DENSITY RATIO ESTIMATORS FOR VARIATIONAL METHODS IN BAYESIAN NEURAL NETWORKS

Alexander Lam

Supervisor: Professor Scott Sisson

School of Mathematics and Statistics
UNSW Sydney

September 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 θ $p_\theta(x|z)$. In this particular field, there are three main problems to solve:

1. Estimation of θ, 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

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, explaining two common representations in literature and programming. After describing the network initialisation method, we conclude the chapter by demonstrating how neural networks are trained with gradient descent and how back-propagation is used to find the partial derivatives for gradient descent.

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 [?].

One key feature of the brain structure is the capability for the neurons to adapt to suit different purposes [?]. 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 purpose [?]. These are the core principles behind neural networks.

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 $\mathbf{\Theta}$ from input $\mathbf{x}$ to output $\mathbf{y}$: $\mathbf{y} = \mathbf{f}_{\mathbf{\Theta}}(\mathbf{x})$. In fact, the universal approximation theorem states that neural networks can approximate any function [?, ?]. 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 price. Another example is the classification problem of recognising handwritten digits (0-9) in a black and white image [?]. 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 would be 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 is the equivalent of the intercept term in a regression problem. Using the example of a node function $h_{\boldsymbol{\theta}}(\mathbf{x})$ with 3 inputs (see Figure 2.1 below), we label these inputs as $\mathbf{x} = [x_0 \quad x_1 \quad x_2 \quad x_3]^\intercal$, with $x_0 = 1$ corresponding to the bias node. These values are multiplied by weights $\boldsymbol{\theta} = [\theta_0 \quad \theta_1 \quad \theta_2 \quad \theta_3]$, and then passed through an activation function $g(\boldsymbol{x})$ [?]. The objective of the activation function is to normalise the network output to a given range, such as $(0, 1)$ or $\mathbb{R}$. A list of common activation functions is given on the next page.

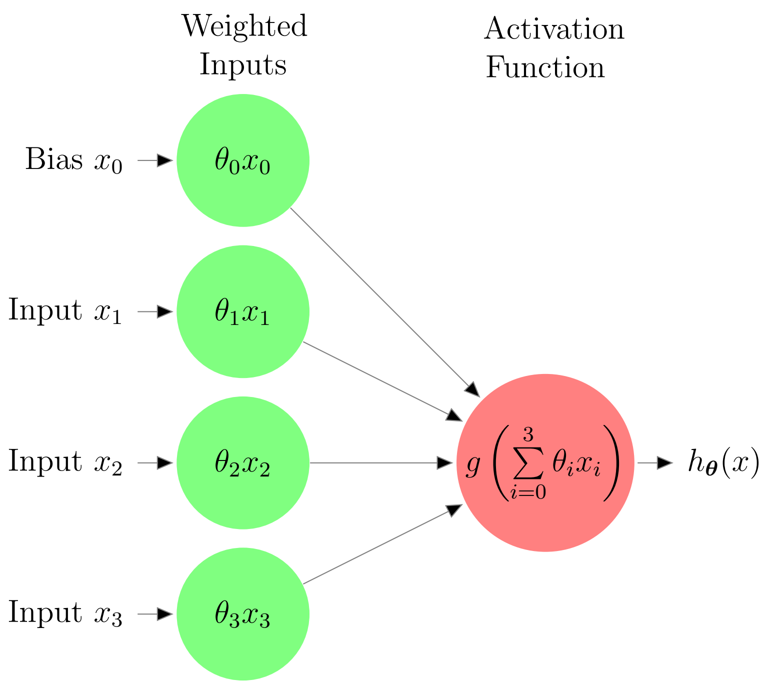


Figure 2.1: Individual Node Structure

Some common activation functions are [?]:

- 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 $h_{\boldsymbol{\theta}}(\mathbf{x}) = \max\{0, \boldsymbol{\theta}^\top \mathbf{x}\}$.
- 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 $h_{\boldsymbol{\theta}}(\mathbf{x}) = (1 + \exp(-\boldsymbol{\theta}^\top \mathbf{x}))^{-1}$.
- The hyperbolic tangent function output ranges between $(-1, 1)$, denoted as $g(x) = \tanh(x)$ corresponding to $h_{\boldsymbol{\theta}}(\mathbf{x}) = \tanh(\boldsymbol{\theta}^\top \mathbf{x})$.
- The linear activation function is used to describe nodes with no activation function, as its formula is $g(x) = x$, corresponding to $h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^\top \mathbf{x}$. It therefore ranges in $\mathbb{R}$.

Their plots are shown in Figure 2.2 below.

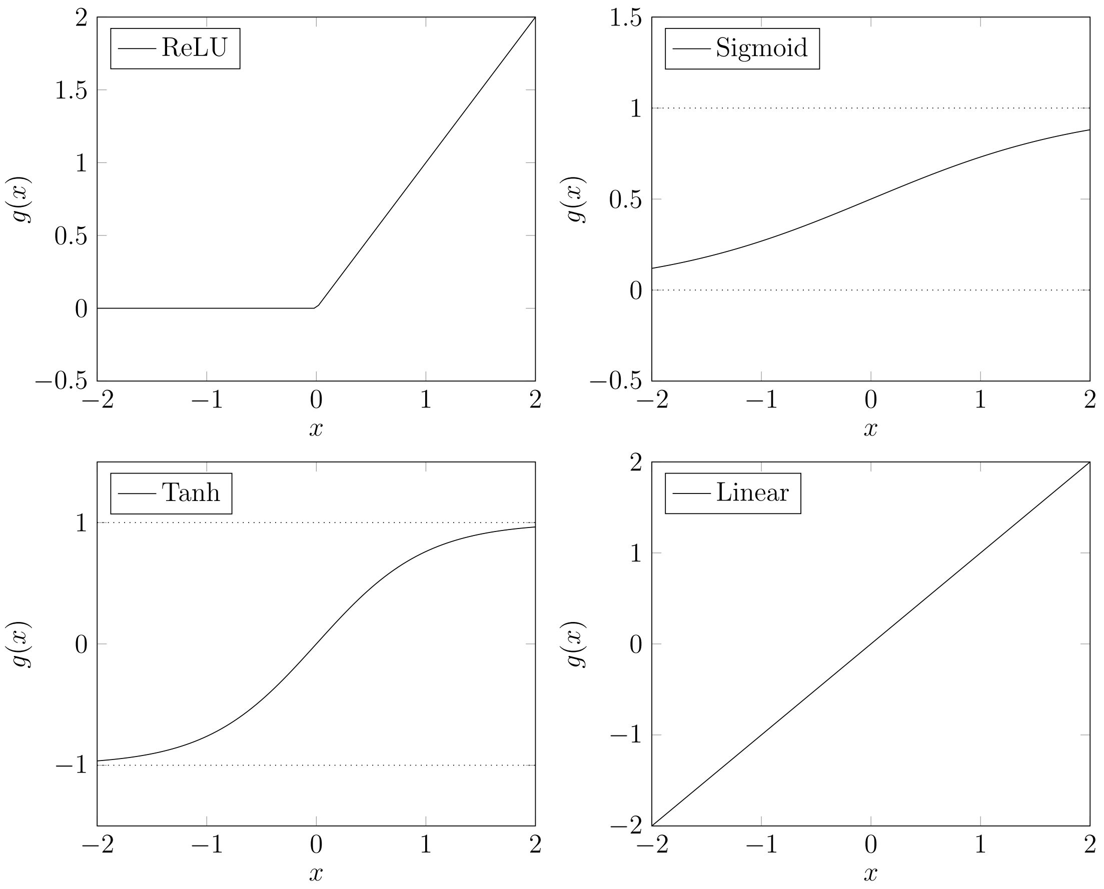


Figure 2.2: Activation Function Plots

The choice of activation function is mostly dependent on the desired range of the node's output. For example, if the node's output is the estimate of a probability (ranging in $(0,1)$), then the sigmoid function would be used. 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.

2.3 Neural Network Structure

A typical neural network is made up of layers of interconnected nodes [?]. 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; its result is simply the output of the network. The layers of nodes between the input and output layer are the hidden layers, as their weights and outputs are not useful to 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. Figure 2.3 below illustrates a simple neural network function $f_{\boldsymbol{\Theta}}(\boldsymbol{x})$ with 3 external inputs, 1 hidden layer with 3 nodes, 1 bias node per non-output layer and 1 output node. This example, along with the notations is described in Example 2.3.1 on the next page.

