page.

```
Data: Dataset $q^*(x)$, true (implicit) prior $p(z)$, true likelihood $p(x|z)$, noise
      distribution $\pi(\epsilon)$
Result: Optimized posterior generator $\mathscr{G}_\phi(\epsilon; x)$

for $j = 1$ to $J$ do
    for $k = 1$ to $K$ do
        Sample $\{\epsilon^{(i,k)}\}_{i=1}^{B} \sim \pi(\epsilon)$;
        Sample $\{z_p^{(i,k)}\}_{i=1}^{B} \sim p(z)$;
        Sample $\{x^{(i,k)}\}_{i=1}^{B} \sim q^*(x)$;
        foreach $\epsilon^{(i,k)}, x^{(i,k)}$ do
            Sample $z_q^{(i,k)} = \mathscr{G}(\epsilon^{(i,k)}; x^{(i,k)})$;
        end
        update $\alpha$ by optimization step on
          $\min_\alpha -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log D_\alpha(\mathscr{G}_\phi(\epsilon; x), x)] - \mathbb{E}_{p(z)q^*(x)}[\log(1 - D_\alpha(z, x))]$;
    end
    Sample $\{x^{(i)}\}_{i=1}^{B} \sim q^*(x)$;
    Sample $\{\epsilon^{(i)}\}_{i=1}^{B} \sim \pi(\epsilon)$;
    update $\phi$ by optimization step on
      $\min_\phi -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log p(x|\mathscr{G}_\phi(\epsilon; x))] + \mathbb{E}_{q^*(x)\pi(\epsilon)}\left[\log \frac{D_\alpha(\mathscr{G}_\phi(\epsilon;x),x)}{1-D_\alpha(\mathscr{G}_\phi(\epsilon;x),x)}\right]$;
end
```

**Algorithm 4:** Prior-Contrastive Class Probability Estimation

### *4.1.3 Joint-Contrastive Algorithm*

We now restate from line (3.9.1) our optimization objective when the likelihood distribution is implicit:

$$\min_{\phi} \mathbb{E}_{q^*(x)q_\phi(z|x)} \log \frac{q(z,x)}{p(z,x)},$$

or

$$\min_{\phi} \mathbb{E}_{q^*(x)q_\phi(z|x)} \log \frac{q_\phi(z|x)q^*(x)}{p(x|z)p(z)}.$$

Similar to the prior-contrastive case, we can label samples from $q(z,x)$ with $y=1$ and samples from $p(z,x)$ with $y=0$, leading to the density ratio expression:

$$\frac{q(z,x)}{p(z,x)} = \frac{P(y=1|z,x)}{P(y=0|z,x)}.$$

Again, we use a discriminator neural network $D_\alpha(z,x) = P(y=1|z,x)$ to determine the probability that samples $(z,x)$ came from the joint variational distribution $q(z,x)$. Using this discriminator function, our density ratio expression becomes:

$$\frac{q(z,x)}{p(z,x)} \simeq \frac{D_\alpha(z,x)}{1-D_\alpha(z,x)}.$$

The class probability algorithm again cycles between Bernoulli loss minimisation of the discriminator:

$$\min_{\alpha} -\mathbb{E}_{q^*(x)q_\phi(z|x)}[\log D_\alpha(z,x)] - \mathbb{E}_{p(z)p(x|z)}[\log(1-D_\alpha(z,x))] \tag{4.1.4}$$

and optimization of the variational posterior:

$$\min_{\phi} \mathbb{E}_{q^*(x)q_\phi(z|x)} \log \frac{D_\alpha(z,x)}{1-D_\alpha(z,x)}. \tag{4.1.5}$$

Using the posterior generator parametrization $\mathscr{G}_\phi(\epsilon;x)$, optimization objectives (4.1.4) and (4.1.5) become:

$$\min_{\alpha} -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log D_\alpha(\mathscr{G}_\phi(\epsilon;x),x)] - \mathbb{E}_{p(z)p(x|z)}[\log(1-D_\alpha(z,x))]$$

$$\min_{\phi} \mathbb{E}_{q^*(x)\pi(\epsilon)} \log \frac{D_\alpha(\mathscr{G}_\phi(\epsilon;x),x)}{1-D_\alpha(\mathscr{G}_\phi(\epsilon;x),x)}.$$

Using Remark 4.1.3, it can be seen that when $q(z,x) = p(z,x)$, the optimal generative loss is $\log \frac{1/2}{1-1/2} = 0$. Pseudocode for the joint-contrastive class probability estimation algorithm is shown in Algorithm 5 on the next page.

```
Data: Dataset $q^*(x)$, true (implicit) prior $p(z)$, true (implicit) likelihood
      $p(x|z)$, noise distribution $\pi(\epsilon)$
Result: Optimized posterior generator $\mathscr{G}_\phi(\epsilon; x)$
for $j = 1$ to $J$ do
    for $k = 1$ to $K$ do
        Sample $\{\epsilon^{(i,k)}\}_{i=1}^B \sim \pi(\epsilon)$;
        Sample $\{x_q^{(i,k)}\}_{i=1}^B \sim q^*(x)$;
        Sample $\{z_p^{(i,k)}\}_{i=1}^B \sim p(z)$;
        foreach $\epsilon^{(i,k)}, x^{(i,k)}$ do
            Sample $z_q^{(i,k)} = \mathscr{G}(\epsilon^{(i,k)}; x_q^{(i,k)})$;
        end
        foreach $z_p^{(i,k)}$ do
            Sample $x_p^{(i,k)} \sim p(x|z)$;
        end
        update $\alpha$ by optimization step on
          $\min_\alpha -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log D_\alpha(\mathscr{G}_\phi(\epsilon; x), x)] - \mathbb{E}_{p(z)p(x|z)}[\log(1 - D_\alpha(z, x))]$;
    end
    Sample $\{\epsilon^{(i)}\}_{i=1}^B \sim \pi(\epsilon)$;
    Sample $\{x_q^{(i)}\}_{i=1}^B \sim q^*(x)$;
    update $\phi$ by optimization step on
      $\min_\phi \mathbb{E}_{q^*(x)\pi(\epsilon)} \log \frac{D_\alpha(\mathscr{G}_\phi(\epsilon;x),x)}{1-D_\alpha(\mathscr{G}_\phi(\epsilon;x),x)}$;
end
```

**Algorithm 5:** Joint-Contrastive Class Probability Estimation

## 4.2 Divergence Minimisation

### *4.2.1 Derivation*

Recall from Definition 3.2.1 that an f-divergence is defined by a convex function $f$ such that $f(1) = 0$. A lower bound for the f-divergence in terms of a direct ratio estimator $r(u) \simeq \frac{q(u)}{p(u)}$ can be found using the following theorem [Nguyen et al., 2010].

**Theorem 4.2.1.** *If $f$ is a convex function with derivative $f'$ and convex conjugate $f^*$, and $\mathscr{R}$ is a class of functions with codomains equal to the domain of $f'$, then we have the lower bound for the f-divergence between distributions $p(u)$ and $q(u)$:*

$$D_f[p(u)\|q(u)] \geq \sup_{r \in \mathscr{R}} \{\mathbb{E}_{q(u)}[f'(r(u))] - \mathbb{E}_{p(u)}[f^*(f'(r(u)))]\},$$

*with equality when $r(u) = q(u)/p(u)$.*

We omit the proof of this theorem as it is beyond the scope of this thesis. Details of the proof can be found in Estimating divergence functionals and the likelihood ratio by convex risk minimization by XuanLong Nguyen.
The derivative and convex conjugate of $f(u) = u \log u$ are

$$f'(u) = 1 + \log u, \qquad f^*(u) = \exp(u - 1),$$

so the convex conjugate of the derivative is simply $f^*(f'(u)) = u$.
Using Theorem 4.2.1, we derive the following lower bound for the reverse KL divergence:

$$\begin{aligned} KL[q(u)\|p(u)] &= D_{RKL}(P\|Q) \\ &\geq \sup_{r \in \mathscr{R}} \{\mathbb{E}_{q(u)}[1 + \log r(u)] - \mathbb{E}_{p(u)}[r(u)]\}. \end{aligned} \tag{4.2.1}$$

Fixing the variational density $q(u)$, $KL(q(u)\|p(u))$ is constant, so recalling that the bound reaches equality when $r(u) = q(u)/p(u)$, we can represent the direct ratio estimator as a neural network parametrised by $\alpha$ and minimize the negative of expression (4.2.1) with respect to $\alpha$:

$$\min_{\alpha} L_r = -\mathbb{E}_{q(u)}[\log r_\alpha(u)] + \mathbb{E}_{p(u)}[r_\alpha(u)]. \tag{4.2.2}$$

Note we have removed the +1 term as it is independent of $\alpha$. Although it is evident from the theorem that $L_r$ is minimized when $r_\alpha(u) = \frac{q(u)}{p(u)}$, we verify this below:

**Lemma 4.2.2.** *The direct ratio estimator reaches optimality at* $r^*_\alpha(u) = \frac{q(u)}{p(u)}$*, minimizing its ratio loss.*

*Proof.* First we write the ratio loss function in integral form:

$$\min_\alpha L_r = -\int q(u) \log r_\alpha(u) + \int p(u) r_\alpha(u).$$

Now we take the functional derivative of the expression and equate it to 0:

$$\frac{\partial L_r}{\partial r_\alpha(u)} = 0$$

$$-\frac{q(u)}{r_\alpha(u)} + p(u) = 0$$

$$r_\alpha = \frac{q(u)}{p(u)}.$$

Observing that $q(u)$ is a density, this expression is a minimum as:

$$\begin{aligned} \frac{\partial^2 L_r}{\partial r^2_\alpha(u)} &= \frac{q(u)}{r^2_\alpha(u)} \\ &> 0. \end{aligned}$$

□

**Corollary 4.2.3.** $r^*_\alpha(u)$ *is bound in* $(0, \infty)$.

**Remark 4.2.4.** *When* $q(u) = p(u)$, $r^*_\alpha(u) = 1$ *and* $L_r = 1$.

Like Remark 4.1.3, we can use the optimal discriminative loss to check posterior convergence.

### *4.2.2 Prior-Contrastive Algorithm*

We begin by restating the optimization objective from line (4.1.1):

$$\min_{\phi} -\mathbb{E}_{q^*(x)q_\phi(z|x)}[\log p(x|z)] + \mathbb{E}_{q^*(x)q_\phi(z|x)}\left[\log \frac{q_\phi(z|x)}{p(z)}\right].$$

Applying Theorem 4.2.1, we have the lower bound for the reverse KL divergence:

$$KL[q_\phi(z|x)\|p(z)] \geq \sup_{\hat{r}\in\mathscr{R}}\{\mathbb{E}_{q_\phi(z|x)}[1+\log r(z)] - \mathbb{E}_{p(z)}[r(z)]\}.$$

Now we let our direct ratio estimator be a neural network parametrized by $\alpha$, and since $q_\phi(z|x)$ is dependent on the input $x \sim q^*(x)$, we add $x$ as an input and amortize the KL divergence across $q^*(x)$:

$$\mathbb{E}_{q^*(x)}KL[q_\phi(z|x)\|p(z)] \geq \sup_{\alpha}\{\mathbb{E}_{q^*(x)q_\phi(z|x)}[1+\log r_\alpha(z,x)] - \mathbb{E}_{q^*(x)p(z)}[r_\alpha(z,x)]\}.$$

Using line (4.2.2), our optimization problem for the direct ratio estimator becomes:

$$\min_{\alpha} -\mathbb{E}_{q^*(x)q_\phi(z|x)}[\log r_\alpha(z,x)] + \mathbb{E}_{q^*(x)p(z)}[r_\alpha(z,x)]. \tag{4.2.3}$$

Since $r_\alpha(z,x) \simeq \frac{q_\phi(z|x)}{p(z)}$, we write our NELBO optimization problem in terms of the direct ratio estimator:

$$\min_{\phi} -\mathbb{E}_{q^*(x)q_\phi(z|x)}[\log p(x|z)] + E_{q^*(x)q_\phi(z|x)}[\log r_\alpha(z,x)]. \tag{4.2.4}$$

Substituting the generator form of the posterior into lines (4.2.3) and (4.2.4), we have the bi-level optimization problem:

$$\min_{\alpha} -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log r_\alpha(\mathscr{G}_\phi(\epsilon;x),x)] + \mathbb{E}_{q^*(x)p(z)}[r_\alpha(z,x)]$$

$$\min_{\phi} -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log p(x|\mathscr{G}_\phi(\epsilon;x)] + E_{q^*(x)\pi(\epsilon)}[\log r_\alpha(\mathscr{G}(\epsilon;x),x)].$$

Similar to class probability estimation, the algorithm involves cycling between optimizing the ratio estimator until convergence and taking one gradient step of ELBO minimisation, as shown in Algorithm 6.

```
Data: Dataset $q^*(x)$, true (implicit) prior $p(z)$, true likelihood $p(x|z)$, noise
      distribution $\pi(\epsilon)$
Result: Optimized posterior generator $\mathscr{G}_\phi(\epsilon; x)$
for $j = 1$ to $J$ do
    for $k = 1$ to $K$ do
        Sample $\{\epsilon^{(i,k)}\}_{i=1}^{B} \sim \pi(\epsilon)$;
        Sample $\{z_p^{(i,k)}\}_{i=1}^{B} \sim p(z)$;
        Sample $\{x^{(i,k)}\}_{i=1}^{B} \sim q^*(x)$;
        foreach $\epsilon^{(i,k)}, x^{(i,k)}$ do
            Sample $z_q^{(i,k)} = \mathscr{G}(\epsilon^{(i,k)}; x^{(i,k)})$;
        end
        update $\alpha$ by optimization step on
          $\min_\alpha -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log r_\alpha(\mathscr{G}_\phi(\epsilon; x), x)] + \mathbb{E}_{q^*(x)p(z)}[r_\alpha(z; x)]$;
    end
    Sample $\{x^{(i)}\}_{i=1}^{B} \sim q^*(x)$;
    Sample $\{\epsilon^{(i)}\}_{i=1}^{B} \sim \pi(\epsilon)$;
    update $\phi$ by optimization step on
      $\min_\phi -\mathbb{E}_{q^*(x)\pi(\epsilon)}\left[\log p(x|\mathscr{G}_\phi(\epsilon; x)\right] + E_{q^*(x)\pi(\epsilon)}[\log r_\alpha(\mathscr{G}(\epsilon; x), x)]$;
end
```

