Contents

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 $X = \{x^{(i)}\}_{i=1}^{N}$ 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 parametrized by θ $p_\theta(z|x)$ 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 $p(x|z)$. 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 $p_\theta(z|x) = \frac{p(z)p(x|z)}{p(x)} = \frac{p(x|z)p(z)}{\int_z p(x,z)dz}$, 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

1.2 Objective Derivation

1.2.1 Implicit Prior

The posterior distribution cannot be computed efficiently if $\int_z p(x,z)dz$ is intractable, which is often the case with large datasets or high dimensional data. Furthermore, the prior $p(z)$ or the likelihood $p(x|z)$ may be implicit. In the framework of variational inference, we approximate the posterior distribution $p_\theta(z|x)$ with a variational distribution parametrized by ϕ: $q_\phi(z)$. We aim to minimize the KL divergence between these distributions, defined as

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

The parameters ϕ of our approximate distribution $q_\phi(z)$ are therefore chosen such that the KL divergence is minimised. Since the true posterior distribution $p_\theta(z|x)$ is dependent on x, it differs with each observation x_n that we take. This would mean we would have to define a variational distribution for each observation. Thus, we amortize our distribution, conditioning it on x and letting its parameters ϕ be constant for each observation:(or better to say condition on x and take expectation w.r.t. $q^*(x)$?)

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

We therefore 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 what is called the ELBO (Evidence Lower Bound), which we want to maximise to minimise the KL divergence.

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

If the prior distribution is implicit (but the likelihood $\log p(x|z)$ isn't), then we can still evaluate and optimize this expression by considering the density ratio between $p(z)$ and $q_\phi(z|x)$:

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

Now we specify our variational distribution $q_\phi(z|x)$ to be a neural network $\mathcal{G}$ parametrized by ϕ that inputs observed variable $x \sim q^*(x)$ and outputs $z \sim q_\phi(z|x)$. The problem with this approach is that neural networks are deterministic and do not capture the probabilistic nature of a distribution. The traditional method of solving this issue is to apply the "reparametrization trick" (Kingma 2014), in which the neural network is instead trained to output the mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\sigma^2}$ vectors of the posterior distribution. These are used to transform a noise variable $\epsilon \sim \pi(\epsilon)$ (typically $\pi(\epsilon) = \mathcal{N}(0, I_{n\times n})$), overall depicting the posterior as a multivariate normal distribution with 0 covariance:

$$\boldsymbol{z} = \boldsymbol{\mu} + \boldsymbol{\sigma^2} \times \boldsymbol{\epsilon},$$

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

In this paper, we instead add noise variables $\epsilon \sim \pi(\epsilon)$ as inputs to the neural network, along with x. (Mescheder 2017) This allows for greater flexibility in our model at the cost of having to do backpropogation over the sampling process and loss of information about our variational posterior parameters ϕ, as it is now an implicit 'black-box'. We therefore denote the generator of the variational posterior samples as $\mathcal{G}_\phi(\epsilon; x)$. **(should insert two comparing dags to highlight difference in methods)**
The ELBO therefore becomes

$$ELBO = \mathbb{E}_{q^*(x)\pi(\epsilon)}[\log p_\theta(x|\mathcal{G}_\phi(\epsilon; x)] + \mathbb{E}_{q^*(x)q_\phi(z|x)}\left[\log \frac{p_\theta(z)}{q_\phi(z|x)}\right].$$

Our objective is to maximize the ELBO, equivalent to minimizing its negative:

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

As previously stated, since the prior distribution is implicit, we use a ratio estimator $\hat{r}_\alpha(x)$ for the true density ratio $r^*(x) = \frac{q_\phi(z|x)}{p_\theta(z)}$. The goal of the density ratio estimation procedure (section ...) is to formulate and optimize an estimator for this density ratio based on samples from the distributions.

1.2.2 Implicit Likelihood (& Prior)

Clearly, if our likelihood distribution is implicit, then this expression cannot be evaluated. Instead, we must rephrase the problem, considering a density ratio between two joint distributions (Tran, 2017). We begin by restating the original objective of choosing parameters ϕ to minimize the KL divergence between two distributions:

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

We therefore have the expression for the 'evidence lower bound' (not actually an evidence lower bound but we keep the name for consistency:

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

Our objective now is to minimize this ELBO with respect to our variational parameters ϕ

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

There are two main methods of estimating and optimizing the density ratio and subsequently, the ELBO: Class Probability Estimation and Divergence Minimisation. The goal of this thesis is to implement and compare these two methods (and hopefully ratio matching) over a variety of models (Hierarchical Lotka-Volterra, Image MNIST etc.) and conditions (implicit prior or likelihood), evaluating their rates of convergence, bias and mean squared error to determine which method is best in which conditions.