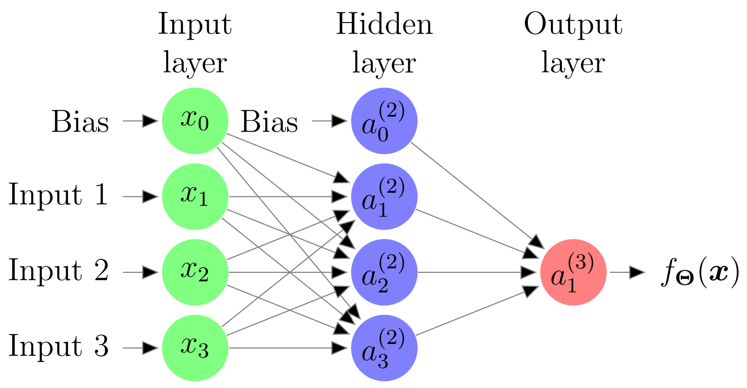


Figure 2.3: Neural Network Structure

Example 2.3.1. In this example, 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 $\Theta^{(j)}$. 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.
Individually, the outputs in the hidden nodes and the output node are:

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

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

Denoting the weights outputting to unit i in layer $j+1$ as $\boldsymbol{\theta}_i^{(j)} = [\Theta_{i,0}^{(j)} \quad \Theta_{i,1}^{(j)} \cdots \Theta_{i,k}^{(j)}]^\intercal$ where $k+1$ is the number of inputs, we have the vectorized notation:

$$a_0^{(2)} = 1$$

$$\begin{aligned}
a_1^{(2)} &= g((\boldsymbol{\theta}_1^{(1)})^\intercal \boldsymbol{x})\\
a_2^{(2)} &= g((\boldsymbol{\theta}_2^{(1)})^\intercal \boldsymbol{x})\\
a_3^{(2)} &= g((\boldsymbol{\theta}_3^{(1)})^\intercal \boldsymbol{x})\\
f_{\boldsymbol{\theta}}(\boldsymbol{x}) = a_1^{(3)} &= g(\boldsymbol{\theta}_1^{(2)})^\intercal \boldsymbol{a}^{(2)})
\end{aligned}$$

where $\boldsymbol{a}^{(2)} = [a_0^{(2)} \quad a_1^{(2)} \quad a_2^{(2)} \quad a_3^{(3)}]^\intercal$.
An even simpler notation is:

$$a_0^{(2)} = 1$$

$$[a_1^{(2)} \quad a_2^{(2)} \quad a_3^{(2)}] = g((\Theta^{(1)})^\intercal \boldsymbol{x})$$

$$f_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \boldsymbol{a}^{(3)} = g((\Theta^{(2)})^\intercal \boldsymbol{a}^{(2)}).$$

2.4 Bias-per-node Representation

The use of a bias node leads to a minor programming inconvenience: the number of inputs in each layer is one more than the number of outputs of each node in the preceding layer. To make these values consistent, the bias node is often replaced with an intercept term added to the weighted input of a node before it passes through the activation function [?]. These intercept terms are optimized alongside the weights. In either case, the bias takes the form of a scalar added to the weighted inputs, which can be optimized. There is no practical difference between these two notations: the sum of intercept terms in a layer is equivalent to the weight corresponding to a bias node for the layer. Figures 2.4 and 2.5 below illustrate this concept, repeating the example of an individual node with three inputs, but with an individual intercept b_i per node.

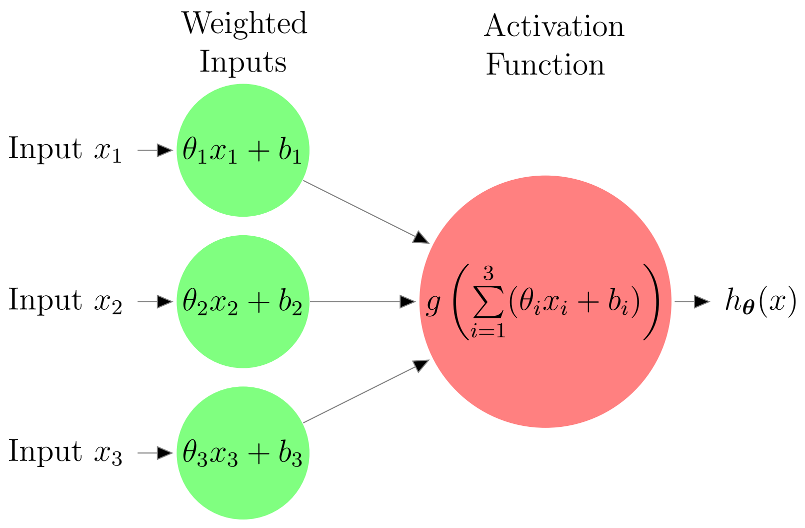


Figure 2.4: Individual Node Structure: Bias-per-node Representation

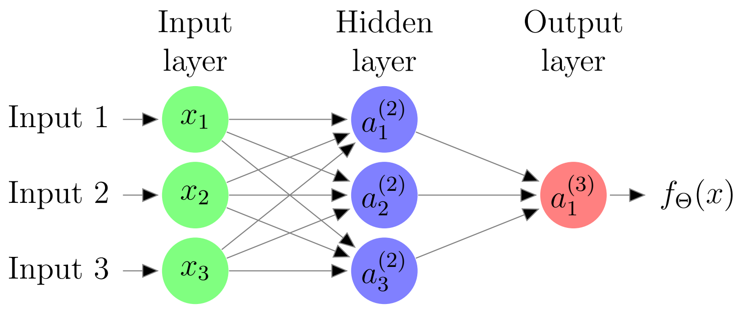


Figure 2.5: Neural Network Structure: Bias-per-node Representation

Note here that $\sum_{i=1}^{3} b_i = \theta_0 x_0$. The representation of the simple neural network example in Example 2.3.1 becomes:

$$a_1^{(2)} = g(\Theta_{1,1}^{(1)} x_1 + b_1^{(1)} + \Theta_{1,2}^{(1)} x_2 + b_2^{(1)} + \Theta_{1,3}^{(1)} x_3 + b_3^{(1)})$$

$$a_2^{(2)} = g(\Theta_{2,1}^{(1)} x_1 + b_1^{(1)} + \Theta_{2,2}^{(1)} x_2 + b_2^{(1)} + \Theta_{2,3}^{(1)} x_3 + b_3^{(1)})$$

$$a_3^{(2)} = g(\Theta_{3,1}^{(1)} x_1 + b_1^{(1)} + \Theta_{3,2}^{(1)} x_2 + b_2^{(1)} + \Theta_{3,3}^{(1)} x_3 + b_3^{(1)})$$

$$f_{\boldsymbol{\Theta}}(\boldsymbol{x}) = a_1^{(3)} = g(\Theta_{1,1}^{(2)} a_1^{(2)} + b_1^{(2)} + \Theta_{1,2}^{(2)} a_2^{(2)} + b_2^{(2)} + \Theta_{1,3}^{(2)} a_3^{(2)} + b_3^{(2)})$$

or in vectorized notation,

$$\boldsymbol{a}^{(2)} = g((\Theta^{(1)})^{\intercal} \boldsymbol{x})$$

$$f_{\boldsymbol{\Theta}}(\boldsymbol{x}) = \boldsymbol{a}^{(3)} = g((\Theta^{(2)})^{\intercal} \boldsymbol{a}^{(2)})$$

2.5 Choice of Activation Function

In this section we describe the activation functions commonly used in the different layers of a neural network. Recall in section 2.2 the four most common activation functions are:

- The rectified linear unit or ReLU activation function $g(x) = \max\{0, x\}$ ranging in $[0, \infty)$.
- The sigmoid or logistic activation function $g(x) = (1 + \exp(-x))^{-1}$ restricted to $(0, 1)$.
- The hyperbolic tangent function $g(x) = \tanh(x)$ ranging between $(-1, 1)$.
- The linear activation function $g(x) = x$ ranging in $\mathbb{R}$.

Rectified linear units are the default choice for the hidden layers for their many advantages [**?**]

- Reduces overfitting: since negative ReLU inputs result in a zero output, on average only half of the units are "active" (non-zero output) when the network is first initialized. The proportion of active units can increase or decrease during training such that overfitting and underfitting is minimized.
- Faster computation: a $\max\{0, x\}$ function is computed much faster than a function that uses exp or tanh.
- Easier and more consistent training: weight training in a neural network (discussed in Section 2.6) uses the gradient of overall node function, which is influenced by the activation function. The rectified linear unit has a consistent gradient θ or 0 resulting in consistent training, as opposed to more complex

units. Since the ReLU function is not differentiable at 0, the derivative $g'(0)$ is usually set to 0.

Since the input layer has no activation function, it can be described as having a linear activation function.

For the output layer, the activation function with the most reasonable output range is used. This depends on what the network is being used for. For example, in classification, the network assigns a probability to each case, so a sigmoid activation function would be most reasonable, as it ranges between 0 and 1. On the other hand, a rectified linear unit would be used for regression of a non-negative quantity such as price or time.