**Algorithm 6:** Prior-Contrastive Divergence Minimisation

### 4.2.3 Joint-Contrastive Algorithm

First, we restate the problem from line (3.9.1) of minimizing the reverse KL divergence between the joint distributions:

$$\min_{\phi} \mathbb{E}_{q^*(x)q_\phi(z|x)} \log \frac{q(z,x)}{p(z,x)}.$$

Applying Theorem 4.2.1, a lower bound for our KL divergence is

$$KL[q(z,x)\|p(z,x)] \geq \sup_{r\in\mathscr{R}}\{\mathbb{E}_{q(z,x)}[1+\log r(z,x)] - \mathbb{E}_{p(z,x)}[r(z,x)]\}.$$

Note here we do not have to amortize our ratio estimator as the joint probability distribution already includes the dataset. Again we set our ratio estimator to take the form of a neural network parametrized by $\alpha$:

$$r_\alpha(z,x) \simeq \frac{q(z,x)}{p(z,x)}$$

so that the lower bound becomes

$$KL[q(z,x)\|p(z,x)] \geq \sup_{\alpha}\{\mathbb{E}_{q(z,x)}[1+\log r_\alpha(z,x)] - \mathbb{E}_{p(z,x)}[r_\alpha(z,x)]\}.$$

From line (4.2.2) we form the following optimization problem for the direct ratio estimator:

$$\min_{\alpha} -\mathbb{E}_{q^*(x)q_\phi(z|x)}[\log r_\alpha(z,x)] + \mathbb{E}_{p(z)p(x|z)}[r_\alpha(z,x)]. \tag{4.2.5}$$

The NELBO minimization problem in terms of the direct ratio estimator is:

$$\min_{\phi} \mathbb{E}_{q^*(x)q_\phi(z|x)}[\log r_\alpha(z,x)]. \tag{4.2.6}$$

Finally, we derive the bi-level optimization problem by writing the variational posterior in lines (4.2.5) and (4.2.6) in terms of their generator form:

$$\min_{\alpha} -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log r_\alpha(\mathscr{G}(\epsilon;x),x)] + \mathbb{E}_{p(z)p(x|z)}[r_\alpha(z,x)]$$

$$\min_{\phi} \mathbb{E}_{q^*(x)\pi(\epsilon)}[\log r_\alpha(\mathscr{G}(\epsilon;x),x)].$$

Again the algorithm involves cycling between multiple steps of optimizing the ratio estimator and taking a single optimization step of the NELBO.

```
Data: Dataset $q^*(x)$, true (implicit) prior $p(z)$, true (implicit) likelihood
      $p(x|z)$, noise distribution $\pi(\epsilon)$
Result: Optimized posterior generator $\mathscr{G}_\phi(\epsilon; x)$
for $j = 1$ to $J$ do
    for $k = 1$ to $K$ do
        Sample $\{\epsilon^{(i,k)}\}_{i=1}^B \sim \pi(\epsilon)$;
        Sample $\{x_q^{(i,k)}\}_{i=1}^B \sim q^*(x)$;
        Sample $\{z_p^{(i,k)}\}_{i=1}^B \sim p(z)$;
        foreach $\epsilon^{(i,k)}, x^{(i,k)}$ do
            Sample $z_q^{(i,k)} = \mathscr{G}(\epsilon^{(i,k)}; x_q^{(i,k)})$;
        end
        foreach $z_p^{(i,k)}$ do
            Sample $x_p^{(i,k)} \sim p(x|z)$;
        end
        update $\alpha$ by optimization step on
          $\min_\alpha -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log r_\alpha(\mathscr{G}(\epsilon; x), x)] + \mathbb{E}_{p(z)p(x|z)}[r_\alpha(z, x)]$;
    end
    Sample $\{\epsilon^{(i)}\}_{i=1}^B \sim \pi(\epsilon)$;
    Sample $\{x_q^{(i)}\}_{i=1}^B \sim q^*(x)$;
    update $\phi$ by optimization step on
      $\min_\phi \mathbb{E}_{q^*(x)\pi(\epsilon)}[\log r_\alpha(\mathscr{G}(\epsilon; x), x)]$;
end
```

**Algorithm 7:** Joint-Contrastive Divergence Minimisation

### *4.2.4 Alternative Derivation of Class Probability Estimation*

If we apply Theorem 4.2.1 with the function $f(u) = u \log u - (u+1)\log(u+1)$, the resulting bi-level optimization problem is equivalent to that of class probability estimation[Tiao et al., 2018]. Before deriving the corresponding f-divergence, we define the Jenson-Shannon divergence which will be used in the f-divergence expression.

**Definition 4.2.5.** The Jensen-Shannon divergence is a symmetric metric used to measure the discrepancy between two probability distributions[Fuglede and Topsoe, 2004]:

$$JS[p(u) \| q(u)] = \frac{1}{2} KL(p(u) \| m(u)) + \frac{1}{2} KL(q(u) \| m(u))$$

where $m(u) = \frac{q(u)+p(u)}{2}$.

We denote this f-divergence as $D_{GAN}$ after the generative adversarial networks which typically use the class probability estimation optimisation problem[Goodfellow et al., 2014].

$$\begin{aligned}
D_{GAN}(p\|q) &= \mathbb{E}_p \left[ \frac{q(u)}{p(u)} \log\left(\frac{q(u)}{p(u)}\right) - \left(\frac{q(u)}{p(u)} + 1\right) \log\left(\frac{q(u)}{p(u)} + 1\right) \right] \\
&= \int q(u) \log\left(\frac{q(u)}{p(u)}\right) - (q(u)+p(u)) \log\left(\frac{q(u)+p(u)}{p(u)}\right) du \\
&= \int q(u)(\log(q(u)) - \log(p(u))) - (q(u)+p(u)) \log(q(u)+p(u)) \\
&\quad + (q(u)+p(u)) \log p(u) \;\; du \\
&= \int q(u) \log\left(\frac{q(u)}{q(u)+p(u)}\right) + p(u) \log\left(\frac{p(u)}{q(u)+p(u)}\right) du \\
&= \int p(u) \log\left(\frac{2p(u)}{q(u)+p(u)}\right) - p(u)\log 2 + q(u) \log\left(\frac{2q(u)}{q(u)+p(u)}\right) \\
&\quad - q(u) \log 2 \;\; du \\
&= \int p(u) \log\left(\frac{p(u)}{m(u)} - \log 2\right) + q(u) \log\left(\frac{q(u)}{m(u)} - \log 2\right) du \\
&\text{where } m(u) = \frac{q(u)+p(u)}{2} \\
&= \mathbb{E}_p \left[ \log \frac{p(u)}{m(u)} - \log 2 \right] + \mathbb{E}_q \left[ \log \frac{q(u)}{m(u)} - \log 2 \right] \\
&= KL(p(u)\|m(u)) + KL(q(u)\|m(u)) - \log 4 \\
&= 2JS[p(u)\|q(u)] - \log 4.
\end{aligned}$$

Recall one of the conditions of $f$ to form an f-divergence is $f(1) = 0$, but in this case, $f(1) = -\log 4$, hence the constant term in our GAN divergence.

The derivative and convex conjugate of $f(u) = u \log u - (u+1)\log(u+1)$ are

$$f'(u) = \log \frac{u}{u+1} \qquad f^*(u) = -\log(1 - \exp u)$$

so the convex conjugate of the derivative is

$$f^*(f'(u)) = \log(u+1).$$

Using Theorem 4.2.1, we derive the following lower bound for the Jensen-Shannon divergence:

$$\begin{aligned} 2JS[p(u)\|q(u)] - \log 4 &= D_{GAN}(p\|q) \\ &\geq \sup_{r \in \mathscr{R}} \{\mathbb{E}_{q(u)}\left[\log \frac{r(u)}{r(u)+1}\right] - \mathbb{E}_{p(u)}[\log(r(u)+1)]\} \\ &= \sup_{D} \{\mathbb{E}_{q(u)}[\log D(u)] + \mathbb{E}_{p(u)}[\log(1 - D(u))]\} \end{aligned}$$

where $D(u) = \frac{r(u)}{r(u)+1}$. Maximisation of the lower bound, equivalent to minimising its negative, follows the optimization problem:

$$\min_{\alpha} -\mathbb{E}_{q(u)}[\log D(u)] - \mathbb{E}_{p(u)}[\log(1 - D(u))],$$

which is the discriminative loss of the class probability estimation approach. Equality of the bound is accomplished at the optimal function of

$$\begin{aligned} D^*(u) &= \frac{\frac{q(u)}{p(u)}}{\frac{q(u)}{p(u)} + 1} \\ &= \frac{q(u)}{q(u) + p(u)} \end{aligned}$$

which is the optimal discriminator of class probability estimation. Thus, the density ratio in terms of this discriminator function is

$$\frac{q(u)}{p(u)} \simeq \frac{D(u)}{1 - D(u)},$$

and therefore our previous formulation of class probability estimation by minimizing the Bernoulli loss of a discriminator function is the same as maximizing the lower bound of the "CPE" divergence between the two distributions.

# Chapter 5

# Initial Experiment - Inference, “Sprinkler” Example

In this chapter we conduct our first experiment comparing class probability estimation and divergence minimisation in both prior-contrastive and joint-contrastive formulations.

## 5.1 Problem Context

We use the simple “Continuous Sprinkler” experiment as described in “Variational Inference using Implicit Distributions” by Huszár [2017]. This experiment involves the hypothetical scenario of grass wetness as a result of two independent sources: rainfall and a sprinkler. The problem exhibits “explaining away” [P. Wellman and Henrion, 1993], a pattern of reasoning in which the confirmation of one cause of an effect reduces the need to discuss alternative causes. In this example, despite the two possible sources of wetness being independent, when we condition on the observation of wet grass, the causes become dependent, as it is correlated with the significance of either water source.

The latent variables $z_1$ and $z_2$ represent the sprinkler and the rain respectively, and the observation $x$ represents the wetness of the grass.

$$p(z_1, z_2) \sim \mathcal{N}(0, \sigma^2 I_{2\times 2})$$

$$p(x|\boldsymbol{z}) \sim EXP(3 + \max(0, z_1)^3 + \max(0, z_2)^3).$$

The prior-contrastive algorithm requires the likelihood function of the likelihood distribution:

$$p(x|\boldsymbol{z}) = -\log(3 + \max(0, z_1)^3 + \max(0, z_2)^3) - \frac{x}{3 + \max(0, z_1)^3 + \max(0, z_2)^3}.$$

The likelihood function is not required for the joint-contrastive algorithms as it is assumed to be implicit.

The unnormalised true posterior is derived as shown below. The lack of normalisation does not affect the variational distribution as the true posterior is assumed to be unknown, and the algorithm only uses samples from the prior and likelihood.
**Posterior Calculation:**

$$\begin{aligned} p(z_1, z_2|x) &= \exp(\log p(z_1, z_2) + \log p(x|z_1, z_2)) \\ &\propto \exp\left(\frac{1}{2}\boldsymbol{z}^T(\sigma^{-2}I_{2\times 2})\boldsymbol{z} - \log(3 + \max(0, z_1)^3 + \max(0, z_2)^3) \right. \\ &\quad \left. - \frac{x}{3 + \max(0, z_1)^3 + \max(0, z_2)^3}\right) \\ &\propto \exp\left(\frac{1}{2\sigma^2}(z_1^2 + z_2^2) - \log(3 + \max(0, z_1)^3 + \max(0, z_2)^3) \right. \\ &\quad \left. - \frac{x}{3 + \max(0, z_1)^3 + \max(0, z_2)^3}\right). \end{aligned} \tag{5.1.1}$$

In the experiment, we let $\sigma^2 = 2$, and consider observations $x = 0, 5, 8, 12, 50$. To plot the true (unnormalised) posterior $p(\boldsymbol{z}|x)$, we use a range of equally spaced points from $(z_1, z_2) = [(-5, -5), \ldots, (5, 5)]$ to form the log prior values and log likelihood values for each observation value, and graph the exponential of the sum of these values. The expressions are as derived above. The plots are as shown in Figure 5.1 below. We are not concerned with the lack of normalisation as the purpose of the plots is to depict the shape and relative density of the posterior distribution.

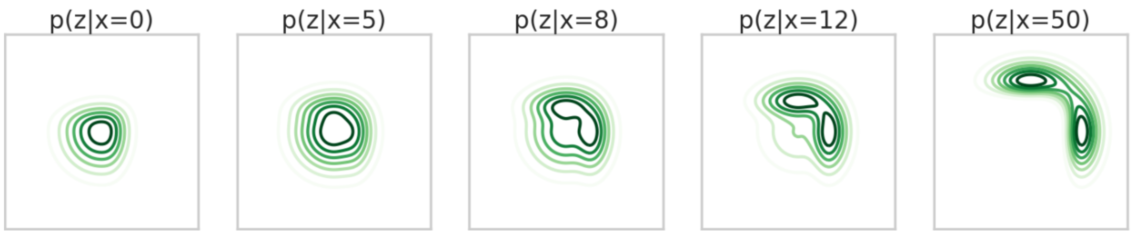


Figure 5.1: This figure depicts the true (unnormalised) posterior plots for the "Continuous Sprinkler" experiment. As $x$ increases, the posteriors become increasingly multimodal and unusually shaped. Clearly a flexible model for the posterior distribution is required, as a typical multivariate Gaussian model would fail to capture these features.