1.3 Class Probability Estimation

1.3.1 Procedure (need better heading)

Firstly, consider the problem of estimating the density ratio between two arbitrary distributions $q(x)$ and $p(x)$: $\frac{q(x)}{p(x)}$. We take m samples from $p(x)$: $X_p = \{x_1^{(p)}, \dots, x_m^{(p)}\}$ and label them with $y = 0$, then we take n samples from $q(x)$: $X_q = \{x_1^{(q)}, \dots, x_n^{(q)}\}$ and label them with $y = 1$. Therefore, $p(x) = P(x|y = 0)$ and $q(x) = P(x|y = 1)$. By applying Bayes' theorem, we derive an expression for the density ratio:

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

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

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

which is the ratio of the probability that an arbitrary sample x was taken from the distribution $q(x)$ to the probability that is was taken from $p(x)$. If we define a discriminator function that finds these probabilities

$$D(x) = P(y = 1|x),$$

then our density ratio can be expressed in terms of this discriminator function:

$$\frac{q(x)}{p(x)} = \frac{D(x)}{1 - D(x)}.$$

1.3.2 Implicit Prior

In class probability estimation, we use a discriminator function $D_\phi(z)$ denoted by a neural network that calculates the probability that a given sample z is from the variational posterior distribution $q_\phi(z|x)$ as opposed to the true prior distribution $p^*(x)$. The ratio estimator can be expressed in terms of the probability that a sample from the posterior distribution correctly classified by the discriminator. Optimization involves optimizing the discriminator by minimizing the discriminative loss (a Bernoulli loss dependent on the discriminators classification accuracy), then minimizing the evidence lower bound, and cycling between these two steps until convergence.

We now turn to our problem of minimizing the negative ELBO:

$$\min_\phi -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log p_\theta(x|\mathcal{G}_\phi(\epsilon;x)] + \mathbb{E}_{q^*(x)q_\phi(z|x)}\left[\log\frac{q_\phi(z|x)}{p_\theta(z)}\right],$$

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 in the form of a neural network with parameters α 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_n :

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

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

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

We can express the expected log ratio estimator $\mathbb{E}_{q^*(x)q_\phi(z|x)}[\log\hat{r}(x)] \simeq \mathbb{E}_{q^*(x)q_\phi(z|x)}[\log\frac{q_\phi(z|x)}{p_\theta(z)}]$ in terms of the probability that a posterior sample is correctly classified by the discriminator:

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

Our optimization objective of minimizing negative ELBO can now be written as:

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

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 take the expressions:

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

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

```
Data: Dataset $q^*(x)$, true (implicit) prior $p(z)$, true likelihood $p(x|z)$, noise distribution $\pi(\epsilon)$
Result: Optimized posterior generator $\mathcal{G}_\phi(\epsilon; x)$
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 $\{z_q^{(i,k)}\}_{i=1}^B = \mathcal{G}(\epsilon^{(i,k)}; x^{(i,k)})$;
        end
        update $\alpha$ by optimization step on
         $\min_\alpha \mathbb{E}_{q^*(x)\pi(\epsilon)}[-\log D_\alpha(\mathcal{G}_\phi(\epsilon; x), x)] + \mathbb{E}_{p_\theta(z)q^*(x)}[-\log(1 - D_\alpha(z, x))]$;
    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
     $\min_\phi -\mathbb{E}_{q^*(x)\pi(\epsilon)}[\log p_\theta(x|\mathcal{G}_\phi(\epsilon; x))] + \mathbb{E}_{q^*(x)\pi(\epsilon)}\left[\log \frac{D_\alpha(\mathcal{G}_\phi(\epsilon;x),x)}{1-D_\alpha(\mathcal{G}_\phi(\epsilon;x),x)}\right]$;
end
```

Algorithm 1: Implicit Prior Class Probability Estimation

1.3.3 Implicit Likelihood (& Prior)

As previously stated, if the likelihood distribution is implicit, we have the optimization objective of:

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

or

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

Like in the implicit prior 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 parametrized by α to determine the probability that samples (z,x) came from the joint variational distribution $q(z,x)$:

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

Using this discriminator function, our density ratio expression becomes:

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

Overall, the class probability algorithm is (again) a cycle of optimizing the discriminator until convergence by minimizing its Bernoulli loss:

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

and taking an optimization step of the variational posterior:

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

Since our posterior is defined as a generative neural network model $\mathcal{G}_\phi(\epsilon;x)$, these optimization objectives become:

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

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

Note that the prior and likelihood terms don't appear in the ELBO training objective, so they can both be implicit. Another difference with the implicit prior algorithm is that instead of sampling x from the dataset, it is taken from the likelihood, conditional on the z sample.

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

Algorithm 2: Implicit Likelihood Class Probability Estimation

1.4 Divergence Minimisation

1.4.1 Procedure (Again need better heading)

First, we define the f-divergence of continuous probability distributions q from p as

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

where f is a convex function such that $f(1) = 0$.
A lower bound for the f-divergence in terms of a ratio estimator $\hat{r}(u) \simeq \frac{q(u)}{p(u)}$ can be found using the following theorem (Nguyen et al. 2010).
Theorem 1. If f is a convex function with derivative f' and convex conjugate f^*, and $\mathcal{R}$ 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 $q(u)$:

$$\mathcal{D}_f[p(u)||q(u)] \geq \sup_{\hat{r}\in\mathcal{R}}\{\mathbb{E}_{q(u)}[f'(\hat{r}(u))] - \mathbb{E}_{p(u)}[f^*(f'(\hat{r}(u)))]\}$$

with equality when $\hat{r}(u) = q(u)/p(u)$.

For the 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 $f(u) = u \log u$ is

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

so the convex conjugate of the derivative is simply

$$f^*(f'(u)) = u.$$

Using Theorem 1, we derive the following lower bound for the KL divergence:

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

1.4.2 Implicit Prior

In divergence minimisation, we convert the expected density ratio expression into a KL divergence, and use a theorem (name of theorem?) to find a lower bound of the divergence in terms of a ratio estimator denoted by a neural network. We then perform bi-level optimization of maximizing the KL divergence lower bound and minimizing the negative evidence lower bound.