2.6 Weight Initialization

Proper initialization of the weights is ideal to improve convergence, as if the weights are too low, then the nodal outputs will continually decrease through the layers and become very small, requiring many iterations of training to fix. Similarly, if the weights are too high, then the result output of the network will be extremely large. In this section we discuss Xavier Initialization [**?**], which aims to keep the signal variance constant throughout the network. To derive the initialization algorithm, first consider in the bias-per-node representation a single node with n inputs, and let z denote the weighted sum of the inputs $\boldsymbol{\theta}^\top \mathbf{x}$ before it is passed through the activation function. This is written as

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

$\sum_{i=1}^{n} b_i$ is a constant term, so $\mathrm{Var}(\sum_{i=1}^{n} b_i) = 0$. Now under the assumption that the inputs and weights have 0 mean, we find the variance of the other terms:

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

Assuming that the weights and inputs are also independent and 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

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

2.7 Optimization

The goal of optimizing the network is to train the weights of the network such that a loss function, which we will denote as L as 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. [?]
The weights are initialized randomly, and through back-propagation (Section 2.8), their values are used to calculate the partial derivative of the loss function with respect to each individual weight (and bias). These partial derivatives are used in the gradient-based optimization of the weights. There are many variations of neural network optimization algorithms, but they are mostly based off gradient descent, which we will cover in this section. [?]

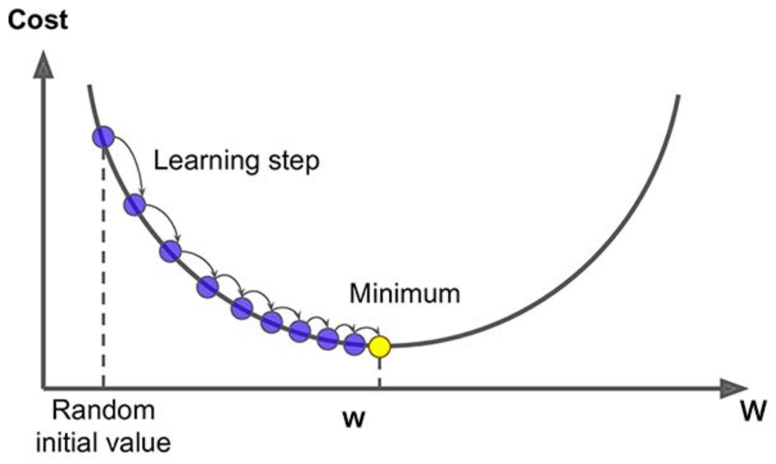


Figure 2.6: Gradient Descent

Definition 2.7.1. At point $\boldsymbol{x}^{(n)}$, $\boldsymbol{s}^{(n)}$ is a descent direction if $\nabla f\left(\boldsymbol{x}^{(n)}\right)^\intercal \boldsymbol{s}^{(n)} < 0$.

Gradient descent [?] 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 $\boldsymbol{s}^{(n)}$:

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

A simple diagram illustrating this is shown in Figure 2.6 above.

Proposition 2.7.2. *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

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

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

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

$$= \sum_{i=1}^{k} \frac{\partial f(\boldsymbol{x})}{\partial x_i}|_{\boldsymbol{x}=\boldsymbol{x}^{(n)}+\alpha \boldsymbol{s}^{(n)}} \boldsymbol{s}_i^{(n)}$$
$$= \nabla f(\boldsymbol{x}^{(n)} + \alpha \boldsymbol{s}^{(n)})^\intercal \boldsymbol{s}^{(n)}$$

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

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

Therefore for small $\alpha > 0$,

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

$\square$

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 $-\nabla f(\boldsymbol{x}^{(n)})^\intercal \nabla f(\boldsymbol{x}^{(n)}) = -||\nabla f(\boldsymbol{x}^{(n)})||^2 < 0$.
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 [?]. 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, and as previously described, its smoothness can cause the algorithm to converge to local minima. In stochastic gradient descent, only one observation is used per iteration, so the latent randomness associated with each observation effectively leads to noise added to each step, and 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 [?].
Gradient descent convergence can be improved by using an adaptive learning rate (ie. decreasing α 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 [?].
The Adam algorithm [?] incorporates these two concepts to form an effective optimization algorithm that is commonly applied to neural networks. In this thesis, we use this algorithm, but omit the specifics.

2.8 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

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

so that gradient descent can be performed to optimize the weights. [?] For each training sample $(\boldsymbol{x}^{(I)}, \boldsymbol{y}^{(I)})$, $I = 1, \dots, N$, the input signal is propagated forward throughout the network to calculate $\boldsymbol{a}^{(j)}$ for $j = 2, \dots, J$, 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, preventing overfitting [**?**]

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

There is no significant change when the bias node is regularized.
Pseudocode for back-propagation is shown in Algorithm 1 on the next page.
Having derived the partial derivatives of the loss function with respect to the individual weights, we can use a gradient descent based optimization method to update the weights. The partial derivatives are re-calculated after each optimization update until convergence is achieved.

```
Data: Training Data $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \dots, (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})\}$, Regularizer Term $\lambda$
Result: Cost Function Partial Derivatives $\frac{\partial}{\partial \Theta^{(j)}_{m,n}} L(\Theta)$
begin
    Randomly initialize weights $\Theta$;
    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 different, 'variational' distribution to approximate posterior distributions in the context of Bayesian statistics. The name comes 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, their parametrisation is unknown but it is possible to sample from them.

3.1 Context

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

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{z}, \mathbf{x})}{p(\mathbf{x})} = \frac{p(\mathbf{z})p(\mathbf{x}|\mathbf{z})}{\int_{\mathbf{z}} p(\mathbf{z}, \mathbf{x}) d\mathbf{z}}$$

where

- $p(\mathbf{z})$ is the prior distribution: the initial distribution of $\mathbf{z}$ before the data $\mathbf{x}$ is observed. This can be initialised to represent our initial beliefs, or it can be parametrised randomly,
- $p(\mathbf{x}|\mathbf{z})$ is the likelihood: the distribution of data $\mathbf{x}$ conditioned on the parameters $\mathbf{z}$,
- $p(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}, \mathbf{x}) d\mathbf{z}$ is the marginal likelihood, or the evidence: the density of the data averaged across all possible parameter values.

If the evidence integral $p(\mathbf{x}) = \int_{\mathbf{z}} p(\mathbf{z}, \mathbf{x}) d\mathbf{z}$ is computationally intractable, then we are unable to evaluate the posterior density. Traditional MCMC (Markov Chain Monte Carlo) methods overcome this obstacle by sampling from a Markov chain that converges to the stationary distribution $p(\mathbf{z}, \mathbf{x})$. However, these methods tend to have slow convergence for large datasets or complex models. 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, but they are known to underestimate the true posterior variance [?], leading to a trade-off between speed of convergence and mean squared error.

3.2 The KL Divergence

We now present the KL (Kullback-Leibler) divergence, a type of f-divergence that is commonly used in variational inference. It is a measure of how much two probability distributions differ [?].

Definition 3.2.1. 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 3.2.2. *The KL divergence is not symmetric:*

$$KL(p(x)||q(x)) \neq KL(q(x)||p(x)).$$

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. The expectation is taken with respect to different distributions.

Lemma 3.2.3. *The KL divergence is non-negative, and it is equal to zero if and only if $p(x) = q(x)$ for almost all x:*

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

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

Proof. In the third line we use $\log x \leq x - 1 \quad \forall x > 0$ 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.

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

□

3.3 Introduction to Variational Inference

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

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

where D_f denotes an f-divergence [**?**]. 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:

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

From lemma 3.2.3, it is evident that $KL(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x}))$ attains a minimal value of 0 when $q(\mathbf{z}) = p(\mathbf{z}|\mathbf{x})$.
There is an issue with solving equation 3.3.2 directly: we cannot evaluate the reverse KL divergence as $p(\mathbf{z}|\mathbf{x})$ is unknown, a result of $p(\mathbf{x})$ being intractable. 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:

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

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

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

We refer to $\log p(\mathbf{x}) - KL(q(\mathbf{z})||p(\mathbf{z}|\mathbf{x}))$ 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 $q(\mathbf{z})$ and $p(\mathbf{z}|\mathbf{x})$ 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:

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

Note that this expression attains a maximum at the marginal likelihood of the dataset $\log p(\mathbf{x})$, hence we may use the ELBO to construct a model selection criterion.

3.5 Mean-Field Variational Family

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

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

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 ϕ_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:

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

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

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

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

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:

$$
ELBO(q) - \sum_{i=1}^{M} \lambda_i \int_{\mathcal{z}_i} q_i(z_i) dz_i = 0
$$

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

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

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

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

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

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

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:

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

This expression can be used in an expectation-maximization algorithm, in which the $q_j^*(z_j)$ is evaluated and iterated from $j = 1 \dots M$. This particular algorithm is called coordinate ascent variational inference (CAVI) (Algorithm 2):

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

Consider the hierarchical model

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

This is a Bayesian mixture of univariate Gaussian random variables with unit variance. In this model, we draw K μ_k variables from a prior Gaussian distribution $N(0, \sigma^2)$ (σ^2 is a hyperparameter), forming the vector $\boldsymbol{\mu} = (\mu_1, \dots, \mu_K)^\top$. 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 contains the 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 $N(c_i^\top \boldsymbol{\mu}, 1)$.

Here, our latent variables are $\mathbf{z} = \{\mathbf{c}, \boldsymbol{\mu}\}$. Assuming n samples, our joint density is

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

From this, we derive the marginal likelihood

$$p(\mathbf{x}) = \int p(\boldsymbol{\mu}) \prod_{i=1}^{n} \sum_{c_i} p(c_i) p(x_i | c_i, \boldsymbol{\mu}) d\boldsymbol{\mu}.$$

This integral is intractable, as the time complexity of evaluating it is $\mathcal{O}(K^n)$, which is exponential in K. To evaluate the posterior distribution over the latent variables $p(\boldsymbol{\mu}, \mathbf{c} | \mathbf{x})$, we would have to apply variational inference, approximating it with a variational distribution $q(\boldsymbol{\mu}, \mathbf{c})$. We will assume this distribution follows the mean-field variational family:

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

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

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

Using this and equation 3.6.1, we can derive the evidence lower bound as a function of the variational parameters:

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

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

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

Now since c_i 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:

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

In each line, terms constant with respect to c_{ik} have been taken out of the expression. Our optimal categorical factor becomes

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

By proportionality, we then have the variational update

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

Now we find the variational density of the kth mixture component, again using equation 3.5.7 with the ELBO and ignoring terms independent of $p(.)$ and μ_k:

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

The log of this density is

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

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 ϕ_{ik} and the variational density parameters m_k and s_k^2 until the ELBO converges.