## 5.2 Program Structure

Recall from Section 3.9.1 that our generator $\mathscr{G}_\phi(x, \epsilon)$ is a neural network that takes in noise $\epsilon \sim \pi(\epsilon)$ along with data sample $x \sim q^*(x)$ to output a sample from the variational posterior distribution $z \sim q_\phi(z|x)$. In this experiment, the generator has 3 noise inputs $\boldsymbol{\epsilon} = (\epsilon_1, \epsilon_2, \epsilon_3)^\top \sim \mathcal{N}(0, I_{3\times 3})$ along with 1 data input $x$, and 2 posterior outputs $z_1, z_2$. The structure of the neural network is as depicted in Figure 5.2 below:

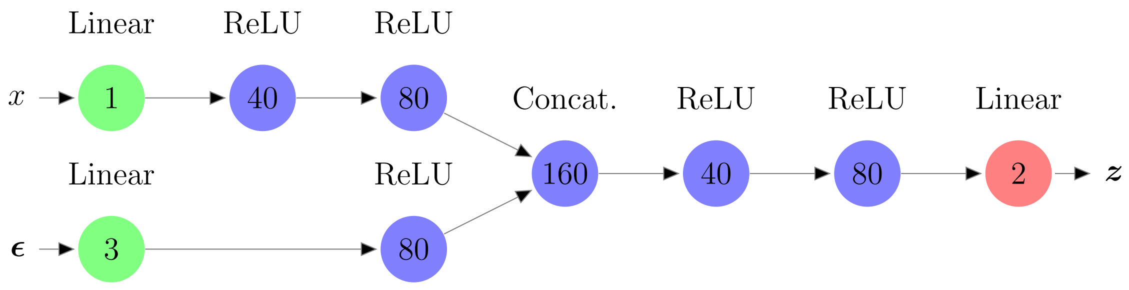


Figure 5.2: This figure illustrates the structure of the generator network $\mathscr{G}(x, \boldsymbol{\epsilon})$ used in the "Continuous Sprinkler" experiment. The number inside the node indicates how many nodes the layer has, and the text above the node describes the activation function. Recall that the input layer does not have an activation function, and Rectified Linear Units (ReLU) are used for most of the hidden layers due to their many advantages. In the Concat. layer, the two input vectors are concatenated and there is no activation function. We do not use an activation function for the output layer as $q_\phi(z|x) \in \mathbb{R}$.

The structure of the estimator (discriminator $D_\alpha(z, x)$ or ratio estimator $r_\alpha(z, x)$) is similar to that of the the generator, with an additional hidden layer associated with the $z$ input and a different activation function for the output layer, corresponding to the estimator's identity. Note that the only differences between the two estimators are the activation function of the their output layers and the loss functions being minimised. The estimator structure is shown in Figure 5.3 below:

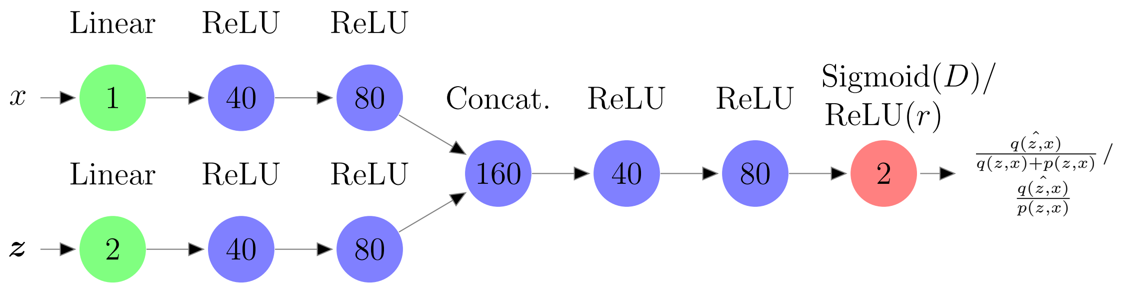


Figure 5.3: This figure depicts the structure of the estimator network for the "Continuous Sprinkler Experiment". The notation is the same as in Figure 5.2. When the estimator takes the parametrisation of a discriminator $D_\alpha(z, x) \simeq \frac{q(z,x)}{q(z,x)+p(z,x)}$, a sigmoid activation function is used for the output layer [Goodfellow et al., 2014]. A ReLU output layer is used when the estimator output is the direct density ratio $r_\alpha(z, x) \simeq \frac{q(z,x)}{p(z,x)}$ [Liu et al., 2017; Nam and Sugiyama, 2015].

The network weights are initialized with Xavier initialization, and trained using the Adam optimizer with learning rate 0.0001 for the prior-contrastive setting and 0.00001 for joint-contrastive. Preliminary tests show that these are the highest learning rates at which both families of algorithms converge consistently; any higher learning rates lead to the direct ratio estimator in divergence minimisation consistently entering a "failure" state. This scenario is discussed in section 5.3.

The network is only trained on data values $x = 0, 5, 8, 12, 50$. In each training iteration, 200 samples are taken for each x-value, corresponding to a batch size of 1000. This large batch size makes training more consistent, as the samples are probabilistic in nature. Due to the similar network structure in both algorithms, they have near-identical runtime, as the majority of the training time is spent in back-propagation [Goodfellow et al., 2016].

To prevent problems with taking $\log 0$ in the loss functions, we add a small constant $c = 10^{-18}$ to the log function's input. The estimator is pre-trained for 5000 iterations to ensure that optimization of the variational network begins with an optimal estimator; an inaccurate ratio estimation leads to incorrect optimization of the posterior network. The variance of the results is also reduced as the initial parametrisation of the estimator will be approximately the same for a fixed generator initialization. Afterwards, the generator is optimized for 10000 iterations when we have an implicit prior, and 40000 iterations in the joint-contrastive formulation. The algorithm alternates between 100 training steps of the estimator and 1 training step of the generator. The processes for class probability estimation and divergence minimisation, based off the algorithms derived in chapter 4, are shown in Algorithms 8 and 9 on the following pages.

We evaluate the techniques by comparing the average KL divergence between the true and variational posteriors at the end of the program's runtime:

$$\mathbb{E}_{q^*(x)} KL(q_\phi(\boldsymbol{z}|x) \| p(\boldsymbol{z}|x)).$$

To do so, we estimate the probability density function of the variational posterior output for each of the 5 data points with a Gaussian kernel density estimator $\hat{q}(\boldsymbol{z}|x)$. We also use the unnormalised true posterior as derived in equation (5.1.1). The normalisation constant $p(x)$ is the same regardless of the variational posterior $q_\phi(\boldsymbol{z}|x)$ that we compare it with, so its omission has no impact on our comparison of the algorithms. This density estimation technique is suitable for our low dimensional problem, and the accuracy is independent of the algorithm used. An explanation of kernel density estimation can be found in Appendix A. Each algorithm was run 30 times and at the end of each posterior optimisation step, the estimator loss and NELBO estimation was stored. Every 100 iterations, kernel density estimation was used to estimate the true average KL divergence.

```
Data: Dataset $q^*(x) = \{0, 5, 8, 12, 20\}$,
(Implicit) Prior $p(z) \sim \mathcal{N}(0, 2I_{2\times2})$,
Likelihood $p(x|z) \sim EXP(3 + \max(0, z_1)^3 + \max(0, z_2)^3)$,
Noise distribution $\pi(\epsilon) \sim \mathcal{N}(0, I_{3\times3})$
Result: Optimized posterior generator $\mathcal{G}_\phi(\epsilon; x)$
for $j = 1$ to 5000 do
    Sample $\{\epsilon^{(i,j)}\}_{i=1}^{1000} \sim \pi(\epsilon)$;
    Sample $\{0, 5, 8, 12, 50\}_{i=1}^{200} = \{x^{(i,j)}\}_{i=1}^{1000} \sim q^*(x)$;
    Sample $\{z_p^{(i,j)}\}_{i=1}^{1000} \sim p(z)$;
    foreach $\epsilon^{(i,j)}, x^{(i,j)}$ do
        Sample $z_q^{(i,j)} = \mathcal{G}(\epsilon^{(i,j)}; x_q^{(i,j})$;
    end
    update Estimator weights $\alpha$
end
for $j = 1$ to 10000 do
    for $k = 1$ to 100 do
        Sample $\{\epsilon^{(i,k)}\}_{i=1}^{1000} \sim \pi(\epsilon)$;
        Sample $\{0, 5, 8, 12, 50\}_{i=1}^{200} = \{x^{(i,k)}\}_{i=1}^{1000} \sim q^*(x)$;
        Sample $\{z_p^{(i,k)}\}_{i=1}^{1000} \sim p(z)$;
        foreach $\epsilon^{(i,k)}, x^{(i,k)}$ do
            Sample $z_q^{(i,k)} = \mathcal{G}(\epsilon^{(i,k)}; x_q^{(i,k)})$;
        end
        update Estimator weights $\alpha$
    end
    Sample $\{\epsilon^{(i)}\}_{i=1}^{1000} \sim \pi(\epsilon)$;
    Sample $\{0, 5, 8, 12, 50\}_{i=1}^{200} = \{x^{(i)}\}_{i=1}^{1000} \sim q^*(x)$;
    update Variational posterior weights $\phi$
end
```

**Algorithm 8:** Sprinkler Prior-Contrastive Algorithm

```
Data: Dataset $q^*(x) = \{0, 5, 8, 12, 20\}$,
(Implicit) Prior $p(z) \sim \mathscr{N}(0, 2I_{2\times 2})$,
(Implicit) Likelihood $p(x|z) \sim EXP(3 + \max(0, z_1)^3 + \max(0, z_2)^3)$,
Noise distribution $\pi(\epsilon) \sim \mathscr{N}(0, I_{3\times 3})$
Result: Optimized posterior generator $\mathscr{G}_\phi(\epsilon; x)$
for $j = 1$ to 5000 do
    Sample $\{\epsilon^{(i,j)}\}_{i=1}^{1000} \sim \pi(\epsilon)$;
    Sample $\{0, 5, 8, 12, 50\}_{i=1}^{200} = \{x_q^{(i,j)}\}_{i=1}^{1000} \sim q^*(x)$;
    Sample $\{z_p^{(i,j)}\}_{i=1}^{1000} \sim p(z)$;
    foreach $\epsilon^{(i,j)}, x^{(i,j)}$ do
        Sample $z_q^{(i,j)} = \mathscr{G}(\epsilon^{(i,j)}; x_q^{(i,j)})$;
    end
    foreach $z_p^{(i,j)}$ do
        Sample $x_p^{(i,j)} \sim p(x|z)$;
    end
    update Estimator weights $\alpha$
end
for $j = 1$ to 40000 do
    for $k = 1$ to 100 do
        Sample $\{\epsilon^{(i,k)}\}_{i=1}^{1000} \sim \pi(\epsilon)$;
        Sample $\{0, 5, 8, 12, 50\}_{i=1}^{200} = \{x_q^{(i,k)}\}_{i=1}^{1000} \sim q^*(x)$;
        Sample $\{z_p^{(i,k)}\}_{i=1}^{1000} \sim p(z)$;
        foreach $\epsilon^{(i,k)}, x^{(i,k)}$ do
            Sample $z_q^{(i,k)} = \mathscr{G}(\epsilon^{(i,k)}; x_q^{(i,k)})$;
        end
        foreach $z_p^{(i,k)}$ do
            Sample $x_p^{(i,k)} \sim p(x|z)$;
        end
        update Estimator weights $\alpha$
    end
    Sample $\{\epsilon^{(i)}\}_{i=1}^{1000} \sim \pi(\epsilon)$;
    Sample $\{0, 5, 8, 12, 50\}_{i=1}^{200} = \{x_q^{(i)}\}_{i=1}^{1000} \sim q^*(x)$;
    update Variational posterior weights $\phi$
end
```

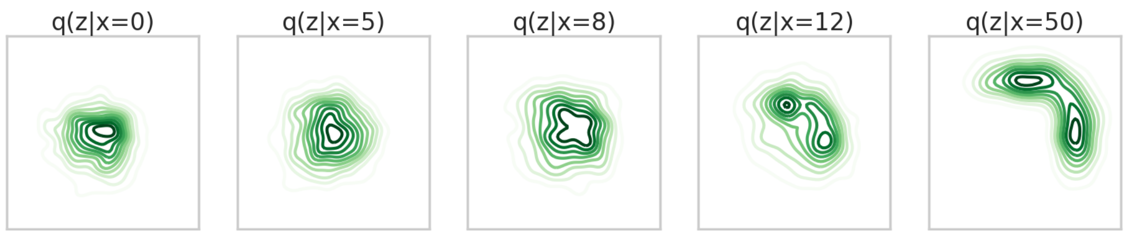
**Algorithm 9:** Sprinkler Joint-Contrastive Algorithm

## 5.3 Results

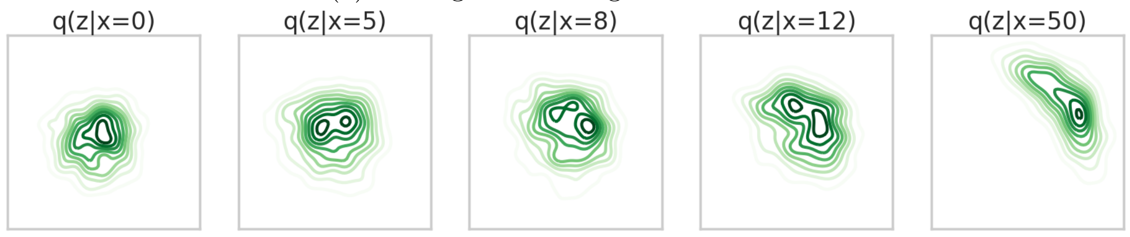
The mean and standard deviation of the estimated average KL divergence at the end of the program runtime is tabulated in Table 5.1 below. To visually compare the posterior outputs, Gaussian kernel density estimation was used to plot 1000 variational posterior samples for each data point. This is depicted in Figure 5.4 below. The arrays storing the average KL divergences, estimator losses and NELBOs over the iterations were averaged over the 30 repetitions. Plots of these arrays are shown in Figures 5.5-5.7.

| Algorithm | Mean KL Divergence | Standard Deviation |
|---|---|---|
| PC Divergence Minimisation | 1.3807 | 0.0391 |
| PC Class Probability Estimation | 1.3267 | 0.0041 |
| JC Divergence Minimisation | 1.6954 | 0.4337 |
| JC Class Probability Estimation | 1.3688 | 0.0415 |

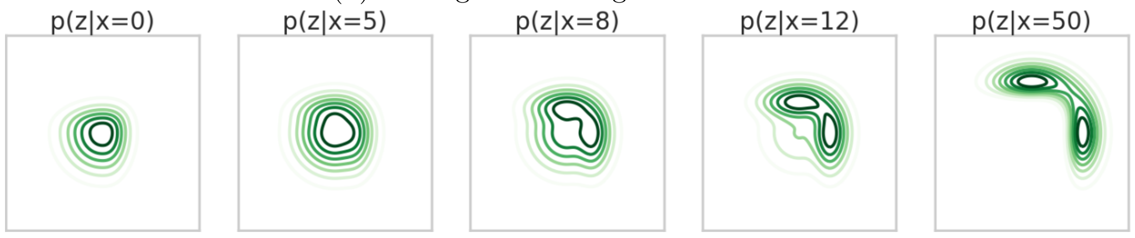
Table 5.1



(a) Average KL Divergence of 1.3288

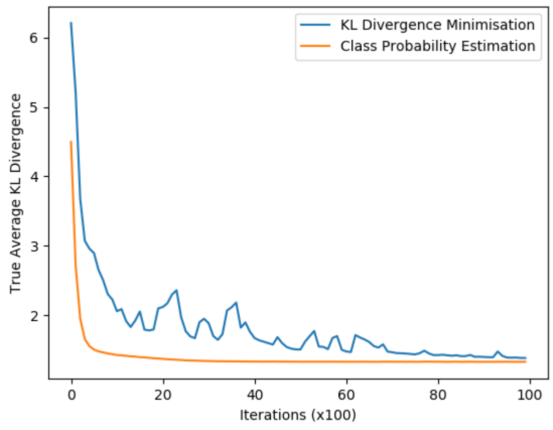


(b) Average KL Divergence of 1.3963

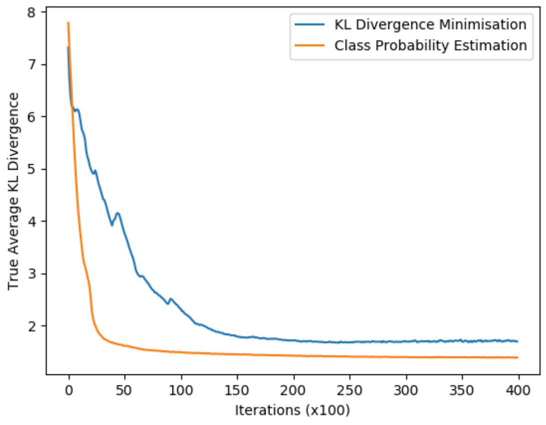


(c) True Posterior Plot

Figure 5.4: Initial tests show that the variational posterior reaches optimality at an average KL divergence of about 1.325: as seen in sub-figure (a), this corresponds to outputs similar to the true posterior in sub-figure (c). The posterior output in sub-figure (b) with a larger average KL divergence of 1.3963 is less flexible and only uni-modal for $x = 50$.