Firstly, note that maximising the ELBO is equivalent to minimising its negative, so our overall objective is

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

or

$$\min_{\phi} -\mathbb{E}_{q^*(x)\pi(\epsilon)}\left[\log p(x|\mathcal{G}_\phi(\epsilon;x)\right] + \mathbb{E}_{q^*(x)} KL[q_\phi(z|x)||p(z)]$$

Denoting the true density ratio $\frac{q_\phi(z|x)}{p(z)}$ as $r^*(z)$, we aim to find an estimator $\hat{r}(z)$.

We apply the result from the previous section to find a lower bound for our KL divergence:

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

Now we let our 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 consider the expectation across $q^*(x)$:

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

To optimize our ratio estimator, we fix ϕ and optimize α such that the lower bound is maximised, reducing the gap between the lower bound and the true KL divergence.

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

We remove the $+1$ term as it is independent of α, and use the generator form of the posterior to rewrite this optimization objective as

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

Simultaneously, we want to minimise our overall objective, which is the negative evidence lower bound. Noting that $\mathbb{E}_{q^*(x)} KL[q_\phi(z|x)||p(z)] \simeq E_{q^*(x)q_\phi(z|x)}[\log r_\alpha(z;x)]$, accomplish this by fixing α and optimizing ϕ to minimize the lower bound:

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

Standardizing our maximization objective of the ratio estimator to a minimization goal and using the generator form of the posterior, we therefore have the bi-level optimization problem:

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

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

Similar to class probability estimation, the algorithm involves cycling between optimizing the ratio estimator until convergence and taking one gradient step of ELBO minimisation.
Note that this estimator is naturally biased as it follows a lower bound.
(Also note that $f(u) = u\log u - (u+1)\log(u+1)$ leads to class probability estimation algorithm)

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

Algorithm 3: Implicit Prior Divergence Minimisation

1.4.3 Implicit Likelihood (& Prior)

First, we restate the implicit likelihood problem of minimizing the following density ratio

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

This expression can be written as a KL divergence as follows:

$$\begin{aligned}\min_{\phi} \mathbb{E}_{q^*(x)q_\phi(z|x)} \log \frac{q(z,x)}{p(z,x)} &= \min_{\phi} \mathbb{E}_{q(z,x)} \log \frac{q(z,x)}{p(z,x)} \\ &= \min_{\phi} KL[q(z,x)||p(z,x)]\end{aligned}$$

Now our ratio estimator approximates the joint density ratio:

$$\hat{r}(z,x) \simeq \frac{q(z,x)}{p(z,x)}$$

Applying Theorem 1, a lower bound for our KL divergence is

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

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 our KL divergence lower bound becomes

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

Like before, we want to maximize this lower bound with respect to α, so our ratio estimator optimization objective is

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

Note we have expanded out the joint distributions and removed the constant $+1$ term.
We also consider our original objective of minimizing the joint density ratio, writing it in terms of the ratio estimator:

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

Finally, we derive the bi-level optimization problem by writing the posterior in terms of its generator form and converting the maximisation problem to a minimization expression:

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

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

Again the algorithm involves cycling between optimizing the ratio estimator until convergence and taking an optimization step of the ELBO.

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

Algorithm 4: Implicit Likelihood Divergence Minimisation

2 Learning

2.1 Variational Inference

2.1.1 Context

In Bayesian statistics, a common problem is to estimate posterior densities, so that we may perform inference to determine an unknown parameter. Consider a set of unknown, latent variables $\mathbf{Z} = \{z_i\}_{i=1}^{M}$ and a dataset of known variables $\mathbf{X} = \{x_i\}_{i=1}^{N}$. These sets have a joint density of $P(\mathbf{Z}, \mathbf{X})$. In the Bayesian framework, inference is often performed on the posterior density (the distribution of the parameters $\mathbf{Z}$ after the data $\mathbf{X}$ is observed) $P(\mathbf{Z}|\mathbf{X})$, which, after applying Bayes' theorem, can be written as:

$$P(\mathbf{Z}|\mathbf{X}) = \frac{P(\mathbf{Z})P(\mathbf{X}|\mathbf{Z})}{P(\mathbf{X})} = \frac{P(\mathbf{Z})P(\mathbf{X}|\mathbf{Z})}{\int_{\mathcal{Z}} P(\mathbf{Z}, \mathbf{X})d\mathbf{Z}}$$

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 $\mathbf{Z}$,
- $P(\mathbf{X}) = \int_{\mathcal{Z}} P(\mathbf{Z}, \mathbf{X})d\mathbf{Z}$ is the marginal likelihood, or the evidence: the density of the data averaged across all possible parameter values.

If the evidence integral $P(\mathbf{X}) = \int_{\mathcal{Z}} P(\mathbf{Z}, \mathbf{X})d\mathbf{Z}$ is impossible or difficult to compute (possibly because it is unavailable in closed form or the dimensionality is too high), then we are unable to evaluate the posterior density. Traditionally, MCMC(Markov Chain Monte Carlo) methods overcome this obstacle by constructing a Markov chain that converges to the stationary distribution $P(\mathbf{Z}|\mathbf{X})$, then sampling from the chain to create an empirical estimate for the posterior distribution. However, these methods rely on the speed of convergence, which can be slow 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.