```
Data: Data $\mathbf{x}$, 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 $\mathbf{m}, \mathbf{s}^2$ and $\boldsymbol{\phi}$;
    while ELBO has not converged do
        for $i = 1$ to $n$ do
            Set $\phi_{ik} \propto \exp(\mathbb{E}[\mu_k; m_k, s_k^2] x_i - \mathbb{E}[\mu_k^2; m_k, s_k^2]/2)$;
        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 $ELBO(\mathbf{m}, \mathbf{s}^2, \boldsymbol{\phi})$;
    end
    Return $q(\mathbf{m}, \mathbf{s}^2, \boldsymbol{\phi})$;
end
```

Algorithm 3: CAVI Algorithm for Bayesian mixture of Gaussians

3.7 Amortized Inference

One disadvantage of mean field variational inference is that a specific set of variational parameters needs to be derived and optimized for each data point. This can be computationally expensive for large datasets. 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 [?]. Our variational distribution therefore conditions on the observation, taking the form

$$q_\phi(z|x).$$

Figures 3.1 and 3.2 highlight the difference between mean-field and amortized variational inference. In the figures there are N data points, and for each data point, $\boldsymbol{z}^{(i)}$ is M-dimensional. Therefore for mean-field variational inference, there are $M \times N$ sets of ϕ parameters, whilst in amortized inference there is only one set of ϕ parameters, as we use $M \times N$ data points as inputs.

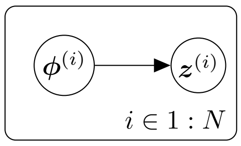


Figure 3.1: DAG for Mean-Field Variational Inference

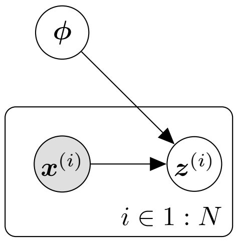


Figure 3.2: DAG for Amortized Inference

Clearly, a very complex variational distribution is required to model such a structure, so it often takes the form of a neural network with 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 $p(x)$ represents the marginal likelihood, or 'true' distribution of the data. This is typically never available, we often instead have a sample dataset representing it. Denoting the dataset as $q^*(x)$, we want to optimize ϕ across the observations from our dataset, so our objective now is to choose parameters ϕ such that the expected KL divergence with respect to $q^*(x)$ is minimized:

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

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

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

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

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

Since ϕ 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 is often represented as a neural network parametrized by θ: $p_\theta(x|z)$. 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 $q_\phi(z|x)$ that "compresses" a data point x into a lower dimensional latent representation z, and a decoder $p_\theta(x|z)$ that "reconstructs" the data point from the latent variable [**?**]. It has two main purposes: representation learning of data and data generation. Typically, the prior $p(z)$ is simply a standard k-dimensional multivariate normal distribution $\mathcal{N}(0, I_{k\times k})$. We now reiterate our optimization problem in line 3.7.1, this time including the optimization of the decoder parameters θ:

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

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 a standard 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 properties of a probability distribution. We can therefore generate new data x by sampling $z \sim p(z)$ and feeding it through the decoder.

At our current formulation, we have the problem that a neural network is deterministic, so for any given data point x, the encoder will always output the same z, which also contradicts the properties of a distribution. The solution is to have the encoder output a mean vector $\boldsymbol{\mu} = (\mu_1, \mu_2, \cdots, \mu_k)^\intercal$ and variance vector $\boldsymbol{\sigma}^2 = (\sigma_1^2, \sigma_2^2, \cdots, \sigma_k^2)^\intercal$, each with dimensions equal to that of latent variable z. We have now defined z as an output from a multivariate normal distribution with means and variances as specified by the encoder output:

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

In practice, to reduce computational time, this is achieved by sampling random standard normal noise ϵ, multiplying it by the variance and adding the result to the mean:

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

The expression in our optimization problem is tractable. It is often possible to parametrise an explicit form of the likelihood from the neural network output. For example, if $q^*(x)$ is a data set of grey-scale images, then a Bernoulli distribution can be used to represent the pixel values, with parameters given by the sigmoid output of the likelihood network. Since we have the explicit form of the multivariate normal $q_\phi(z|x)$ and $p(z)$ densities, the KL divergence term can be calculated through the equation:

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

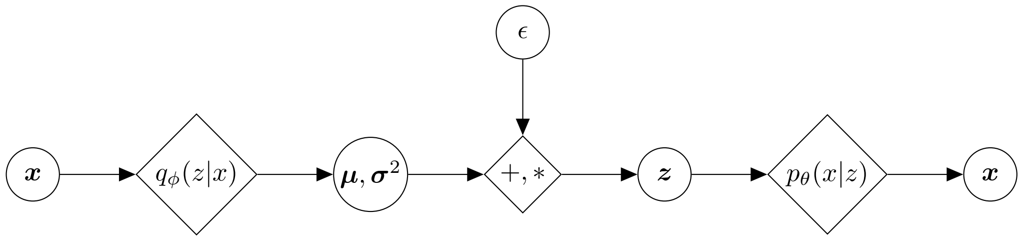


Figure 3.3: Variational Autoencoder

3.9 Problems with Implicit Distributions

The above sections assume that the true prior $p(z)$, likelihood $p(x|z)$ and variational posterior $q_\phi(z|x)$ have an explicit form. However, there are two main scenarios in which at least one of these densities are implicit, that is, their parametrization is unknown but we are able to 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 true prior or variational posterior densities is implicit, but the likelihood is known. An implicit prior is rare but may occur in posterior inference. The implicit variational posterior occurs more often (in both posterior inference and data generation), detailed in Adversarial Variational Bayes by Mescheder [**?**]. 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 $(0, I_{p\times p})$

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. We therefore denote the generator of posterior samples as $\mathscr{G}(\epsilon; x)$. A diagram illustrating this is shown in Figure 3.4 below.

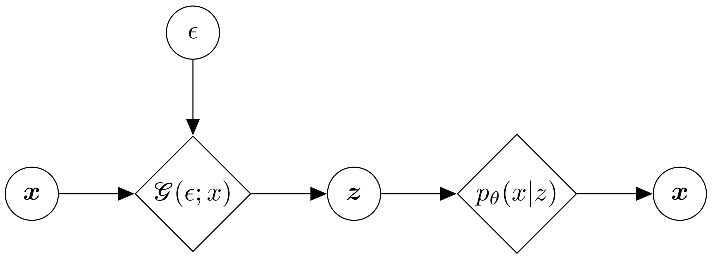


Figure 3.4: Adversarial Variational Bayes

Due to the complicated nature of the neural network, it is extremely difficult to discern the explicit form of $q_\phi(z|x)$ 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:

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

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

3.9.2 Implicit Likelihood

If the likelihood distribution is implicit, then the optimization problem is intractable even with density ratio estimation. This can occur in posterior inference or in some data generation algorithms [?]. Consequently, we rephrase the problem, instead considering a density ratio between the two joint distributions [?]. We begin 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:

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

Recall that $p(x)$ is typically unavailable, so we are unable to evaluate $KL(q^*(x)||p(x))$ even with density ratio estimation techniques. We therefore add it to both sides of the equation, leading to the following expression for the negative evidence lower bound:

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

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

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

Note that we are now minimizing the reverse KL divergence between the two joint densities $KL(q(z,x)||p(z,x))$, so the NELBO attains its minimum when $q(z,x) = p(z,x)$. The optimal posterior is therefore $q_\phi(z|x) = \frac{p(x)p(z|x)}{q^*(x)}$, dependent on the density ratio between the true data distribution $p(x)$ and our dataset $q^*(x)$.
Also 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, both the prior and variational posterior densities can also be implicit, hence the "Adversarial Variational Bayes" representation of the variational network can be used.
In data generation, we would additionally have to maximize the forward KL divergence with respect to the likelihood parameters [**?**]:

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

Chapter 4

Density Ratio Estimation

In this chapter, we derive two common methods of density ratio estimation: 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" [?].

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 $p(u)$: $U_p = \{u_1^{(p)}, \dots, u_m^{(p)}\}$ and label them with $y = 0$, then we take n samples from $q(x)$: $U_q = \{u_1^{(q)}, \dots, u_n^{(q)}\}$ 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 α trained with Bernoulli loss:

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

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

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:

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

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

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 $L_D = \log 4$.*

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 θ and optimize the same function with respect to those parameters. 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

Definition 4.2.1. The f-divergence of continuous probability density $q(u)$ from $p(u)$ is

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

where f is a convex function 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 4.2.2. *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 $r(u) = q(u)/p(u)$.

For the reverse KL divergence, $f(u) = u\log u$, so we have

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

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 4.2.2, we derive the following lower bound for the KL divergence:

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

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 α and minimize the negative of expression (4.2.1) with respect to α:

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

Note we have removed the +1 term as it is independent of α. 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 4.2.3. *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:

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

□

Corollary 4.2.4. *$r^*_\alpha(u)$ is bound in $(0, \infty)$.*

Remark 4.2.5. *When $q(u) = p(u)$, $r^*_\alpha(u) = 1$ and $L_r = 1$.*

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 4.2.2, we have the lower bound for tne 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 α, 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:

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

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

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

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

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 4.2.2, 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 α:

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

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 4.2.2 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. First, we derive an expression for the respective divergence, which we denote as D_{CPE}:

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

Recall one of the conditions of f to form an f-divergence is that $f(1) = 0$, but in this case, $f(1) = -\log 4$, hence the constant term added to our Jensen-Shannon 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 4.2.2, we derive the following lower bound for the Jensen-Shannon divergence:

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

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

5.1 Problem Context

For the first experiment, we use the simple ”Continuous Sprinkler” model as described in (Huszar 2017). 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.

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

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

Since the joint-contrastive algorithms assume both the prior and likelihood distributions are implicit, therefore only requiring samples from them, their likelihood functions are not required. The prior-contrastive algorithm requires the likelihood function of the likelihood distribution:

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

The 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:

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

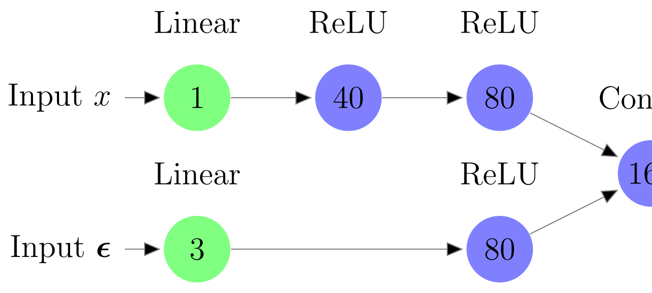
This problem exhibits ”explaining away” (Explaining ”Explaining away” Wellman 1993), a pattern of reasoning in which the confirmation of one cause of an effect reduces the need to discuss alternative causes. In this example, despite the two possible causes of the wet grass being independent, when we condition on the observation of wet grass, the causes become dependent, as either cause being significant could lead to wet grass.

In the experiment, we let $\sigma^2 = 2$, and consider observations $x = 0, 5, 8, 12, 50$. To plot the true (unnormalised) posterior $p(z|x)$, we simply 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 (**need to insert figure**). The plots are not affected by the lack of normalisation, as $p(x)$ is constant for each x-value, and the purpose of the plots is to show the shape and relative density of the posterior distribution.

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 is too simple here.

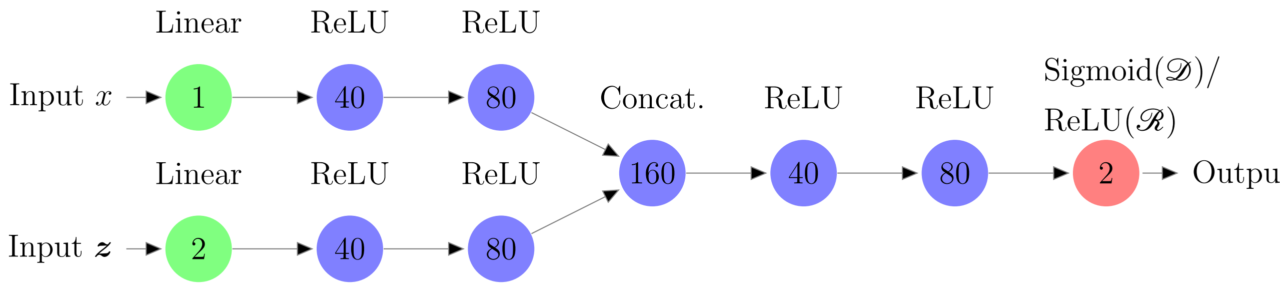
5.2 Program Structure

As described in section (), our generator $\mathscr{G}_\phi(x, \epsilon)$ is a neural network that takes in noise $\epsilon \sim \pi(\epsilon)$ along with the data sample $x \sim q^*(x)$ to create a sample from the variational posterior distribution $z \sim q_\phi(z, x)$. In this experiment, we have 3 noise inputs $\epsilon_1, \epsilon_2, \epsilon_3$ 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 idk:
The number inside the node indicates how many nodes the layer has, and the text above the node describes the activation function in the layer. Recall that the input layer does not have an activation function (Linear), 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 placed on the layer. Since arbitrary probability distribution samples can take any real number, we do not use an activation function for the output layer.

The structure of the estimator (discriminator $\mathscr{D}_\alpha(z, x)$ or ratio estimator $\mathscr{R}_\alpha(z, x)$) 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 in terms of code, the only differences between class probability estimation and divergence minimisation are the activation function of the estimator's output layer and the loss functions being minimised. A sigmoid output layer is used for a discriminator, whilst a rectified linear output is used for a ratio estimator. The estimator structure is shown in Figure below:



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 runs show that these are the highest learning rates at which both algorithms converge consistently; any higher learning rates lead to the program constantly entering a "failure" state (discussed in next section). The network is only trained on the 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 relatively large batch size makes training more consistent, as performing back-propagation along a stochastic neural network can lead to random and erratic training. To prevent problems associated with taking $\log 0$, 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 weights and biases of the estimator will be approximately the same for a fixed initialization of the generator. Afterwards, the generator is optimized for 10000 iterations in the implicit prior formulation, and 40000 iterations when we have an implicit likelihood. The algorithm alternates between 100 training steps of the estimator and 1 training step of the generator. This process for class probability estimation and divergence minimisation is shown in Algorithm and Algorithm below:

```
Data: Dataset $q^*(x) = \{0, 5, 8, 12, 20\}$, true (implicit) prior
      $p(z) \sim \mathscr{N}(0, 2I_{2\times 2})$, true (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 5001 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 $\{z_q^{(i,j)}\}_{i=1}^{1000} = \mathscr{G}(\epsilon^{(i,j)}; x_q^{(i,j})$;
    end
    update $\alpha$ by optimization step on
     $\min_\alpha -\frac{1}{1000}\sum_{i=1}^{1000}\{\log[D_\alpha(\mathscr{G}_\phi(\epsilon^{(i,j)}; x^{(i,j)}), x^{(i,j)})] + \log[1 -$
     $D_\alpha(z_p^{(i,j)}, x^{(i,j)})]\}$;
end
for $j = 1$ to 10001 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 $\{z_q^{(i,k)}\}_{i=1}^{1000} = \mathscr{G}(\epsilon^{(i,k)}; x_q^{(i,k)})$;
        end
        update $\alpha$ by optimization step on
         $\min_\alpha -\frac{1}{1000}\sum_{i=1}^{1000}\{\log[D_\alpha(\mathscr{G}_\phi(\epsilon^{(i,k)}; x^{(i,k)}), x^{(i,k)})] + \log[1 -$
         $D_\alpha(z_p^{(i,k)}, x^{(i,k)})]\}$;
    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 $\phi$ by optimization step on
     $\min_\phi \frac{1}{1000}\sum_{i=1}^{1000}\{-\log p(x|\mathscr{G}_\phi(\epsilon^{(i)}; x^{(i)})) + \log\frac{D_\alpha(\mathscr{G}_\phi(\epsilon^{(i)};x^{(i)}),x^{(i)})}{1-D_\alpha(\mathscr{G}_\phi(\epsilon^{(i)};x^{(i)}),x^{(i)})}\}$;
end
```
Algorithm 8: Sprinkler Prior-Contrastive Class Probability Estimation