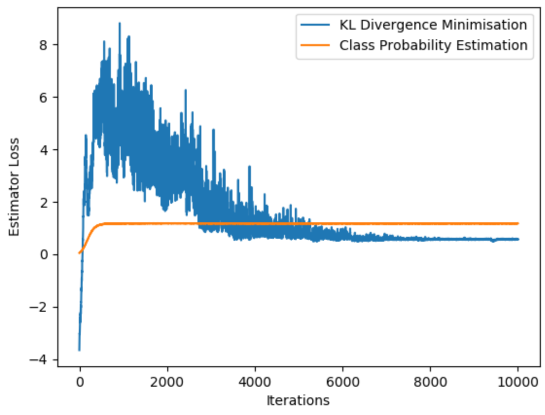


(a) Prior-Contrastive

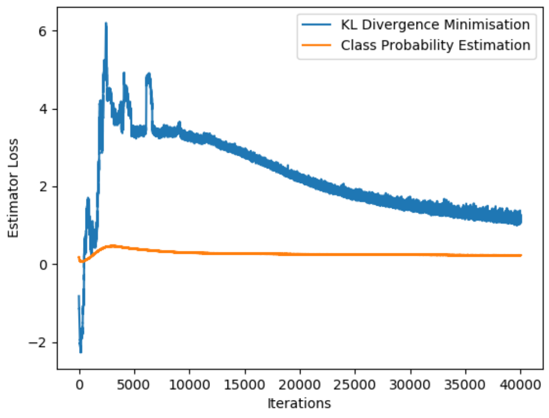


(b) Joint-Contrastive

Figure 5.5: Average KL Divergences (KDE Estimate)

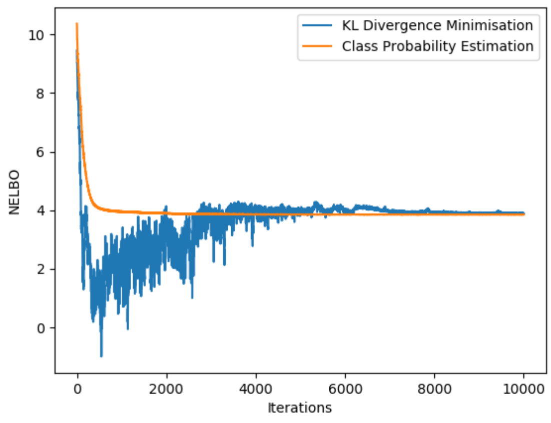


(a) Prior-Contrastive

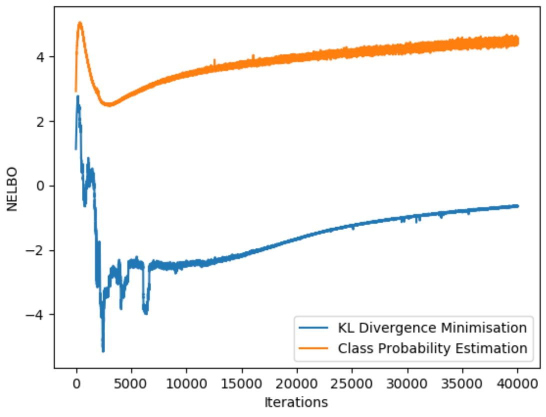


(b) Joint-Contrastive

Figure 5.6: Estimator Losses



(a) Prior-Contrastive



(b) Joint-Contrastive

Figure 5.7: NELBOs

As one would expect, the prior-contrastive algorithms converged much faster their joint-contrastive equivalent, as expected since knowledge of the likelihood improves posterior optimization. It is unclear in the joint-contrastive case why the NELBO appears to increase throughout the majority of the iterations. This is inconsistent with the reduction of the KDE estimated KL divergence over the same period, but it justifies our use of a metric independent of the estimator loss or estimated NELBO.

Overall, for both families of algorithms, we found that class probability estimation performed consistently better than KL divergence minimisation. The inconsistency of KL minimisation can be seen in the wild fluctuations early in the estimator loss and NELBO graphs, implying that the NELBO is being estimated inaccurately. This corresponds to the instances in the true KL Divergence graphs where the posterior output worsens over certain periods.

Furthermore, in approximately half of the simulations, the ratio loss of the KL divergence minimisation algorithm would start at a relatively high number (42.3 as opposed to 3.5), and it would be stuck there for the entire runtime of the program; the thousands of iterations spent minimizing the ratio loss did not decrease it. The algorithm was programmed to automatically restart every time this happened.

# Chapter 6

# Activation Function Experiment

In this chapter, we fix the 'failures' of KL minimisation by suggesting an exponential activation function to replace the ReLU output layer. The two methods are compared in both prior-contrastive and joint-contrastive contexts.

## 6.1 Theory

The 'failures' of the KL minimisation approach involve the estimator loss initialising at 41.4465 or reaching that value during its runtime, and remaining constant throughout the program runtime. Analysis of the estimator during this occurrence revealed that it was outputting an estimated ratio of 0. Now recall our estimator loss of

$$-\mathbb{E}_q[\log r_\alpha(z,x)] + \mathbb{E}_p[r_\alpha(z,x)]$$

and the constant term $c = 10^{-18}$ that we add to the log input. When $r_\alpha(z,x) = 0$, the estimator loss is $-\log[10^{-18}] = 41.4465$, which verifies that this is the cause of the 'failures'. It can be deduced that the network weights are transforming the estimator input into a negative number, which is mapped to 0 by the ReLU output layer. Since the gradient of the ReLU function is 0 for negative inputs, back-propagation will record that the partial derivative of the loss function with respect to the weights is 0, and therefore gradient descent will fail to update the weights. This is known as the 'dying ReLU' problem [Glorot et al., 2011], and in other contexts, it can be fixed using a 'leaky ReLU', which has a small slope for negative inputs[Maas, 2013]:

$$g(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$$

where $\alpha$ is a small number in $(0,1)$ such as 0.01. However, this allows a negative output, which causes problems with the log function in our estimator loss.

Even when the 'dying ReLU' problem does not occur, the estimator loss is still unstable for the majority of the program runtime, leading to an inaccurate NELBO estimation and therefore slower convergence. This is likely caused by the linearity of the ReLU output layer causing an imbalance between density ratios in $(0, 1)$ and $(1, \infty)$. Recall that the neural network output ranges in $\mathbb{R}$ before being mapped by the output layer's activation function. The density ratio $\frac{q(u)}{p(u)}$ ranges in $(0, 1)$ when $q(u) < p(u)$, and it takes values in $(1, \infty)$ when $q(u) > p(u)$, so to balance the two cases, it is ideal if they are each mapped to by complementary half-spaces, e.g. $\mathbb{R}^-$ maps to $(0, 1)$ and $\mathbb{R}^+$ maps to $(1, \infty)$. When the ReLU function is used, there is an infinitely larger space used to represent ratios where $q(u) > p(u)$, and the displacement in network weights required to reach optimality is this case is much greater. For example, if the current estimated ratio is $r(u) = 2$ (implying $q(u) = 2p(u)$) but we actually have $q(u) = 10p(u)$, then the weights would have to change such that the output increases by 8. On the other hand, if the estimated ratio is $r(u) = \frac{1}{2}$ (implying $2q(u) = p(u)$), but the true ratio is such that $10q(u) = p(u)$, then the weights only have to displace the output by $\frac{2}{5}$. The gradient-based optimisation would therefore require small training rates for cases where $r(u) \in (0, 1)$ and large training rates when $r(u) \in (1, \infty)$, but this scenario is not accounted for in optimisation algorithms using an adaptive training rate. This leads to the unstable estimator losses seen in Figure 5.6.

The sigmoid output of the class probability estimation approach does not experience this problem. Since $D^*(u) = \frac{q(u)}{q(u)+p(u)}$, we have $D(u) \in (0, \frac{1}{2})$ when $q(u) < p(u)$ and $D(u) \in (\frac{1}{2}, 1)$ when $q(u) > p(u)$. The sigmoid activation function $g(x) = \frac{1}{1+e^{-x}}$ maps $\mathbb{R}^-$ to $(0, \frac{1}{2})$ and $\mathbb{R}^+$ to $(\frac{1}{2}, 1)$, and therefore the two cases have half-space representations.

Following "f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization" by Nowozin et al. [2016], we propose an exponential output activation function $g(x) = e^x$ for the divergence minimisation approach. This activation function maps $\mathbb{R}^-$ to $(0, 1)$ and $\mathbb{R}^+$ to $(1, \infty)$, resulting in an even representation for the two cases. Following the same example as before, changing from an estimated ratio of 2 to 10 equates to a neural network output displacement of $|\log 10 - \log 2| = \log 5$, which is the network change as changing from $\frac{1}{2}$ to $\frac{1}{10}$: $|\log \frac{1}{2} - \log \frac{1}{10}| = \log 5$. The first contribution of this thesis will be to compare the ReLU and exponential activation functions for the estimator network used in divergence minimisation density ratio estimation, and to show experimentally that the exponential activation function is more stable and leads to faster posterior convergence.

## 6.2 Experiment Outline

In this experiment, we compare the activation functions in both prior-contrastive and joint-contrastive contexts, using the exact same experimental set up as described in Chapter 5. The divergence minimisation program is exactly the same, but we now have an additional program that uses an exponential activation function instead of ReLU for the estimator's output layer. To save computational time, we reuse our ReLU results from the previous experiment, only running instances of our new program. Again, we estimate the variational distribution with a Gaussian kernel density estimator to calculate the KL divergence, which we use to evaluate to convergence.

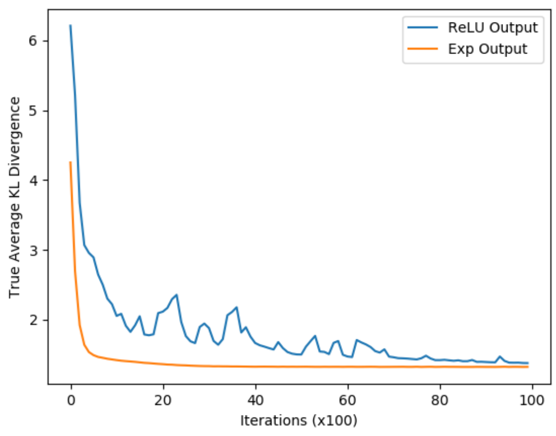
## 6.3 Results

The mean and standard deviation of the final average KL divergence between the true and variational distributions over 30 experiment repetitions is tabulated in Table 6.1 below. It is evident that the exponential output activation function results in vastly superior posterior convergence, with significantly lower KL divergence mean and standard deviation than the algorithm using a ReLU output. Comparing Table 6.1 with Table 5.1, it can be seen that divergence minimisation with an exponential output produces similar results to class probability estimation. It can be deduced that the perceived inferiority of divergence minimisation is a result of using an inappropriate activation function for the output layer.