2.1.2 Introduction to Variational Inference

Variational inference chooses another distribution $Q(\mathbf{Z})$ from a select family of variational distributions (approximate densities) $\mathcal{Q}$ to serve as an approximation to $P(\mathbf{Z}|\mathbf{X})$, and then minimizes the divergence between the two distributions in an optimization problem:

$$Q^*(\mathbf{Z}) = \underset{Q(\mathbf{Z}) \in \mathcal{Q}}{\arg\min}\, D(Q(\mathbf{Z})||P(\mathbf{Z}|\mathbf{X})) \tag{2.1}$$

where D denotes an f-divergence (a measure of how divergent two probability distributions are, it is minimized if $Q = P$). This results in an analytical approximation to the posterior density. Additionally, a lower bound for the marginal likelihood of the dataset is derived, which can be used as a model selection criterion. Due to the stochastic nature of the optimization, variational inference methods can be much faster than MCMC, but the solution is only locally optimal as there is no guarantee of global convergence.

2.1.3 Derivation of the ELBO

The most common f-divergence used in variational inference is the KL(Kullback-Leibler) divergence, defined as the expected logarithmic difference between two distrbutions Q and P with respect to Q:

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

Note that the KL divergence is not symmetric. We use the reverse KL divergence instead of the forward KL divergence $KL(P||Q)$ because it leads to an expectation maximization algorithm as opposed to an expectation propagation algorithm.
Using this expression, we can rewrite equation (1) as:

$$\begin{aligned}
Q^*(\mathbf{Z}) &= \arg\min_{Q(\mathbf{Z}) \in \mathcal{Q}} KL(Q(\mathbf{Z})||P(\mathbf{Z}|\mathbf{X})) \\
&= \arg\min_{Q(\mathbf{Z}) \in \mathcal{Q}} \mathbb{E}_{Q(\mathbf{Z})}[\log Q(\mathbf{Z}) - \log P(\mathbf{Z}|\mathbf{X})] \\
&= \arg\min_{Q(\mathbf{Z}) \in \mathcal{Q}} \mathbb{E}_{Q(\mathbf{Z})} \left[\log Q(\mathbf{Z}) - \log \frac{P(\mathbf{X}|\mathbf{Z})P(\mathbf{Z})}{P(\mathbf{X})} \right] \\
&= \arg\min_{Q(\mathbf{Z}) \in \mathcal{Q}} \left(\mathbb{E}_{Q(\mathbf{Z})}[\log Q(\mathbf{Z}) - \log P(\mathbf{X}|\mathbf{Z}) - \log P(\mathbf{Z})] + \log P(\mathbf{X}) \right).
\end{aligned}$$

Note in the last line $\mathbb{E}_{Q(\mathbf{Z})}[P(\mathbf{X})] = P(\mathbf{X})$ as it is not dependent on $Q(\mathbf{Z})$. Also note that the KL divergence is dependent on $P(\mathbf{X})$, which we have determined to be intractable, so this optimization problem cannot be solved in this form. This issue is resolved by rearranging the KL divergence expression as follows:

$$\begin{aligned}
KL(Q(\mathbf{Z})||P(\mathbf{Z}|\mathbf{X})) &= \mathbb{E}_{Q(\mathbf{Z})}[\log Q(\mathbf{Z}) - \log P(\mathbf{X}|\mathbf{Z}) - \log P(\mathbf{Z})] + \log P(\mathbf{X}) \\
\log P(\mathbf{X}) - KL(Q(\mathbf{Z})||P(\mathbf{Z}|\mathbf{X})) &= -\mathbb{E}_{Q(\mathbf{Z})}[\log Q(\mathbf{Z}) - \log P(\mathbf{X}|\mathbf{Z}) - \log P(\mathbf{Z})] \\
&= \mathbb{E}_{Q(\mathbf{Z})}[\log P(\mathbf{X}|\mathbf{Z})] - \mathbb{E}_{Q(\mathbf{Z})}[\log Q(\mathbf{Z}) - \log P(\mathbf{Z})] \\
&= \mathbb{E}_{Q(\mathbf{Z})}[\log P(\mathbf{X}|\mathbf{Z})] - KL(Q(\mathbf{Z})||P(\mathbf{Z})). \qquad (2.2)
\end{aligned}$$

We refer to $\log P(\mathbf{X}) - KL(Q(\mathbf{Z})||P(\mathbf{Z}|\mathbf{X}))$ as $ELBO(Q)$ (evidence lower bound), as it is equal to the marginal probability of the data subtracted by a constant error term. This error term $KL(Q(\mathbf{Z})||P(\mathbf{Z}|\mathbf{X}))$ becomes 0 when $Q(\mathbf{Z}) = P(\mathbf{Z}|\mathbf{X})$, maximizing the ELBO. Note that since $P(\mathbf{X})$ is constant, maximizing the $ELBO$ is equal to minimizing the KL divergence between $Q(\mathbf{Z})$ and $P(\mathbf{Z}|\mathbf{X})$, and that the expression on line (2) is entirely computable. We can therefore rewrite our optimization problem from equation (1) as:

$$\begin{aligned}
Q^*(\mathbf{Z}) &= \arg\min_{Q(\mathbf{Z}) \in \mathcal{Q}} D(Q(\mathbf{Z})||P(\mathbf{Z}|\mathbf{X})) \\
&= \arg\max_{Q(\mathbf{Z}) \in \mathcal{Q}} ELBO(Q) \\
&= \arg\max_{Q(\mathbf{Z}) \in \mathcal{Q}} \left(\mathbb{E}_{Q(\mathbf{Z})}[\log P(\mathbf{X}|\mathbf{Z})] - KL(Q(\mathbf{Z})||P(\mathbf{Z})) \right).
\end{aligned}$$