```
Data: Dataset $q^*(x) = \{0, 5, 8, 12, 20\}$, true (implicit) prior
      $p(z) \sim \mathscr{N}(0, 2I_{2\times 2})$, true (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 5001 do
    Sample $\{\epsilon^{(i,j)}\}_{i=1}^{2000} \sim \pi(\epsilon)$;
    Sample $\{0, 5, 8, 12, 50\}_{i=1}^{400} = \{x^{(i,j)}\}_{i=1}^{2000} \sim q^*(x)$;
    Sample $\{z_p^{(i,j)}\}_{i=1}^{2000} \sim p(z)$;
    foreach $\epsilon^{(i,j)}, x^{(i,j)}$ do
        Sample $\{z_q^{(i,j)}\}_{i=1}^{2000} = \mathscr{G}(\epsilon^{(i,j)}; x^{(i,j)})$;
    end
    update $\alpha$ by optimization step on
      $\min_\alpha -\frac{1}{2000}\sum_{i=1}^{2000}\{\log[r_\alpha(\mathscr{G}_\phi(\epsilon^{(i,j)}; x^{(i,j)}), x^{(i,j)})] + r_\alpha(z_p^{(i,j)}, x^{(i,j)})\}$;
end
for $j = 1$ to 10001 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 $\{z_q^{(i,k)}\}_{i=1}^{1000} = \mathscr{G}(\epsilon^{(i,k)}; x_q^{(i,k)})$;
        end
        update $\alpha$ by optimization step on
          $\min_\alpha -\frac{1}{1000}\sum_{i=1}^{1000}\{\log[r_\alpha(\mathscr{G}_\phi(\epsilon^{(i,k)}; x^{(i,k)}), x^{(i,k)})] + r_\alpha(z_p^{(i,k)}, x^{(i,k)})\}$;
    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 $\phi$ by optimization step on
      $\min_\phi \frac{1}{1000}\sum_{i=1}^{1000}\{-\log p(x|\mathscr{G}_\phi(\epsilon^{(i)}; x^{(i)})) + \log r_\alpha(\mathscr{G}_\phi(\epsilon^{(i)}; x^{(i)}), x^{(i)})\}$;
end
```
Algorithm 9: Sprinkler Prior-Contrastive KL Divergence Minimisation

Data: Dataset $q^*(x) = \{0, 5, 8, 12, 20\}$, true (implicit) prior $p(z) \sim \mathcal{N}(0, 2I_{2\times 2})$, true (implicit) likelihood $p(x|z) \sim EXP(3 + \max(0, z_1)^3 + \max(0, z_2)^3)$, noise distribution $\pi(\epsilon) \sim \mathcal{N}(0, I_{3\times 3})$

Result: Optimized posterior generator $\mathcal{G}_\phi(\epsilon; x)$

```
for $j = 1$ to $5001$ 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 $\{z_q^{(i,j)}\}_{i=1}^{1000} = \mathcal{G}(\epsilon^{(i,j)}; x_q^{(i,j})$;
    end
    foreach $z_p^{(i,j)}$ do
        Sample $\{x_p^{(i,j)}\}_{i=1}^{1000} \sim p(x|z)$;
    end
    update $\alpha$ by optimization step on
      $\min_\alpha -\frac{1}{1000}\sum_{i=1}^{1000}\{\log[D_\alpha(\mathcal{G}_\phi(\epsilon^{(i,j)}; x_q^{(i,j)}), x_q^{(i,j)})] + \log[1 - D_\alpha(z_p^{(i,j)}, x_p^{(i,j)})]\}$;
end
for $j = 1$ to $40001$ 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 $\{z_q^{(i,k)}\}_{i=1}^{1000} = \mathcal{G}(\epsilon^{(i,k)}; x_q^{(i,k)})$;
        end
        foreach $z_p^{(i,k)}$ do
            Sample $\{x_p^{(i,k)}\}_{i=1}^{1000} \sim p(x|z)$;
        end
        update $\alpha$ by optimization step on
          $\min_\alpha -\frac{1}{1000}\sum_{i=1}^{1000}\{\log[D_\alpha(\mathcal{G}_\phi(\epsilon^{(i,k)}; x_q^{(i,k)}), x_q^{(i,k)})] + \log[1 - D_\alpha(z_p^{(i,k)}, x_p^{(i,k)})]\}$;
    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 $\phi$ by optimization step on
      $\min_\phi \frac{1}{1000}\sum_{i=1}^{1000} \log \frac{D_\alpha(\mathcal{G}_\phi(\epsilon^{(i)}; x_q^{(i)}), x_q^{(i)})}{1 - D_\alpha(\mathcal{G}_\phi(\epsilon^{(i)}; x_q^{(i)}), x_q^{(i)})}$;
end
```

Algorithm 10: Sprinkler Joint-Contrastive Class Probability Estimation