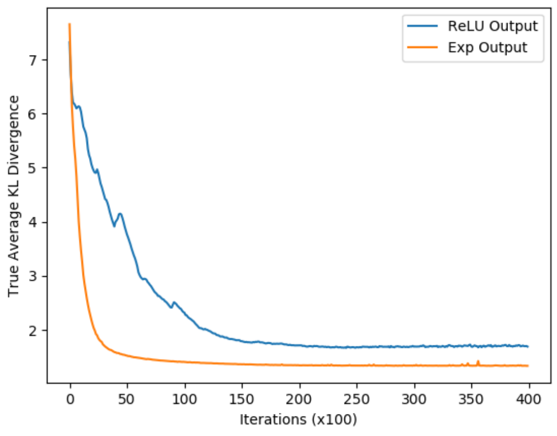
We have also taken the average of the arrays storing the average KL divergence, estimator loss and estimated NELBO for each posterior iteration. Plots of these arrays are shown in Figures 6.1-6.3 on the next page.

| Algorithm | Mean KL Divergence | Standard Deviation |
|---|---|---|
| PC ReLU Output | 1.3807 | 0.0391 |
| PC Exponential Output | 1.3265 | 0.0045 |
| JC ReLU Output | 1.6954 | 0.4337 |
| JC Exponential Output | 1.3397 | 0.0066 |

Table 6.1

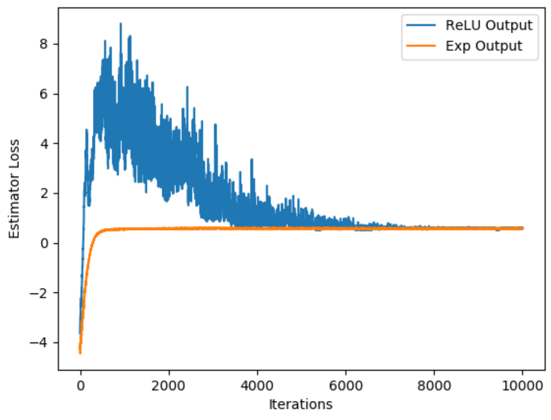


(a) Prior-Contrastive

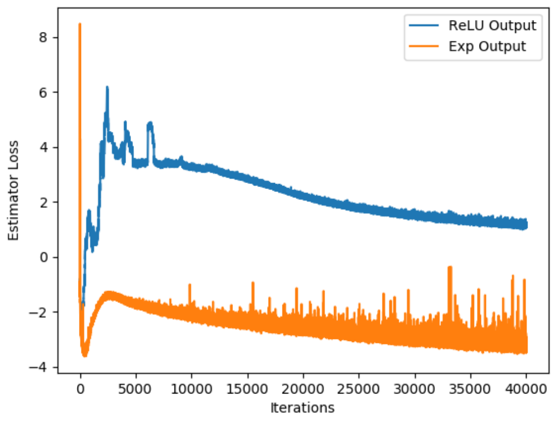


(b) Joint-Contrastive

Figure 6.1: Average KL Divergences (KDE Estimate)

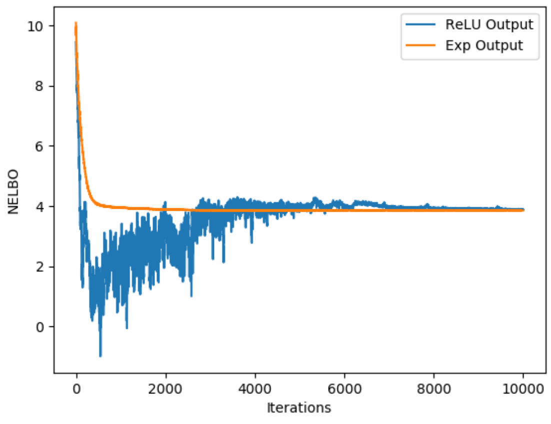


(a) Prior-Contrastive

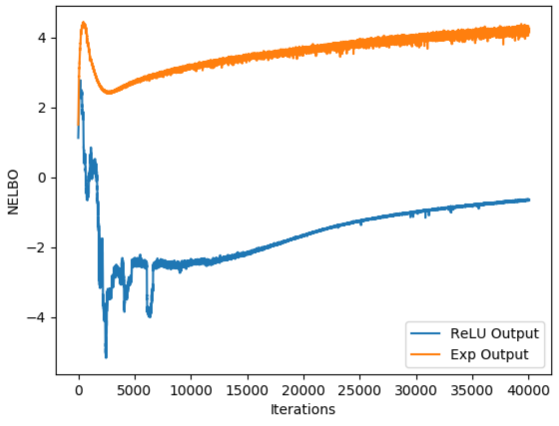


(b) Joint-Contrastive

Figure 6.2: Estimator Losses



(a) Prior-Contrastive



(b) Joint-Contrastive

Figure 6.3: NELBOs

The average KL divergence for the exponential output appears to converge much more smoothly, and it is consistently lower than in the ReLU output. For the prior-contrastive case, the estimator loss and estimated NELBO from the program using an exponential activation function is much more stable. However, in the joint-contrastive case, these values show more stability early on, then grow increasingly unstable compared to the ReLU case. It is unclear why this occurs, but it does not appear to have a detrimental impact on posterior convergence, as the average KL divergence is still significantly lower during this period.

# Chapter 7
# Algorithm Analysis

In this chapter, we present the two density ratio estimation techniques in a comparable form, that is, in terms of the relevant f-divergence's lower bound as derived in section 4.2 using Theorem 4.2.1. From this, we generalise the density ratio algorithms to a selection of f-divergence to formulate the lower bound, and a parametrisation of the density ratio estimator. We also suggest a third estimator parametrisation: the direct log density ratio estimator $T_\alpha(u)$. The formal generalisation of density ratio estimation algorithms is the second contribution of this thesis.

## 7.1 Introduction

First, recall that within each of the prior-contrastive and joint-contrastive formulations, both class probability estimation and divergence minimisation methods use the same loss function to optimise the posterior weights. It is therefore evident that the only differences between the two density ratio estimation techniques are the lower bound of the loss function and the parametrization of the estimator. We demonstrate this by restating the f-divergence lower bounds used to formulate these loss functions, as derived in Section 4.2. For divergence minimisation, the lower bound is achieved using the reverse KL divergence $D_{RKL}(p\|q) = KL[q(u)\|p(u)]$:

$$D_{RKL}(p\|q) \geq \sup_{\alpha}\{\mathbb{E}_{q(u)}[1 + \log r_\alpha(u)] - \mathbb{E}_{p(u)}[r_\alpha(u)]\},$$

and class probability estimation follows the GAN divergence $D_{GAN}(p\|q) = 2D_{JS}(p\|q) - \log 4$:

$$D_{GAN}(p\|q) \geq \sup_{\alpha}\{\mathbb{E}_{q(u)}\left[\log \frac{r_\alpha(u)}{r_\alpha(u)+1}\right] - \mathbb{E}_{p(u)}[\log(r_\alpha(u)+1)]\},$$

with equality at $r_\alpha(u) = \frac{q(u)}{p(u)}$.
To convert the latter equation to class probability estimation loss function, we have used the estimator transformation $r_\alpha(u) = \frac{D_\alpha(u)}{1-D_\alpha(u)} \iff D_\alpha(u) = \frac{r_\alpha(u)}{r_\alpha(u)+1}$. Since

this mapping is bijective and monotonically increasing, the equality of the bound is retained at equivalent points. We can therefore also use this transformation on the lower bound of the reverse KL divergence, deriving its respective loss function as a function of $D_\alpha \simeq \frac{q(u)}{q(u)+p(u)}$ instead of $r_\alpha \simeq \frac{q(u)}{p(u)}$.

We also propose a third estimator parametrisation, the direct log density ratio estimator

$$T_\alpha(u) \simeq \log \frac{q(u)}{p(u)}.$$

This is derived by simply using the transformation $T_\alpha(u) = \log r_\alpha(u)$.

## 7.2 Algorithm Generalisation

For each f-divergence, we can derive a loss function for each of our three estimator parametrisations.

### *7.2.1 Reverse KL Divergence*

**Direct Ratio Estimator** $r_\alpha(u)$:

$$\min_\alpha -\mathbb{E}_{q(u)}[\log r_\alpha(u)] + \mathbb{E}_{p(u)}[r_\alpha(u)]$$

**Class Probability Estimator/Discriminator** $D_\alpha(u)$:

$$\min_\alpha \mathbb{E}_{q(u)}\left[\log \frac{1-D_\alpha(u)}{D_\alpha(u)}\right] + \mathbb{E}_{p(u)}\left[\frac{D_\alpha(u)}{1-D_\alpha(u)}\right]$$

**Direct Log Ratio Estimator** $T_\alpha(u)$:

$$\min_\alpha -\mathbb{E}_{q(u)}[T_\alpha(u)] + \mathbb{E}_{p(u)}[e^{T_\alpha(u)}]$$

### *7.2.2 GAN Divergence*

**Direct Ratio Estimator** $r_\alpha(u)$:

$$\min_\alpha \mathbb{E}_{q(u)}\left[\log \frac{r_\alpha(u)+1}{r_\alpha(u)}\right] + \mathbb{E}_{p(u)}[\log(r_\alpha(u)+1)]$$

**Class Probability Estimator/Discriminator** $D_\alpha(u)$:

$$\min_\alpha -\mathbb{E}_{q(u)}[\log D_\alpha(u)] - \mathbb{E}_{p(u)}[\log(1-D_\alpha(u))]$$

**Direct Log Ratio Estimator** $T_\alpha(u)$:

$$\min_\alpha \mathbb{E}_{q(u)}\left[\log \frac{e^{T_\alpha(u)}+1}{e^{T_\alpha(u)}}\right] + \mathbb{E}_{p(u)}[\log(e^{T_\alpha(u)}+1)]$$

We have therefore generalised the choice of algorithm for training density ratio estimators to a choice of f-divergence:

- KL Divergence: $D_{KL}(p\|q) = \mathbb{E}_{q(u)}[1 + \log \frac{q(u)}{p(u)}] - \mathbb{E}_{p(u)}\left[\frac{q(u)}{p(u)}\right]$
- GAN Divergence: $D_{GAN}(p\|q) = \mathbb{E}_{q(u)}\left[\log \frac{q(u)}{q(u)+p(u)}\right] + \mathbb{E}_{p(u)}\left[\log \frac{p(u)}{q(u)+p(u)}\right]$

and a choice of estimator parametrization:

- Direct ratio estimator: $r_\alpha(u) \simeq \frac{q(u)}{p(u)}$
- "Class probability" estimator/Discriminator: $D_\alpha(u) \simeq \frac{q(u)}{q(u)+p(u)}$
- Direct log ratio estimator: $T_\alpha(u) \simeq \log \frac{q(u)}{p(u)}$.

The original "KL divergence minimization approach" simply chooses the KL Divergence and the direct ratio estimator, and "class probability estimation" uses the CPE Divergence and the "Class probability" estimator. These are just 2 variations of the 6 available algorithms.

## 7.3 Optimization Algorithms

The ratio estimator in the posterior loss functions also has to be transformed accordingly. In the following sections we give the specific prior-contrastive and joint-contrastive NELBOs as functions of each estimator parametrisation. For completeness, we also list the two different estimator loss functions for each case, corresponding to the two f-divergences that we have discussed.

### *7.3.1 Prior-Contrastive*

**Direct Ratio Estimator** $r_\alpha(u)$:

$$\text{Reverse KL: } \min_\alpha -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log r_\alpha(\mathscr{G}(\epsilon;x),x)] + \mathbb{E}_{p(z)q^*(x)}[r_\alpha(z,x)]$$

$$\text{GAN: } \min_\alpha \mathbb{E}_{q^*(x)\pi(\epsilon)}\left[\log \frac{r_\alpha(\mathscr{G}(\epsilon;x),x)+1}{r_\alpha(\mathscr{G}(\epsilon;x),x)}\right] + \mathbb{E}_{p(z)q^*(x)}[\log(r_\alpha(z,x)+1)]$$

$$\min_\phi -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log p(x|\mathscr{G}_\phi(\epsilon;x)] + \mathbb{E}_{q^*(x)\pi(\epsilon)}[\log r_\alpha(\mathscr{G}(\epsilon;x),x)]$$

**Class Probability Estimator/Discriminator** $D_\alpha(u)$:

$$\text{Reverse KL: } \min_\alpha \mathbb{E}_{q^*(x)\pi(\epsilon)}\left[\log \frac{1-D_\alpha(\mathscr{G}(\epsilon;x),x)}{D_\alpha(\mathscr{G}(\epsilon;x),x)}\right] + \mathbb{E}_{p(z)q^*(x)}\left[\frac{D_\alpha(\mathscr{G}(\epsilon;x),x)}{1-D_\alpha(\mathscr{G}(\epsilon;x),x)}\right]$$

$$\text{GAN: } \min_{\alpha} \mathbb{E}_{q^*(x)\pi(\epsilon)}[\log D_\alpha(\mathscr{G}(\epsilon; x), x)] + \mathbb{E}_{p(z)q^*(x)}[\log(1 - D_\alpha(z, x))]$$

$$\min_{\phi} -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log p(x|\mathscr{G}_\phi(\epsilon; x)] + \mathbb{E}_{q^*(x)\pi(\epsilon)}\left[\log \frac{D_\alpha(\mathscr{G}(\epsilon; x), x)}{1 - D_\alpha(\mathscr{G}(\epsilon; x), x)}\right]$$

**Direct Log Ratio Estimator $T_\alpha(u)$:**

$$\text{Reverse KL: } \min_{\alpha} -\mathbb{E}_{q^*(x)\pi(\epsilon)}[T_\alpha(\mathscr{G}(\epsilon; x), x)] + \mathbb{E}_{p(z)q^*(x)}[\exp(T_\alpha(z, x))]$$

$$\text{GAN: } \min_{\alpha} \mathbb{E}_{q^*(x)\pi(\epsilon)}\left[\log \frac{e^{T_\alpha(\mathscr{G}(\epsilon;x),x)} + 1}{e^{T_\alpha(\mathscr{G}(\epsilon;x),x)}}\right] + \mathbb{E}_{p(z)q^*(x)}[\log(e^{T_\alpha(z,x)} + 1)]$$

$$\min_{\phi} -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log p(x|\mathscr{G}_\phi(\epsilon; x)] + \mathbb{E}_{q^*(x)\pi(\epsilon)}[T_\alpha(\mathscr{G}(\epsilon; x), x)]$$

### *7.3.2 Joint-Contrastive*

**Direct Ratio Estimator $r_\alpha(u)$:**

$$\text{Reverse KL: } \min_{\alpha} -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log r_\alpha(\mathscr{G}(\epsilon; x), x)] + \mathbb{E}_{p(z)p(x|z)}[r_\alpha(z, x)]$$

$$\text{GAN: } \min_{\alpha} \mathbb{E}_{q^*(x)\pi(\epsilon)}\left[\log \frac{r_\alpha(\mathscr{G}(\epsilon; x), x) + 1}{r_\alpha(\mathscr{G}(\epsilon; x), x)}\right] + \mathbb{E}_{p(z)p(x|z)}[\log(r_\alpha(z, x) + 1)]$$

$$\min_{\phi} \mathbb{E}_{q^*(x)\pi(\epsilon)} \log r_\alpha(\mathscr{G}_\phi(\epsilon; x), x)$$

**Class Probability Estimator/Discriminator $D_\alpha(u)$:**

$$\text{Reverse KL: } \min_{\alpha} \mathbb{E}_{q^*(x)\pi(\epsilon)}\left[\log \frac{1 - D_\alpha(\mathscr{G}(\epsilon; x), x)}{D_\alpha(\mathscr{G}(\epsilon; x), x)}\right] + \mathbb{E}_{p(z)p(x|z)}\left[\frac{D_\alpha(\mathscr{G}(\epsilon; x), x)}{1 - D_\alpha(\mathscr{G}(\epsilon; x), x)}\right]$$

$$\text{GAN: } \min_{\alpha} \mathbb{E}_{q^*(x)\pi(\epsilon)}[\log D_\alpha(\mathscr{G}(\epsilon; x), x)] + \mathbb{E}_{p(z)p(x|z)}[\log(1 - D_\alpha(z, x))]$$

$$\min_{\phi} \mathbb{E}_{q^*(x)\pi(\epsilon)} \log \frac{D_\alpha(\mathscr{G}_\phi(\epsilon; x), x)}{1 - D_\alpha(\mathscr{G}_\phi(\epsilon; x), x)}$$

**Direct Log Ratio Estimator $T_\alpha(u)$:**

$$\text{Reverse KL: } \min_{\alpha} -\mathbb{E}_{q^*(x)\pi(\epsilon)}[T_\alpha(\mathscr{G}(\epsilon; x), x)] + \mathbb{E}_{p(z)p(x|z)}[\exp(T_\alpha(z, x))]$$

$$\text{GAN: } \min_{\alpha} \mathbb{E}_{q^*(x)\pi(\epsilon)}\left[\log \frac{e^{T_\alpha(\mathscr{G}(\epsilon;x),x)} + 1}{e^{T_\alpha(\mathscr{G}(\epsilon;x),x)}}\right] + \mathbb{E}_{p(z)p(x|z)}[\log(e^{T_\alpha(z,x)} + 1)]$$

$$\min_{\phi} \mathbb{E}_{q^*(x)\pi(\epsilon)} T_\alpha(\mathscr{G}_\phi(\epsilon; x), x)$$

# Chapter 8

# Comparing Optimal Estimators

In this chapter, we verify that for a fixed f-divergence, each estimator parametrisation leads to similar levels of convergence when optimally trained. In this experiment, we also determine whether the f-divergence used to derive the estimator loss function has any effect on posterior convergence.