2.1.4 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 $\{z_i\}_{i=1}^{M}$:

$$Q(\mathbf{Z}) = \prod_{i=1}^{M} q_i(z_i). \qquad (2.3)$$

The individual factors $q_i(z_i)$ can take any form. We want to choose these factors so that $ELBO(Q)$ is maximized. To derive an expression for the optimal factor $q_i^*(z_i)$, we substitute equation (3) 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:

$$\begin{aligned}
ELBO(Q) &= \mathbb{E}_{Q(Z)}[\log P(\mathbf{X}|\mathbf{Z})] - KL(Q(\mathbf{Z})||P(\mathbf{Z})) \\
&= \mathbb{E}_{Q(Z)}[\log P(\mathbf{X}|\mathbf{Z}) + \log P(\mathbf{Z}) - \log Q(\mathbf{Z})] \\
&= \mathbb{E}_{Q(Z)}[\log P(\mathbf{X},\mathbf{Z}) - \log Q(\mathbf{Z})] \\
&= \int_{\mathcal{Z}} Q(\mathbf{Z})(\log P(\mathbf{X},\mathbf{Z}) - \log Q(\mathbf{Z}))d\mathbf{Z}.
\end{aligned}$$

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

$$\begin{aligned}
ELBO(Q) &= \int_{\mathcal{Z}} \left[\prod_{i=1}^{M} q_i(z_i)\right] \left(\log P(\mathbf{X},\mathbf{Z}) - \sum_{i=1}^{M} \log q_i(z_i)\right) d\mathbf{Z} \\
&= \int_{\ddagger_|} q_j(z_j) \left(\int_{\ddagger_{-|}} \log P(\mathbf{X},\mathbf{Z}) \prod_{i\neq j} q_i(z_i) d\mathbf{z}_{-j}\right) dz_j \\
&\quad - \int_{\ddagger_|} q_j(z_j) \left(\int_{\ddagger_{-j}} \left[\prod_{i\neq j} q_i(z_i)\right] \sum_{i=1}^{M} q_i(z_i) d\mathbf{z}_{-j}\right) dz_j \\
&= \int_{\ddagger_|} q_j(z_j)\mathbb{E}_{\mathbf{z}_{-j}}[\log P(\mathbf{X},\mathbf{Z})]dz_j - \int_{\ddagger_|} q_j(z_j) \log q_j(z_j) \left(\int_{\ddagger_{-j}} \prod_{i\neq j} q_i(z_i) dz_{-j}\right) dz_j \\
&\quad - \int_{\ddagger_|} q_j(z_j) \left(\int_{\ddagger_{-j}} \left[\prod_{i\neq j} q_i(z_i)\right] \sum_{i\neq j} q_i(z_i) d\mathbf{z}_{-j}\right) dz_j \\
&= \int_{\ddagger_|} q_j(z_j)\mathbb{E}_{\mathbf{z}_{-j}}[\log P(\mathbf{X},\mathbf{Z})]dz_j - \int_{\ddagger_|} q_j(z_j) \log q_j(z_j) dz_j \\
&\quad - \int_{\ddagger_{-j}} \left[\prod_{i\neq j} q_i(z_i)\right] \sum_{i\neq j} q_i(z_i) d\mathbf{z}_{-j} \qquad (2.4) \\
&= \int_{\ddagger_|} q_j(z_j) \left(\mathbb{E}_{\ddagger_{-j}}[\log P(\mathbf{X},\mathbf{Z})] - \log q_j(z_j)\right) dz_j + \text{const.} \qquad (2.5)
\end{aligned}$$

The term in line (4) becomes a constant as it does not depend on $q_j(z_j)$. Now our Lagrangian equation with the constraint that $q_i(z_i)$ are probability density functions is:

$$ELBO(Q) - \sum_{i=1}^{M} \lambda_i \int_{\ddagger_\rangle} q_i(z_i)dz_i = 0$$

or using our expression for $ELBO(Q)$ in line (5),

$$\int_{\ddagger_|} q_j(z_j) \left(\mathbb{E}_{\ddagger_{-j}}[\log P(\mathbf{X},\mathbf{Z})] - \log q_j(z_j)\right) dz_j - \sum_{i=1}^{M} \lambda_i \int_{\ddagger_\rangle} q_i(z_i)dz_i + \text{const} = 0. \qquad (2.6)$$

Using the Euler-Lagrange equation (need to put this in), we then take the functional derivative of (6) with respect to $q_j(z_j)$ (in this case, the partial derivative with respect to $q_j(z_j)$ of the expression inside the integral):

$$\begin{aligned}\frac{\partial ELBO(q)}{\partial q_j(z_j)} &= \frac{\partial}{\partial q_j(z_j)}\left[q_j(z_j)\left(\mathbb{E}_{\mathbf{z}_{-j}}[\log P(\mathbf{X},\mathbf{Z})] - \log q_j(z_j)\right) - \lambda_j q_j(z_j)\right] \\ &= \mathbb{E}_{\mathbf{z}_{-j}}[\log P(\mathbf{X},\mathbf{Z})] - \log q_j(z_j) - 1 - \lambda_j \end{aligned} \tag{2.7}$$

Equating expression (7) to 0 and letting $1+\lambda_j$ be a constant (as it is independent of z), we have:

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

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

$$\begin{aligned}q_j^*(z_j) &\propto \exp\left(\mathbb{E}_{\mathbf{z}_{-j}}[\log P(\mathbf{X},\mathbf{Z})]\right) \\ &\propto \exp\left(\mathbb{E}_{\mathbf{z}_{-j}}[\log P(z_j|\mathbf{z}_{-j},\mathbf{X})]\right).\end{aligned} \tag{2.8}$$