```
Data: Dataset $q^*(x) = \{0, 5, 8, 12, 20\}$, true (implicit) prior
      $p(z) \sim \mathscr{N}(0, 2I_{2\times2})$, true (implicit) likelihood
      $p(x|z) \sim EXP(3 + \max(0, z_1)^3 + \max(0, z_2)^3)$, noise distribution
      $\pi(\epsilon) \sim \mathscr{N}(0, I_{3\times3})$
Result: Optimized posterior generator $\mathscr{G}_\phi(\epsilon; x)$
for $j = 1$ to 5001 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 $\{z_q^{(i,j)}\}_{i=1}^{1000} = \mathscr{G}(\epsilon^{(i,j)}; x_q^{(i,j})$;
    end
    foreach $z_p^{(i,j)}$ do
        Sample $\{x_p^{(i,j)}\}_{i=1}^{1000} \sim p(x|z)$;
    end
    update $\alpha$ by optimization step on
     $\min_\alpha -\frac{1}{1000}\sum_{i=1}^{1000}\{\log[r_\alpha(\mathscr{G}_\phi(\epsilon^{(i,j)}; x_q^{(i,j)}), x_q^{(i,j)})] + r_\alpha(z_p^{(i,j)}, x_p^{(i,j)})\}$;
end
for $j = 1$ to 40001 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 $\{z_q^{(i,k)}\}_{i=1}^{1000} = \mathscr{G}(\epsilon^{(i,k)}; x_q^{(i,k)})$;
        end
        foreach $z_p^{(i,k)}$ do
            Sample $\{x_p^{(i,k)}\}_{i=1}^{1000} \sim p(x|z)$;
        end
        update $\alpha$ by optimization step on
         $\min_\alpha -\frac{1}{1000}\sum_{i=1}^{1000}\{\log[r_\alpha(\mathscr{G}_\phi(\epsilon^{(i,k)}; x_q^{(i,k)}), x_q^{(i,k)})] + r_\alpha(z_p^{(i,k)}, x_p^{(i,k)})\}$;
    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 $\phi$ by optimization step on
     $\min_\phi \frac{1}{1000}\sum_{i=1}^{1000}\log r_\alpha(\mathscr{G}_\phi(\epsilon^{(i)}; x_q^{(i)}), x_q^{(i)})$;
end
```

Algorithm 11: Sprinkler Joint-Contrastive KL Divergence Minimisation

5.3 Results

We evaluate the techniques by comparing the average KL divergence between the true and variational posterior distributions at the end of the program's runtime: $\mathbb{E}_{q^*(x)} KL(q(z|x)||p(z|x)) = \mathbb{E}_{q^*(x)}\mathbb{E}_{q(z|x)} \log \frac{q(z|x)}{p(z|x)}$. We do not use the ELBO output of the program to calculate this KL divergence, nor do we compare the ELBOs of the algorithms, as the accuracy of the ELBO estimator is dependent on the algorithm used. Instead, we use a non-parametric approach, estimating the probability density function of the variational posterior output with a Gaussian kernel density estimator $\hat{q}(z|x)$. By using five kernel density estimators for the five data points, we are able to create an estimate of the average KL divergence with an accuracy that is independent of the algorithm used: $\mathbb{E}_{q^*(x)} KL(\hat{q}(z|x)||p(z|x)) = \mathbb{E}_{q^*(x)}\mathbb{E}_{q(z|x)} \log \frac{\hat{q}(z|x)}{p(z|x)}$.
Each algorithm was run 30 times and the estimator losses and NELBO estimations for each iteration was stored. Every 100 iterations, kernel density estimation was used to estimate the average KL divergence. The arrays storing the three recorded values were averaged over the 30 repetitions, to enforce reliable experimentation.
Clearly, the prior-contrastive algorithms converged much faster their joint-contrastive equivalent, as expected since knowledge of the likelihood improves posterior optimization. Overall, for both PC and JC, we found that class probability estimation performed consistently better than KL divergence minimisation. The means and standard deviations of the results can be found in Table, and it can be seen that the KL minimisation approach had higher average KL divergence and standard deviation, implying that the method is slower and more inconsistent. The estimator loss, estimated NELBO and 'true' KL Divergence over the iterations can be found in Figures. 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. Each iteration took approximately the same amount of time to compute (0.32 seconds on me PC), as the bulk of the training time is spent training the neural networks, which have similar structures.
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. At higher training rates, the KL minimization algorithm would sometimes enter this "failure" stage during the main optimization stage. The high ratio loss signifies a

very inaccurate ratio estimator, leading to a completely inaccurate variational posterior (i have a fig to isnert here). Thus, the program was programmed to restart every time the ratio loss initialized at a high number, until a reasonable value was initialized.

Chapter 6

Theory Crafting

In an attempt to explain why class probability estimation is significantly superior to KL minimization, we compare the two techniques in terms of their 'lower bound' formulation, and analyze their fundamental differences.

6.1 Introduction

For both the prior-contrastive and joint-contrastive formulations, it can be seen that the only differences between CPE and KL min are the function that is optimized to formulate the density ratio estimator, and the parametrization of the estimator. Recall that optimization of the estimator for the KL min algorithm follows the lower bound:

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

with equality at $r_\alpha(u) = \frac{q(u)}{p(u)}$ and that optimization for class probability estimation follows:

$$2D_{JS}(p||q) - \log 4 \geq \sup_{\alpha}\{\mathbb{E}_{q(u)}[\log \mathscr{D}_\alpha(u)] + \mathbb{E}_{p(u)}[\log(1 - \mathscr{D}_\alpha(u))]\}$$

with equality at $D_\alpha(u) = \frac{q(u)}{q(u)+p(u)}$.
These two estimators can be used interchangeably, as $D_\alpha(u) = \frac{r_\alpha(u)}{r_\alpha(u)+1}$ and $r_\alpha(u) = \frac{D_\alpha(u)}{1-D_\alpha(u)}$. In fact, the former transformation is used to derive this formulation of the discriminative loss for class probability estimation. By transforming the estimators used in the two f-divergences, we can formulate two new estimator optimization problems:

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

and

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

The optimization problem for the ELBO in both prior-contrastive and joint-contrastive methods involves density ratio estimation between the true and variational distributions, and the optimization of the likelihood term in the prior-contrastive method is independent of the ratio estimator. We can therefore denote the two ELBO optimization problems for the two estimator parametrizations $D_\alpha(u)$ and $r_\alpha(u)$ as:

$$\min_\phi \mathbb{E}_{q_\phi(u)} \log \frac{D_\alpha(u)}{1 - D_\alpha(u)} \quad (+\text{ Likelihood})$$

and

$$\min_\phi \mathbb{E}_{q_\phi(u)} \log r_\alpha(u) \quad (+\text{ Likelihood}).$$

Additionally, we can parametrize the estimator such that it estimates the log ratio directly: $T_\alpha(u) = \log r_\alpha(u)$. This simplifies the objective function for our variational neural network weights to:

$$\min_\phi -\mathbb{E}_{q_\phi(u)}[T_\alpha(u)] \quad (+\text{ Likelihood}).$$

For the KL Divergence, the estimator is optimized as follows:

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

The CPE divergence equivalent is:

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

6.2 Recap because its confusing

To recap, we have mathematically simplified the choice of algorithm for training ratio estimators for implicit variational neural networks to a choice of f-divergence:

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

and a choice of estimator parametrization:

- Direct ratio estimator: $r^*_\alpha(u) = \frac{q(u)}{p(u)}$
- "Class probability" estimator: $D^*_\alpha(u) = \frac{q(u)}{q(u)+p(u)}$
- Direct log ratio estimator: $T^*_\alpha(u) = \log \frac{q(u)}{p(u)}$

The original "KL 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.

6.3 First and Second Derivatives of all 6 possible algorithms

To evaluate the effectiveness of gradient descent optimization in these algorithms, we formulate the first and second derivatives of the objective functions. We first take the "Class probability" parametrization and observe its optimization problem for the CPE divergence:

$$\begin{aligned}
f_{CPE}(u) &= -\mathbb{E}_{q(u)}[\log D(u)] - \mathbb{E}_{p(u)}[\log(1 - D(u))] \\
&= -\int_u q(u) \log D(u) du - \int_u p(u) \log(1 - D) du \\
\frac{df}{dD(u)} &= -\frac{q(u)}{D(u)} + \frac{p(u)}{1 - D(u)} \\
\frac{d^2 f}{dD^2(u)} &= \frac{q(u)}{D^2(u)} + \frac{p(u)}{(1 - D(u))^2} \\
&> 0 \quad \forall q(u) \in (0,1), p(u) \in (0,1), D(u) \in (0,1)
\end{aligned}$$

Since the second derivative is always greater than zero, the function is strictly convex and therefore converges linearly with gradient descent.
Now observe the optimization problem for the KL Divergence:

$$\begin{aligned}
f_{KL}(u) &= -\mathbb{E}_{q(u)} \left(\log \frac{D(u)}{1 - D(u)} \right) + \mathbb{E}_{p(u)} \left(\frac{D(u)}{1 - D(u)} \right) \\
&= -\int_u q(u) \left(\log \frac{D(u)}{1 - D(u)} \right) du + \int_u p(u) \left(\frac{D(u)}{1 - D(u)} \right) du \\
\frac{df}{dD(u)} &= -\frac{q(u)}{D(u)} - \frac{q(u)}{1 - D(u)} + \frac{p(u)}{1 - D(u)} + \frac{p(u)D(u)}{(1 - D(u))^2} \\
&= -\frac{q(u)}{D(u)} - \frac{q(u)}{1 - D(u)} + \frac{p(u)}{(1 - D(u))^2} \\
\frac{d^2 f}{dD^2(u)} &= \frac{q(u)}{D^2(u)} - \frac{q(u)}{(1 - D(u))^2} + \frac{2p(u)}{(1 - D(u))^3}
\end{aligned}$$

CPE with ratio parametrisation:

$$f_{CPE}(u) = -\mathbb{E}_{q(u)} \left[\log \frac{r(u)}{r(u) + 1} \right] + \mathbb{E}_{p(u)} \left[\log(r(u) + 1) \right]$$

$$
\begin{aligned}
&= -\int_u q(u)\left[\log\frac{r(u)}{r(u)+1}\right]du + \int_u p(u)\left[\log(r(u)+1)\right]du \\
\frac{df}{dr(u)} &= -\frac{q(u)}{r(u)} + \frac{q(u)}{r(u)+1} + \frac{p(u)}{r(u)+1} \\
\frac{d^2f}{dr^2(u)} &= \frac{q(u)}{r^2(u)} - \frac{q(u)}{(r(u)+1)^2} - \frac{p(u)}{(r(u)+1)^2}
\end{aligned}
$$

KL with ratio parametrisation:

$$
\begin{aligned}
f_{KL}(u) &= -\mathbb{E}_{q(u)}(\log r(u)) + \mathbb{E}_{p(u)}r \\
&= -\int_u q(u)\log r(u)du + \int_u p(u)r(u)du \\
\frac{df}{dr(u)} &= -\frac{q(u)}{r(u)} + p(u) \\
\frac{d^2f}{dr^2(u)} &= \frac{q(u)}{r^2(u)} \\
&> 0
\end{aligned}
$$

CPE with log ratio parametrisation:

$$
\begin{aligned}
f_{CPE}(u) &= -\mathbb{E}_{q(u)}[T(u) - \log(e^{T(u)}+1)] + \mathbb{E}_{p(u)}[\log(e^{T(u)}+1)] \\
&= \int_u q(u)[\log(e^{T(u)}+1) - T(u)]du + \int_u p(u)[\log(e^{T(u)}+1)]du \\
\frac{df}{dT(u)} &= -q(u) + \frac{(q(u)+p(u))e^{T(u)}}{e^{T(u)}+1} \\
\frac{d^2f}{dT^2(u)} &= \frac{(q(u)+p(u))\exp(T(u))}{e^{T(u)}+1} - \frac{(q(u)+p(u))e^{2T(u)}}{(e^{T(u)}+1)^2} \\
&= \frac{(q(u)+p(u))(e^{2T(u)}+e^{T(u)}) - (q(u)+p(u))e^{2T(u)}}{(e^{T(u)}+1)^2} \\
&= \frac{(q(u)+p(u))e^{T(u)}}{(e^{T(u)}+1)^2} \\
&> 0
\end{aligned}
$$

KL with log ratio parametrisation:

$$
\begin{aligned}
f_{KL}(u) &= -\mathbb{E}_{q(u)}[T(u)] + \mathbb{E}_{p(u)}[e^{T(u)}] \\
&= -\int_u q(u)T(u)du + \int_u p(u)e^{T(u)}du \\
\frac{df}{dT(u)} &= -q(u) + p(u)e^{T(u)}