## 8.1 Theory

Recall from Section 7.1 that the estimator transformation preserves the equality of the bound. There should be no significant difference in posterior convergence when the estimators are optimized sufficiently between each posterior iteration step, as they reach equality of the f-divergence lower bound, therefore optimally estimating the NELBO.

Theorem 4.2.1 states that equality of the bound is attained at $r_\alpha(u) = \frac{q(u)}{p(u)}$ regardless of the f-divergence used, so we also expect similar levels of convergence between the f-divergences. However, "f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization" by Nowozin et al. [2016] shows experimentally that the fastest posterior convergence is attained when the f-divergence used to optimize the variational posterior is also used to derive the lower bound. Since we train our posterior network by minimising its reverse KL divergence with the true posterior, this paper implies the estimator loss functions corresponding to the reverse KL divergence would correspond to superior posterior convergence.

## 8.2 Experiment Outline

Again, the experiment parameters are the same as in the previous two experiments. The low training rates assure smooth estimator and posterior convergence, and we continue to ensure that the estimator attains optimality before it is used to estimate the NELBO by initialising it with 5000 optimisation steps, then training it for 100 steps between each posterior iteration. To save computational time, we reuse our previous experimental results corresponding to the reverse KL divergence with the direct ratio estimator, and the GAN divergence with the class probability estimator. We do not use an output activation function for the direct log ratio estimator as the log of a positive number ranges in $\mathbb{R}$.

## 8.3 Results

Tables 8.1 and 8.2 below compare the final posterior convergence between the 30 iterations, using the same Gaussian kernel density estimation technique as in the previous 2 experiments. There is no significant difference in posterior convergence for the algorithms in the prior-contrastive context, and the variation in the results is very low. However, Figure 5.4 suggests that the variational posterior reaches optimality at the 'true' KL divergence of around 1.326. This was easily verified by running a prior-contrastive algorithm for twice as long (20000 posterior iterations) and observing that the KL divergence did not decrease any further. Since the programs reached optimality in the prior-contrastive case, it is uncertain from those results whether the choice of f-divergence or estimator parametrisation impacts posterior convergence. However, observing the joint-contrastive results, it is evident that for each f-divergence, the different estimator parametrisations demonstrate similar posterior convergence. As Nowozin's paper suggests, the GAN divergence demonstrates slower and more inconsistent results than the reverse KL divergence. This contradicts our intuition as explained in Section 8.1.

| Algorithm | Mean KL Divergence | Standard Deviation |
|---|---|---|
| PC Reverse KL - $D_\alpha(z,x)$ | 1.3271 | 0.0041 |
| PC Reverse KL - $r_\alpha(z,x)$ | 1.3265 | 0.0045 |
| PC Reverse KL - $T_\alpha(z,x)$ | 1.3262 | 0.0041 |
| PC GAN - $D_\alpha(z,x)$ | 1.3267 | 0.0041 |
| PC GAN - $r_\alpha(z,x)$ | 1.3263 | 0.0035 |
| PC GAN - $T_\alpha(z,x)$ | 1.3258 | 0.0039 |

Table 8.1: Prior-Contrastive Results

| Algorithm | Mean KL Divergence | Standard Deviation |
|---|---|---|
| JC Reverse KL - $D_\alpha(z,x)$ | 1.3416 | 0.0068 |
| JC Reverse KL - $r_\alpha(z,x)$ | 1.3397 | 0.0066 |
| JC Reverse KL - $T_\alpha(z,x)$ | 1.3446 | 0.0108 |
| JC GAN - $D_\alpha(z,x)$ | 1.3648 | 0.0242 |
| JC GAN - $r_\alpha(z,x)$ | 1.3657 | 0.0302 |
| JC GAN - $T_\alpha(z,x)$ | 1.3670 | 0.0387 |

Table 8.2: Joint-Contrastive Results

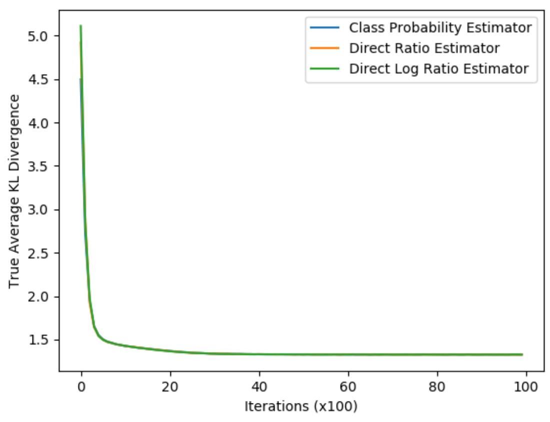
The following two pages hold figures corresponding to the average KL divergence, estimator loss and estimated NELBO over the runtime of the program, for both prior-contrastive and joint-contrastive cases.

Observe in the prior-contrastive case that all three estimators demonstrate equal, indistinguishable convergence, and that the plots are smooth with stable estimator loss and NELBO. This is also the case for the average KL divergence plot in the joint-contrastive case, with the exception of the class probability estimator with GAN divergence, but this is likely the result of an outlier experiment as their posterior convergences are similar.

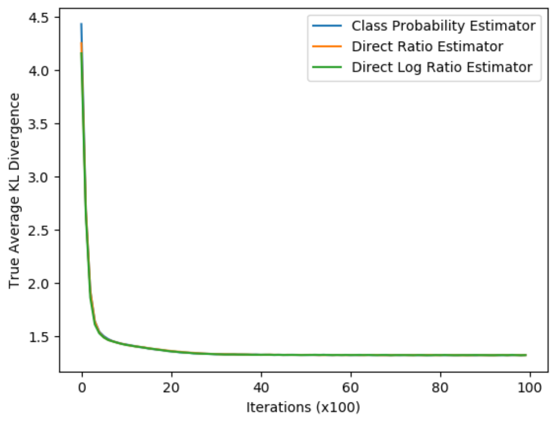
For the GAN divergence, the estimator loss plots show similar stability between the estimators, but the class probability estimator appears to have higher average loss. This is reflected in the corresponding NELBO plot, which has a lower NELBO estimation by the class probability estimator than by the other two estimator parametrisations.

The different estimators in the reverse KL divergence have unstable but identical estimator losses during the later part of the program runtime. This is again reflected in the NELBO plot.

The GAN and reverse KL divergence NELBO plots are comparable as they use identical scales on the y-axis, though the axes have different ranges. The GAN divergence is much more unstable during the later part of the program runtime, and appears to have a consistently greater NELBO estimation. This correlates with the improved posterior convergence associated with the reverse KL divergence.