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 1):

```
Data: Dataset $\mathbf{X}$ and Model P($\mathbf{X}, \mathbf{Z}$)
Result: Approximate 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 $q_j(z_j) \propto \exp(\mathbb{E}[\log P(z_j|\mathbf{z}_{-j}, \mathbf{X})])$;
        end
        Calculate $ELBO(Q) = \mathbb{E}[\log P(\mathbf{Z},\mathbf{X})] - \mathbb{E}[\log Q(\mathbf{Z})]$;
    end
    Return $Q(\mathbf{Z})$;
end
```

Algorithm 5: Coordinate Ascent Variational Inference (CAVI)

2.1.5 Example: Bayesian mixture of Gaussians

To illustrate the variational inference approach, we will use the Bayesian mixture of Gaussians example from (Blei, 2018/16 idk).
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 $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_K)^\top$. 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

$$p(\boldsymbol{\mu}, \mathbf{c}, \mathbf{x}) = p(\boldsymbol{\mu}) \prod_{i=1}^{n} p(c_i) p(x_i | c_i, \boldsymbol{\mu}). \tag{2.9}$$

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 (9), we can derive the evidence lower bound as a function of the variational parameters:

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

From equation (8), we derive the optimal categorical factor by only considering terms from the true distribution $p(.)$ dependent on c_i:

$$q^*(c_i; \boldsymbol{\phi}_i) \propto \exp\left(\log p(c_i) + \mathbb{E}[\log p(x_i | c_i, \boldsymbol{\mu}); \mathbf{m}, \mathbf{s}^2] \right). \tag{2.10}$$

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 (10):

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

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 (8) 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 2), 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 6: CAVI Algorithm for Bayesian mixture of Gaussians

2.1.6 References

To be organised properly and moved to the end later:
$https://en.wikipedia.org/wiki/Variational_Bayesian_methods$
http://bjlkeng.github.io/posts/variational-bayes-and-the-mean-field-approximation/
https://arxiv.org/pdf/1601.00670.pdf
Pattern recognition and machine learning by Bishop (2006) pages 461-dunno
https://www.cs.cmu.edu/ epxing/Class/10708-17/notes-17/10708-scribe-lecture13.pdf
http://dept.stat.lsa.umich.edu/ xuanlong/Papers/Nguyen-Wainwright-Jordan-10.pdf

2.2 Neural Networks

2.2.1 Motivation

Originally, neural networks were an attempt creating 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 computational power became sufficient for most applications at the time.

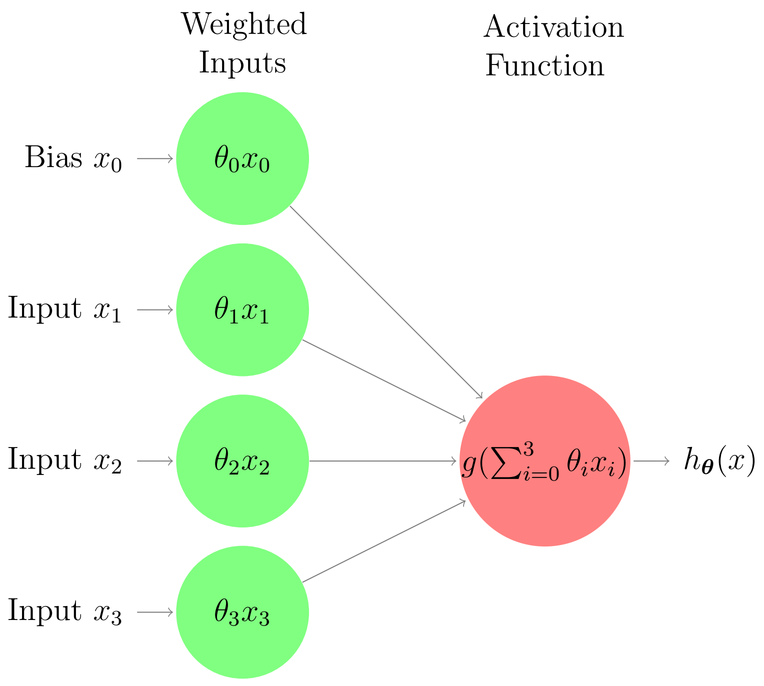
One key feature of the brain structure is the capability for the neurons to adapt to suit any purpose. Neuroscientists have conducted experiments on animals where they rewired the optic nerve from the eye to the auditory cortex. They found that the auditory cortex eventually adapted to process the visual 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.

It can be deduced that each neuron has a similar structure, regardless of its location in the brain, in which inputs in the form of electrical signals are changed in some way and outputted to other neurons. Furthermore, a network of neurons is capable of processing almost any input electrical signal in almost any way. These are the core principles behind neural networks.

2.2.2 Neural Network Structure

The primary goal of a neural network is to approximate some function $f^*(\mathbf{x})$ using a mapping with parameters $\boldsymbol{\theta}$ from the input $\mathbf{x}$ to the output $\mathbf{y}$: $\mathbf{y} = \mathbf{f}_{\boldsymbol{\theta}}(\mathbf{x})$. In fact, it is known that neural networks can approximate any function (universal approximation theorem). 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 recognizing 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. Another function would be used to select the digit with the highest probability.