\end{aligned}
$$

$$\begin{aligned}\frac{d^2 f}{dT^2(u)} &= p(u)e^{T(u)} \\ &> 0\end{aligned}$$

6.4 Difference between divergences

Recall that both divergences attain the same global minimum, which is a parametrization of the estimator, and that these estimators are optimized via stochastic gradient descent. Thus, for a fixed estimator parametrization, the divergences will have varying rates of convergence, which can be analyzed by observing the second derivative. The convergence rate of a gradient descent method is proportional to the size of its second derivative. Due to the variable nature of $p(u)$, $q(u)$ and $D(u)/r(u)/T(u)$, it is uncertain whether a divergence is superior in this aspect when its second derivative is not strictly greater than the other divergence's equivalent. We are therefore unable to arrive at a certain conclusion for the "Class probability" and direct log ratio estimators, but it can be seen that in the direct ratio parametrisation, the KL divergence is strictly greater than the CPE divergence, implying that it would be superior. However, due to the significant number of iterations taken for each estimator optimization step, it is unlikely the different divergences have a significant impact on the quality of the method. This can be additionally seen in the non-significant decrease in estimator loss over the later half of its optimization step.
The effectiveness of different divergences has also been tested in (Nowozin, 2016). They found that the ideal f-divergence used for the estimator's loss function is the same f-divergence that is being minimized in the variational posterior training: in this case, it is the KL divergence. This provides support for our theory that the KL divergence is superior (at least in the direct ratio case).

6.5 Difference between estimators

Again, we compare the second derivatives of the techniques, this time fixing the divergence and observing the difference in estimator parametrization. Due to the exponential term in the log ratio parametrisation, no conclusion can be drawn from its comparisons. We cannot conclude anything from comparing the remaining two estimators for the KL divergence either. It can however be seen that for the CPE divergence, the "Class probability" parametrization is strictly superior.
Now consider the bounds on the estimators. The class probability estimator with an optimal form of $\frac{q(u)}{q(u)+p(u)}$ 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 is bound in $\mathbb{R}^+ \ \{0\}$, and the direct log ratio estimator 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.

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. From the bounds, it is intuitive that the class probability estimator's global minimum displaces less than the direct ratio estimator. It can also be shown that $|D^*_{final} - D^*_{init}| < |r^*_{final} - r^*_{init}|$:

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

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

If $\epsilon > 0$, then

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

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

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

For the direct log ratio estimator, we have

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

It is difficult to make a direct comparison with the other displacement expressions. We now address the "failures" experienced by the KL minimization algorithm. Consider that the activation function of the estimator network's output layer, influencing the output range, varies between the estimators. For the direct ratio estimator, ReLU is used for the output, meaning that a 0 is outputted if the previous matrix operations lead to a negative number. Now recall that we take the log of the ratio estimator's output in all of the related optimization problems, and that a small constant c is added to prevent taking the log of 0. This explains the 'failures' of the KL minimization algorithm: the neural network has been initialised such that a 0 is outputted, signifying that the previous matrix operations led to a negative number. Its loss function takes a large value according to this scenario (e.g. for KL minimization, it is $-\log c$), and does not decrease over training iterations, as a slight change in the network weights would still output the same result, meaning that the partial derivative of the weights with respect to the loss function is 0. Due to this occurrence, the "failures" would additionally occur in the direct log ratio estimator, as its output layer is simply the log of a ReLU output, which would suffer from the same problem. On the other hand, the sigmoid activation function of the "Class probability" estimator is bound in $(0, 1)$, so it does not experience any "failures".
Aside from the failures, the difference in output activation functions could also explain our experimental results; the rectified linear unit models the density ratio in a linear manner, meaning that it would be asymmetric, in the sense that cases where $q(u) > p(u)$ result in any output in $(1, \infty)$, but if $p(u) < q(u)$, then the outputted is limited to $(0, 1)$. On the other hand, the sigmoid function has symmetric output,

as if $q(u) > p(u)$, it is bound in $(0.5, 1)$, and if $q(u) < p(u)$, the outputs lie within $(0, 0.5)$. Taking the log of the rectified linear output and optimizing the objective functions with respect to it may partially resolve this issue, as the cases will become bound in $(0, \infty)$ and $(-\infty, 0)$ respectively, but it would retain an additional issue: having a rectified linear unit for the output layer restricts the previous matrix operations to have an output in $\mathbb{R}^+ \backslash \{0\}$, as its surjective mapping truncates $\mathbb{R}^-$. On the other hand, the sigmoid output layer has a bijective mapping from $\mathbb{R}$ to $(0, 1)$. Instead of using the naive ReLU activation function, we propose to use an exponential activation function for the direct ratio estimator, and a linear activation function for the log ratio estimator. The exponential function is even and bijective, mapping $\mathbb{R}^-$ to $(0, 1)$ and $\mathbb{R}^+$ to $(1, \infty)$. Similarly, the linear activation keeps the output in $\mathbb{R}$, which is the space of the log ratio estimator. We believe that this will resolve the key issues in our KL minimization approach: the estimator outputs will be more consistent throughout the early training period and there will be no failures caused by an output of 0.

Chapter 7

Inference Experiments - "Sprinkler"

7.1 Determining optimal activation function

7.1.1 Experiment Outline

Since we have identified two possible representations for both the direct density ratio and log density ratio estimations, we first compare them to find the ideal activation function. Additionally, if our proposed activation functions prevent the "failures" from occurring, then the posterior training rate can be increased for the following experiments, reducing the computational time required to complete them. The experiment setup is exactly the same as the previous experiment, with the same repetitions, training rates and iterations, but we now aim to compare the ReLU and exponential activation functions for direct density ratio estimation, as well as the log ReLU and linear activation functions for direct log density ratio estimation. These comparisons will be made in four different contexts: both the CPE and KL divergences will be used in both of the prior-contrastive and joint-contrastive environments.

7.1.2 Results

Firstly, we note that the bijective activation functions did not experience any failures. This alone confirms our theory as to why the failures occur, and that the surjective activation functions are inferior for this particular purpose. Comparing the simulations that haven't failed, we have the following tables and figures showing the results. It can be seen that the means and standard deviations of the true KL divergence are consistently lower for the exponential and linear activation functions. This is due to the lack of fluctuations in their estimator losses and NELBO estimations, supporting our theory that the unevenness of the ReLU activation function correlates to inconsistent training and outputs.

7.2 Comparing Estimator Accuracies

7.2.1 Experiment Outline

Now that we have identified optimal activation functions for the estimators, we now verify that the estimators, when optimally trained, lead to similar levels of convergence, regardless of the divergence used or the estimator parametrisation. Again, the setup is the same as in the previous two experiments, with low training rates and high iterations to ensure smooth posterior convergence and optimality of the estimator. Note that at this point, we already have experimental results for all of the divergence and estimator combinations except for the KL divergence with class probability estimator. Thus, to save computational time, we reuse those results.

7.2.2 Results

From the figures below, we can see that for each divergence, the graphs of the three estimators are indistinguishable. The final KL divergences in the table below also shows no significant difference between the divergences or estimator parametrizations used. The key point of this result is that it does not matter which estimator training combination is used as long as it is optimally trained.

7.3 Comparing Accuracies of Undertrained Estimators

7.3.1 Experiment Outline

In this experiment, we aim to confirm our theory that the estimators only differ when improperly trained, and that an estimator's accuracy directly correlates with its training rate. We therefore significantly reduce the amount of estimator training between each posterior iteration, lowering the estimator training rate to 0.00004 and the estimator steps to 15 in the prior-contrastive setting, and 20 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 was reduced to 2000 for prior-contrastive and 4000 for joint-contrastive. The other training parameters were kept consistent with the previous experiments.

7.3.2 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.

Chapter 8

Data Generation - (MNIST image generation)

8.1 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.

8.2 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.

8.3 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 9

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 10

Conclusion

This thesis is hot trash.

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.