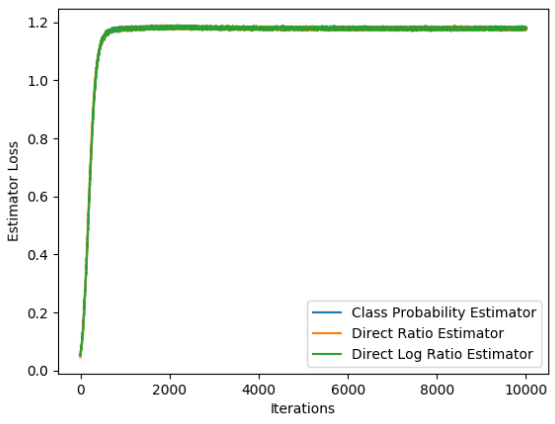


(a) GAN Divergence

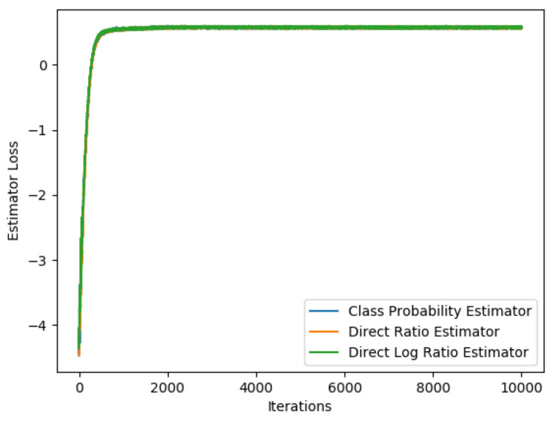


(b) Reverse KL Divergence

Figure 8.1: Prior-Contrastive Average KL Divergence

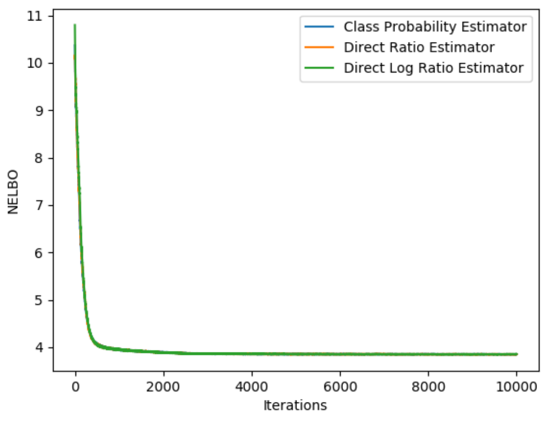


(a) GAN Divergence

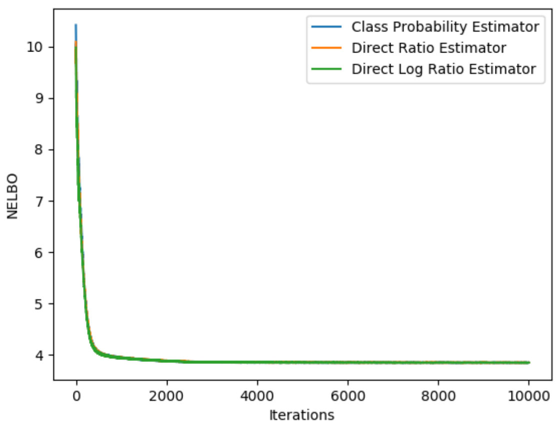


(b) Reverse KL Divergence

Figure 8.2: Prior-Contrastive Estimator Loss

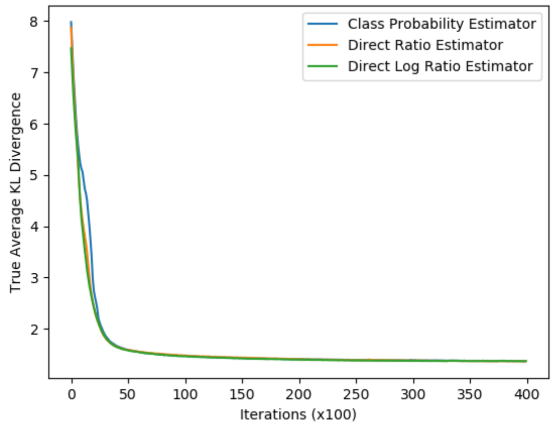


(a) GAN Divergence

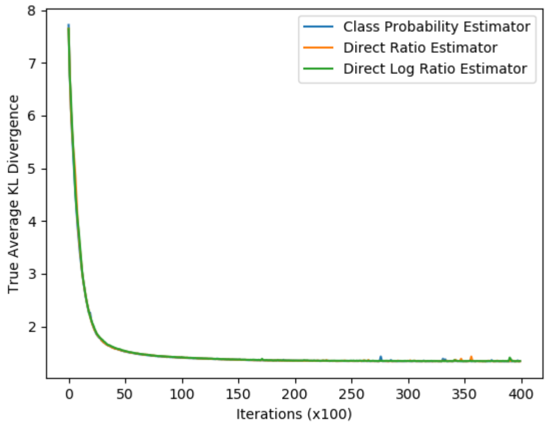


(b) Reverse KL Divergence

Figure 8.3: Prior-Contrastive NELBO

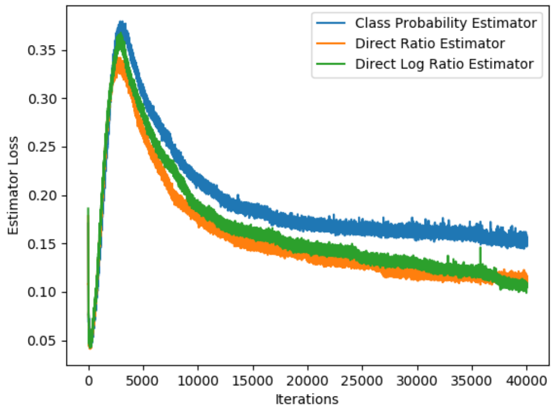


(a) GAN Divergence

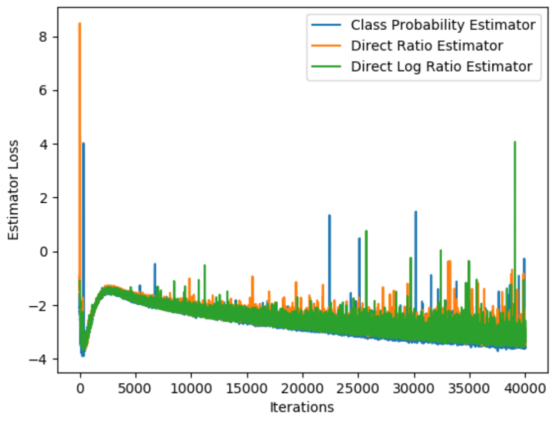


(b) Reverse KL Divergence

Figure 8.4: Joint-Contrastive Average KL Divergence

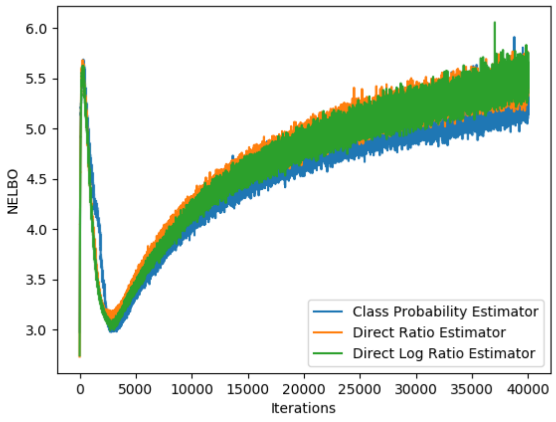


(a) GAN Divergence

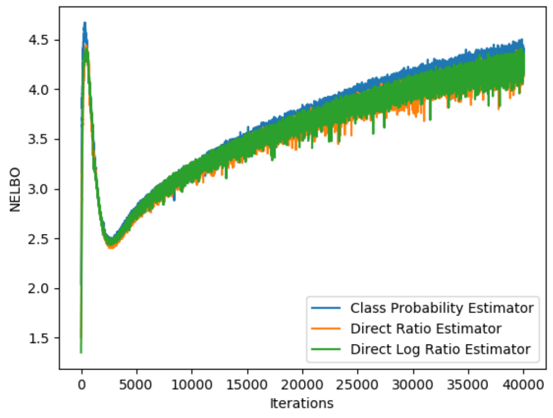


(b) Reverse KL Divergence

Figure 8.5: Joint-Contrastive Estimator Loss



(a) GAN Divergence



(b) Reverse KL Divergence

Figure 8.6: Joint-Contrastive NELBO

# Chapter 9

# Comparing Undertrained Estimators

Since all three estimator parametrisations have similar accuracies at optimality, it does not matter which one is chosen in that context. However, insufficient estimator training iterations can lead to an inaccurate estimation of the $NELBO$, reducing the speed at which the variational posterior converges. In this chapter, we first theoretically compare the rates of estimator convergence between the three parametrisations by analysing the bounds of the estimator outputs and the displacement in the optimal estimator output with each posterior optimisation step. We then perform an experiment in both prior-contrastive and joint-contrastive contexts comparing the performance of undertrained estimators with different parametrisations and f-divergence upper bounds.

## 9.1 Theory

### *9.1.1 Estimator Bounds*

Consider the bounds on the estimator outputs. The class probability estimator $D_\alpha(u) \simeq \frac{q(u)}{q(u)+p(u)}$ is bound in $(0, 1)$, so from any arbitrary starting point in the same space, very few optimization steps are required to reach the global minimum. On the other hand, the direct ratio estimator $r_\alpha(u) \simeq \frac{q(u)}{p(u)}$ is bound in $\mathbb{R}^+\backslash\{0\}$, and the direct log ratio estimator $T_\alpha(u) \simeq \log \frac{q(u)}{p(u)}$ can take any value in $\mathbb{R}$, so optimization can take many iterations if the difference between the estimator's initial and optimal values is too large. This can be problematic if there are insufficient iterations of optimizing the estimator, particularly in the initialization stage: the estimator may not properly converge and the posterior will be trained with an inaccurate density ratio estimation.

### *9.1.2 Displacement of Estimator Optimal Values*

Furthermore, we can also consider the effect of the posterior density displacement from each training step: every time $q_\phi(u)$ changes, the optimal value of the estimator also changes. Consequently, the estimator must take optimization steps to 'catch up', but again, if the displacement is too significant, then the estimator may not converge in time.

**Lemma 9.1.1.** *For a fixed displacement of the variational distribution $q(u)$, the class probability estimator's global minimum displaces less than the direct ratio estimator, that is, $|D^*_{n+1} - D^*_n| < |r^*_{n+1} - r^*_n|$:*

*Proof.* Letting $\epsilon \neq 0$ be the change in $q(u)$ with an optimization step, and noting that $|\epsilon| < q(u)$, we have

$$\begin{aligned}
|D^*_{n+1} - D^*_n| &= \left| \frac{q(u)+\epsilon}{q(u)+\epsilon+p(u)} - \frac{q(u)}{q(u)+p(u)} \right| \\
&= \left| \frac{q^2(u)+q(u)p(u)+\epsilon q(u)+\epsilon p(u)}{(q(u)+\epsilon+p(u))(q(u)+p(u))} - \frac{q^2(u)+\epsilon q(u)+q(u)p(u)}{(q(u)+\epsilon+p(u))(q(u)+p(u))} \right| \\
&= \left| \frac{\epsilon p(u)}{(q(u)+\epsilon+p(u))(q(u)+p(u))} \right| \\
&= \left| \frac{\epsilon}{(q(u)+\epsilon+p(u))\left(\frac{q(u)}{p(u)}+1\right)} \right| \\
|r^*_{n+1} - r^*_n| &= \left| \frac{q(u)+\epsilon}{p(u)} - \frac{q(u)}{p(u)} \right| \\
&= \left| \frac{\epsilon}{p(u)} \right|.
\end{aligned}$$

If $\epsilon > 0$, then

$$|D^*_{n+1} - D^*_n| < |r^*_{n+1} - r^*_n| \text{ as } (q(u)+\epsilon+p(u))\left(\frac{q(u)}{p(u)}+1\right) > p(u).$$

If $\epsilon < 0$, then recalling that $|\epsilon| < q(u) < q(u)+p(u)$,

$$\begin{aligned}
(q(u)+\epsilon+p(u))(\frac{q(u)}{p(u)}+1) &= \frac{q^2(u)}{p(u)} + 2q(u) + p(u) + \epsilon\left(\frac{q(u)}{p(u)}+1\right) \\
&= (q(u)+\epsilon)\left(\frac{q(u)}{p(u)}+1\right) + q(u) + p(u) \\
&> p(u)
\end{aligned}$$

□

**Remark 9.1.2.** *The class probability estimator $D_\alpha(u)$ has more accurate density ratio estimation than the direct ratio estimator $r_\alpha(u)$ when the estimators are undertrained.*

For the direct log ratio estimator, we have

$$\begin{aligned}|T^*_{n+1} - T^*_n| &= \left|\log \frac{q(u)+\epsilon}{p(u)} - \log \frac{q(u)}{p(u)}\right| \\ &= \left|\log \frac{q(u)+\epsilon}{q(u)}\right|\end{aligned}$$

It is difficult to make a direct comparison with the other displacement expressions.

## 9.2 Experiment Outline

In this experiment, we aim to confirm our theory that the estimator parametrisations have differing density ratio estimation accuracies when improperly trained, also determining which undertrained estimator is the most accurate by observing the convergence of the variational posterior.

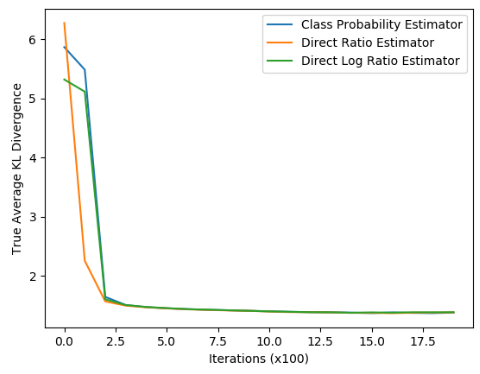
The same "Continuous Sprinkler" experimental setup is used, but with several changes to the training parameters. We significantly reduce the amount of estimator training, lowering the estimator training rate to 0.00004 and the estimator steps before each posterior iteration to 11 in the prior-contrastive setting, and 16 for the joint-contrastive algorithm. We also increase the posterior training rate to 0.0002 and to account for this change, the number of optimization steps of the variational distribution is reduced to 2000 for the prior-contrastive context and 4000 in the joint-contrastive algorithms.

## 9.3 Results

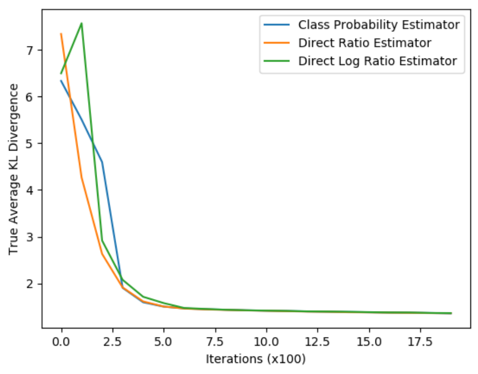
From the table, it can be seen that the class probability estimator is superior to the direct ratio estimator, which is better than the log ratio estimator. This is supportive of our theories detailing the estimator output bounds, displacements and second derivatives as factors affecting training speed. The theory of second derivatives, as well as Nowozin's paper is further supported by the superiority of the KL divergence. It is difficult to discern any related trends in the figures: most of the estimator and NELBO plots have wild fluctuations at the start of the training period, which settles down to a steady decline at about the same point for all three estimators. Most interesting is the joint-contrastive CPE divergence plots for estimator loss and NELBO, which demonstrates greater fluctuations of the direct ratio estimator, deviating from the other two estimators. Despite this, the estimator's effectiveness lies between the other two estimators. This trend is not experienced by any of the other contexts either.

| Algorithm | Mean KL Divergence | Standard Deviation |
|---|---|---|
| PC Reverse KL - $D_\alpha(z,x)$ | 1.3572 | 0.0136 |
| PC Reverse KL - $r_\alpha(z,x)$ | 1.3607 | 0.0199 |
| PC Reverse KL - $T_\alpha(z,x)$ | 1.3641 | 0.0141 |
| PC GAN - $D_\alpha(z,x)$ | 1.3788 | 0.0258 |
| PC GAN - $r_\alpha(z,x)$ | 1.3811 | 0.0365 |
| PC GAN - $T_\alpha(z,x)$ | 1.3849 | 0.0450 |
| JC Reverse KL - $D_\alpha(z,x)$ | 1.3786 | 0.0286 |
| JC Reverse KL - $r_\alpha(z,x)$ | 1.3934 | 0.0410 |
| JC Reverse KL - $T_\alpha(z,x)$ | 1.4133 | 0.0597 |
| JC GAN - $D_\alpha(z,x)$ | 1.4017 | 0.0286 |
| JC GAN - $r_\alpha(z,x)$ | 1.4086 | 0.0555 |
| JC GAN - $T_\alpha(z,x)$ | 1.4214 | 0.0518 |