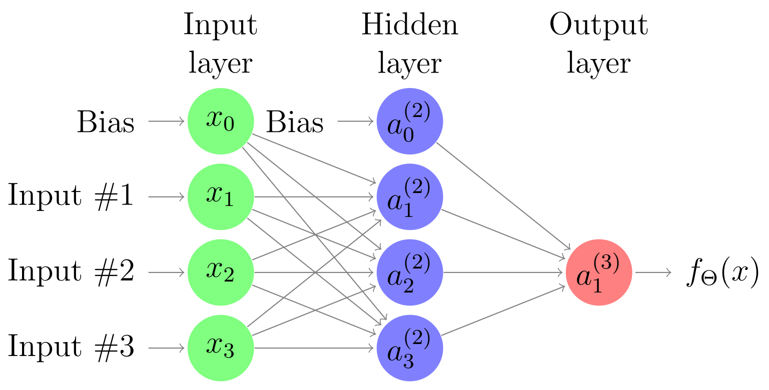
Before discussing the overall structure of the neural network, we describe the structure of an individual node. A typical node takes in inputs from either the external input, or the outputs from other nodes, in addition to a 'bias' node, which is the equivalent of the intercept term in a regression problem. We label these inputs as $\mathbf{x} = [x_0 \quad x_1 \quad x_2 \quad x_3]^\top$, 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 that normalizes the result to a particular range. Denoting the overall node function with $h_{\boldsymbol{\theta}}(\mathbf{x})$ and the activation function as $g(\boldsymbol{x})$, this can be shown in Figure idk below.



The three most common activation functions are:

- The rectified linear unit or ReLU activation function output is within the range $[0, \infty)$. 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 $h_{\boldsymbol{\theta}}(\mathbf{x}) = \tanh(\boldsymbol{\theta}^\top \mathbf{x})$.

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; it's result is simply the output of the network. The layers of nodes between the input and output layer are called the hidden layers, as its weights and outputs are not useful to the user. Hidden layers can have any number of nodes, whilst the nodes in the input and output layers are restricted to the number of inputs and outputs the program has. The figure below illustrates a simple neural network with 3 inputs, 1 hidden layer with 3 nodes and 1 output node.



In this example, we denote the activation function as g, the output of unit 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 unit m in layer $j+1$, and n is the column of the matrix relating to unit 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 $\boldsymbol{\theta}_i^{(j)} = [\Theta_{i,0}^{(j)} \quad \Theta_{i,1}^{(j)} \ldots \Theta_{i,k}^{(j)}]^\top$ where $k+1$ is the number of inputs, we have the vectorized notation:

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

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

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

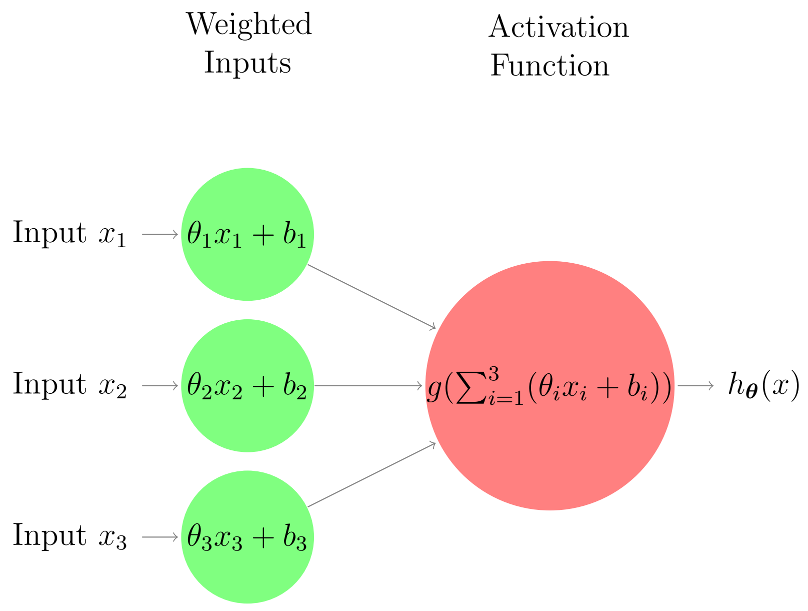
$$a_0^{(2)} = 1$$

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

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

2.2.3 Bias-per-node Representation

In practice, 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. This representation is preferred in code as it makes the dimensionality of the weight matrices consistent, as the number of inputs of a layer becomes equal to the number of outputs of the previous layer. In both cases, the bias takes the form of a flat value added to the weighted inputs, which can be optimized, so 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. Below, we repeat the example of an individual node with three inputs, but with an individual intercept b_i per node.



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

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

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

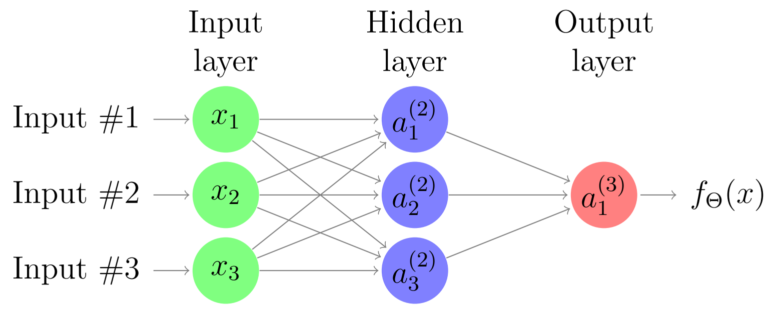
$$a_3^{(2)} = g(\Theta_{3,1}^{(1)} x_1 + b_1^{(1)} + \Theta_{3,2}^{(1)} x_2 + b_2^{(1)} + \Theta_{3,3}^{(1)} x_3 + b_3^{(1)})$$

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

or in vectorized notation,

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

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



2.2.4 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:

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

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 their values are used to calculate the partial derivative of the loss function with respect to each individual weight (and intercept). 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 all based off gradient descent, which we will cover in this section.
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 $\boldsymbol{s}^{(n)}$:

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

Definition: At point $\boldsymbol{x}^{(n)}$, $\boldsymbol{s}^{(n)}$ is a descent direction if $\nabla f(\boldsymbol{x}^{(n)})^{\top}\boldsymbol{s}^{(n)} < 0$.
Preposition: If $\boldsymbol{s}^{(n)}$ is a descent direction, then for small $\alpha > 0$, $f(\boldsymbol{x}^{(n)} + \alpha \boldsymbol{s}^{(n)}) < f(\boldsymbol{x}^{(n)})$.
Proof: (proof copied from jeyakumar optimization lecture notes topic 6)
First we show that

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

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

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

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

$$\begin{aligned} \frac{d}{d\alpha} f(\boldsymbol{x}^{(n)} + \alpha \boldsymbol{s}^{(n)}) &= \frac{d}{d\alpha} f(x_1^{(n)}, \ldots, x_k^{(n)}(\alpha)) \\ &= \frac{\partial f(\boldsymbol{x})}{\partial x_1}\Big|_{\boldsymbol{x}=\boldsymbol{x}^{(n)}+\alpha\boldsymbol{s}^{(n)}} \frac{d(x_1^{(n)}(\alpha)}{d\alpha} + \cdots + \frac{\partial f(\boldsymbol{x})}{\partial x_k}\Big|_{\boldsymbol{x}=\boldsymbol{x}^{(n)}+\alpha\boldsymbol{s}^{(n)}} \frac{d(x_k^{(n)}(\alpha)}{d\alpha} \\ &= \frac{\partial f(\boldsymbol{x})}{\partial x_1}\Big|_{\boldsymbol{x}=\boldsymbol{x}^{(n)}+\alpha\boldsymbol{s}^{(n)}} \boldsymbol{s}_1^{(n)} + \cdots + \frac{\partial f(\boldsymbol{x})}{\partial x_n}\Big|_{\boldsymbol{x}=\boldsymbol{x}^{(n)}+\alpha\boldsymbol{s}^{(n)}} \boldsymbol{s}_k^{(n)} \\ &= \nabla f(\boldsymbol{x}^{(n)} + \alpha \boldsymbol{s}^{(n)})^{\top}\boldsymbol{s}^{(n)} \end{aligned}$$

Setting $\alpha = 0$ and using definition (),

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

Therefore for small $\alpha > 0$,

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

QED.

A common choice of descent direction is the negative of the gradient $-\nabla f(\boldsymbol{x}^{(n)})$, leading to the method of steepest descent. It is clearly a descent direction as $-\nabla f(\boldsymbol{x}^{(n)})^\top \nabla f(\boldsymbol{x}^{(n)}) = -||\nabla f(\boldsymbol{x}^{(n)})||^2 < 0$.
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 if the function is convex, as any local minima are also global minima, but neural networks are not convex as they can approximate any function, including non-convex functions. Those interested in global optimization can refer to (Deterministic Global Optimization, Floudas). In neural network training 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. 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 (Kingma, Ba 2014) incorporates these two concepts to form an effective optimization algorithm that is widely used to optimize neural networks. In this thesis, we use this algorithm, but omit the specifics as they are beyond the scope of this thesis.

2.2.5 Back-propogation

In the back-propogation 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 $(\boldsymbol{x}^{(I)}, \boldsymbol{y}^{(I)})$, $I = 1, \ldots, N$, the input signal is propogated forward throughout the network to calculate $\boldsymbol{a}^{(j)}$ for $j = 2, \ldots, J$, where J is the total number of layers. The difference between the network output and the ideal result is calculated with

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

and this error is propogated backwards through the network to find $\boldsymbol{\delta}^{(J-1)}, \ldots, \boldsymbol{\delta}^{(2)}$ 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 preceeding 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 the back-propogation algorithm is shown below.
Having derived the partial derivatives of the loss function with respect to the individual weights, we can use gradient descent or some other optimization method to update the weights. The partial derivatives are re-calculated after each optimization update until convergence.

Data: Training Data $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), \ldots, (\boldsymbol{x}^{(N)}, \boldsymbol{y}^{(N)})\}$, Regularizer Term λ
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 7: Back-Propogation Algorithm

2.2.6 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 back-propogation training to fix. Similarly, if the weights are too high, then the result output of forward propogation 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 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

$$z = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n.$$

x_0 is a constant term, so $\text{Var}(\theta_0 x_0) = 0$. Now under the assumption that the inputs and weights have 0 mean, we find the variance of the other terms:

$$\text{Var}(\theta_i x_i) = \mathbb{E}[x_i]^2 \text{Var}(\theta_i) + \mathbb{E}[\theta_i]^2 \text{Var}(x_i) + \text{Var}(\theta_i)\text{Var}(x_i) = \text{Var}(\theta_i)\text{Var}(x_i).$$

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

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

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

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

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

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

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

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

2.2.7 References

http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf
Machine Learning by Andrew Ng Stanford online course

3 Experiments

3.1 "Sprinkler" Example

3.1.1 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.

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

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

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 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 plots are as shown in Figure (**need to insert figure**).

As x increases, the posteriors become increasingly multimodal and unusually shaped. Clearly, a flexible model for the posterior distribution is required, as a typical Gaussian model is too simple here.

3.1.2 Network Structure

3.1.3 Results

nothing works lol

3.2 Lotka-Volterra Predator-Prey model

3.2.1 Problem Context

3.3 Autoencoding Variational Bayes (MNIST image generation)

3.3.1 Problem Context

3.4 no idea lol