Table 9.1

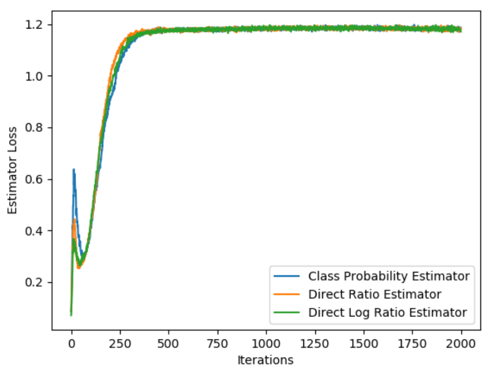


(a) GAN Divergence

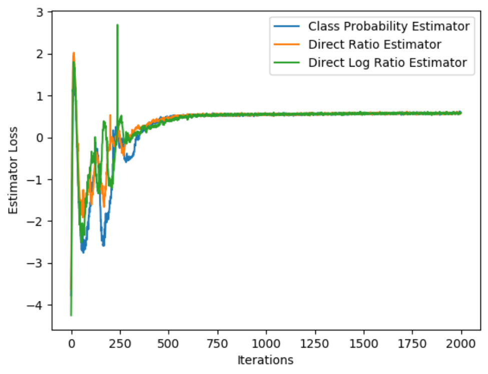


(b) Reverse KL Divergence

Figure 9.1: Prior-Contrastive Average KL Divergence

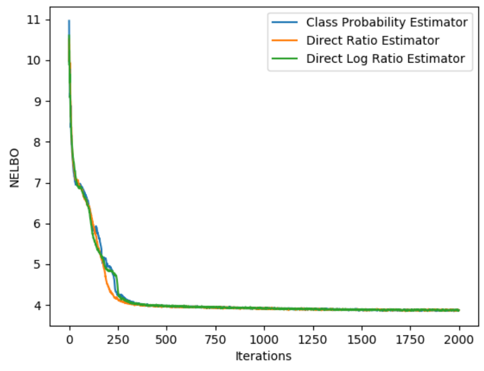


(a) GAN Divergence

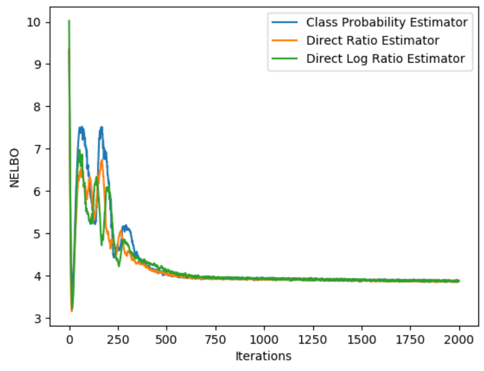


(b) Reverse KL Divergence

Figure 9.2: Prior-Contrastive Estimator Loss

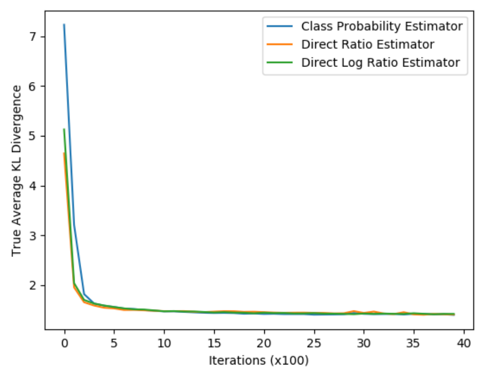


(a) GAN Divergence

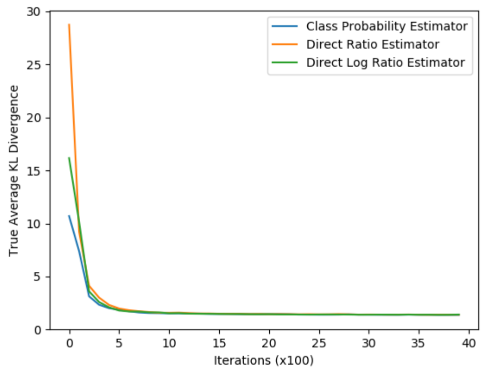


(b) Reverse KL Divergence

Figure 9.3: Prior-Contrastive NELBO

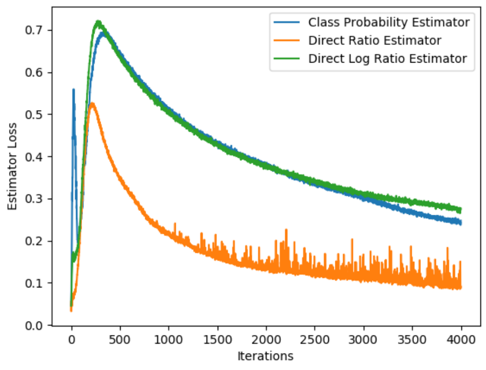


(a) GAN Divergence

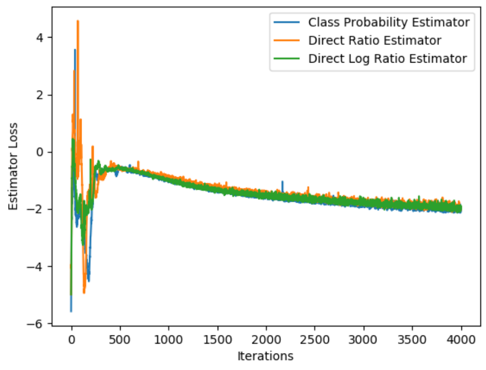


(b) Reverse KL Divergence

Figure 9.4: Joint-Contrastive Average KL Divergence

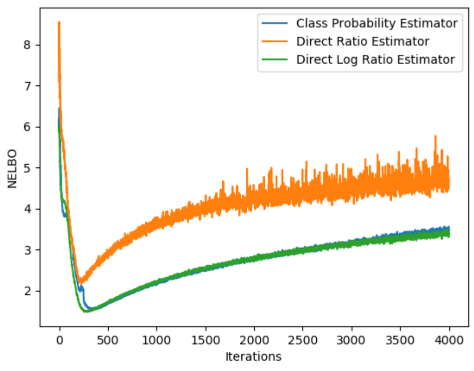


(a) GAN Divergence

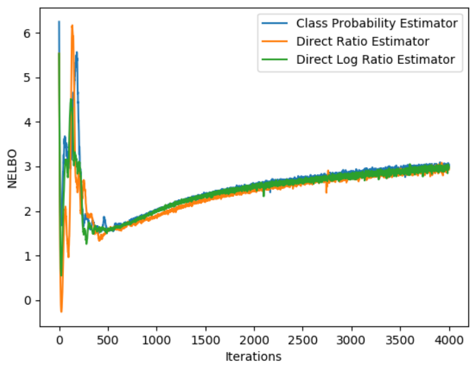


(b) Reverse KL Divergence

Figure 9.5: Joint-Contrastive Estimator Loss



(a) GAN Divergence



(b) Reverse KL Divergence

Figure 9.6: Joint-Contrastive NELBO

# Chapter 10

# Data Generation - (MNIST image generation)

## 10.1 Problem Context

The MNIST (Modified National Institute of Standards and Technology) dataset is widely used for testing machine learning algorithms related to image analysis. It contains 60,000 training and 10,000 testing images of handwritten digits, each grayscale and of 28x28 pixel size. Thus, the image distribution has 784 dimensions and follows a Bernoulli distribution. A sample of the images can be seen in Figure. In this problem, we aim to generate new MNIST images indistinguishable from those in the original dataset. To do so, we use the algorithm described in section, alternating between density ratio estimator training and simultaneous training of an encoder that maps images to a lower dimensional normal distribution and a decoder that generates new images using samples from the same latent space.
Recall that joint-contrastive algorithms are more suited to pairing between different sample spaces than for data generation, so we only formulate prior-contrastive algorithms for this experiment. The main goal of this experiment is to compare the estimator parametrizations for high dimensional data, verifying the result from the last Sprinkler experiment. We repeat the experiment with two different latent spaces, one with 2 dimensions and one with 10 dimensions, to determine if the dimensionality of the densities in the density ratio $\frac{q_\phi(z|x)}{p(z)}$ has any effect.

## 10.2 Program Structure

Both our posterior sample generator and estimator have the same structures as in the Sprinkler problem, except the number of posterior noise inputs and nodes per layer have increased to account for the higher dimensionality of the problem. Although this problem involves image analysis, due to its simplicity we refrain from using convolutional layers, similar to other experiments (insert ref). To model the problem's likelihood function, we use an additional decoder network that is trained simultaneously with the posterior. Its structure is as shown in Figure below:

A training rate of 0.0004 is used for all of the optimization steps, with a batch size of 512. Again, the estimator is pre-trained for 5000 iterations, afterwards the program alternates between 20 iterations of estimator optimizaton and 1 iteration of posterior training, for 4000 total posterior iterations.

## 10.3 Results

The higher dimensional problem shows an estimator that is clearly superior. In this part of the experiment, the values involved with estimating the direct density ratio and log density ratio are greater than the largest positive number representable by a 64-bit floating point number, and the program overflows to Inf. This occurs during the estimator initialization phase when the KL divergence between the two distributions is the greatest. The class probability estimator is the only estimator that does not experience this problem.

This is because the density ratio $\frac{q_\phi(z|x)}{p(z)}$ increases exponentially with the dimensionality of the latent space, eventually reaching a value that is too large to be represented by a computer. For both direct density ratio algorithms, the estimator network is programmed to output $\mathbb{E}_{p(z)q^*(x)}\frac{q_\phi(z|x)}{p(z)}$ directly, and if the log density ratio is estimated instead, its exponential is taken in the estimator loss function, leading back to the direct density ratio. On the other hand, the class probability estimator output is bound in $(0, 1)$, so a large density ratio would cause the estimate to output a number near 1. In fact, since the sigmoid activation function is used to map the network output from $R$ to $(0, 1)$, it can be easily shown that the input of the activation function is the log ratio (we can output this value directly, slightly reducing the computation time needed to calculate the log ratio in this parametrization). Letting $x$ be the neural network output before being mapped to $(0, 1)$, we have:

$$\begin{aligned}
\frac{1}{1+e^{-x}} &= \frac{q_\phi(z|x)}{p(z)+q_\phi(z|x)} \\
e^{-x}+1 &= \frac{p(z)+q_\phi(z|x)}{q_\phi(z|x)} \\
e^{-x} &= \frac{p(z)}{q_\phi(z|x)} \\
x &= \log\frac{q_\phi(z|x)}{p(z)}.
\end{aligned}$$

The calculations involved with the class probability estimator are therefore within the space of representable number by 64-bit, and even 32-bit floating point numbers.

Using a 32-bit representation leads to much faster computations. We can therefore conclude that the class probability estimator is superior in the sense that it is the only feasible parametrization.

# Chapter 11

# Related Work and Discussion

Leave this as a brainstorm list for now

- Of course many other problems and training parameters can be explored, variational inference and implicit models are used for more than inference and generation
- More f-divergences can be explored (Nowozin, 2016)
- Could test estimators for CycleGANs (joint-contrastive formulation for images) (Tiao, 2018)
- Could take a step back from using neural networks to estimate the divergence, instead try divergence estimation via k-nearest-neighbour distances (Wang, 2009)
- I don't know how denoisers work but they may be comparable to the methods discussed in this thesis (Huszar)
- Hopefully in the future everyone will be using class probability estimators trained under the loss function formulated by the KL divergence :)

# Chapter 12

# Conclusion

This thesis is hot trash.

# References

Bengio, Y. (2012). *Practical Recommendations for Gradient-Based Training of Deep Architectures*, pages 437–478. Springer Berlin Heidelberg, Berlin, Heidelberg.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition.* Oxford University Press, Inc., New York, NY, USA.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag, Berlin, Heidelberg.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization.* Cambridge University Press.

Cheng, B. and Titterington, D. M. (1994). Neural networks: A review from a statistical perspective. *Statist. Sci.*, 9(1):2–30.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314.

Doersch, C. (2016). Tutorial on Variational Autoencoders. *ArXiv e-prints.*

Dumoulin, V., Belghazi, I., Poole, B., Mastropietro, O., Lamb, A., Arjovsky, M., and Courville, A. (2016). Adversarially Learned Inference. *ArXiv e-prints.*

E. Rumelhart, D., E. Hinton, G., and J. Williams, R. (1986). Learning representations by back propagating errors. 323:533–536.

Floudas, C. A. (2005). *Deterministic Global Optimization: Theory, Methods and (NONCONVEX OPTIMIZATION AND ITS APPLICATIONS Volume 37) (Nonconvex Optimization and Its Applications).* Springer-Verlag, Berlin, Heidelberg.

Fuglede, B. and Topsoe, F. (2004). Jensen-shannon divergence and hilbert space embedding. In *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.*, pages 31–.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2004). *Bayesian Data Analysis.* Chapman and Hall/CRC, 2nd ed. edition.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In Gordon, G., Dunson, D., and Dudík, M., editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA. PMLR.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Networks. 3.

Goodman, L. A. (1960). On the exact variance of products. *Journal of the American Statistical Association*, 55(292):708–713.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2nd edition.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257.

Huszár, F. (2017). Variational Inference using Implicit Distributions. *ArXiv e-prints*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *ArXiv e-prints*.

Kolen, J. F. and Kremer, S. C. (2001). *Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies*. IEEE.

Kullback, S. (1959). *Information Theory and Statistics*. Wiley, New York.

Li, M., Zhang, T., Chen, Y., and Smola, A. J. (2014). Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 661–670, New York, NY, USA. ACM.

Liu, J., Gao, C., Meng, D., and Hauptmann, A. G. (2017). Decidenet: Counting varying density crowds through attention guided detection and density estimation. *CoRR*, abs/1712.06679.

Maas, A. L. (2013). Rectifier nonlinearities improve neural network acoustic models.

Mescheder, L. M., Nowozin, S., and Geiger, A. (2017). Adversarial Variational Bayes: Unifying Variational Autoencoders and Generative Adversarial Networks. *CoRR*, abs/1701.04722.

Mohamed, S. and Lakshminarayanan, B. (2016). Learning in Implicit Generative Models. *ArXiv e-prints*.

Nam, H. and Sugiyama, M. (2015). Direct density ratio estimation with convolutional neural networks with application in outlier detection. *IEICE Transactions on Information and Systems*, E98.D(5):1073–1079.

Nguyen, X., Wainwright, M. J., and Jordan, M. I. (2010). Estimating Divergence Functionals and the Likelihood Ratio by Convex Risk Minimization. *IEEE Trans. Inf. Theor.*, 56(11):5847–5861.

Nowozin, S., Cseke, B., and Tomioka, R. (2016). f-gan: Training generative neural samplers using variational divergence minimization.

P. Wellman, M. and Henrion, M. (1993). Explaining 'explaining away'. 15:287–292.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747.

Simard, P. Y., Steinkraus, D., and Platt, J. C. (2003). Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*.

Snyman, J. (2005). *Practical Mathematical Optimization*.

Sugiyama, M., Suzuki, T., and Kanamori, T. (2012). *Density Ratio Estimation in Machine Learning*. Cambridge University Press.

Tiao, L. C., Bonilla, E. V., and Ramos, F. (2018). Cycle-Consistent Adversarial Learning as Approximate Bayesian Inference. *ArXiv e-prints*.

Tran, D., Ranganath, R., and Blei, D. M. (2017). Hierarchical Implicit Models and Likelihood-Free Variational Inference. *ArXiv e-prints*.

Wu, H. (2009). Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions. 179:3432–3441.

Zhang, C., Bütepage, J., Kjellström, H., and Mandt, S. (2017). Advances in variational inference. *CoRR*, abs/1711.05597.

Zilles, K. (1992). Neuronal plasticity as an adaptive property of the central nervous system. *Annals of Anatomy - Anatomischer Anzeiger*, 174(5):383 – 391.

# Appendix A

# Kernel Density Estimation

Kernel density estimation is a non-parametric method used to estimate the probability density function of a distribution, using only samples. It can therefore be used to estimate implicit distributions. For simplicity we only explain the univariate form of the kernel density estimator, though the multivariate form is used in this thesis.

Let $\{x^{(i)}\}_{i=1}^{n}$ be an independent and identically distributed sample from a distribution with unknown probability density function $f$. Its kernel density estimator is defined as

$$\hat{f}_h(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x^{(i)}}{h}\right).$$

$K$ is the kernel, a symmetric non-negative weighting function that integrates to 1. Examples of kernel functions are:

- Epanechnikov: $K(u) = \frac{3}{4}(1-u^2), |u| \leq 1$
- Uniform: $K(u) = \frac{1}{2}, |u| \leq 1$
- Gaussian: $K(u) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}u^2\right)$.

Typically, the Gaussian kernel is used due to its statistical properties, but the choice of kernel is not as important as the choice of $h$, the bandwidth.

The bandwidth $h > 0$ acts as a smoothing parameter, determining the width of the kernel. If $h$ is too small, $\hat{f}$ will be 'undersmoothed' as too much weight is placed on the areas nearest the data-points, leading to a spiky estimate with high variance. On the other hand, if $h$ is too large, $\hat{f}$ will be 'oversmoothed' with too little weight on areas nearest to the data-points, resulting in a relatively flat estimate with high bias. It is therefore ideal to choose $h$ such that the mean integrated square error $MISE(h) = \mathbb{E}\left[\int (\hat{f}_h(x) - f(x))^2 dx\right]$ is minimized. For a Gaussian kernel, this is approximately $h = 1.06\hat{\sigma}n^{-1/5}$ where $\hat{\sigma}$ is the sample standard deviation. We omit the proof in this thesis.

The kernel density estimator works by placing a kernel on each data point and summing up the kernels to produce a smooth curve. Each point on the curve is essentially a weighted average of nearby data points. Regions of the curve with many data points will therefore have a high estimated probability density.