DD2434 - Machine Learning, Advanced Course
Assignment 1B

Tristan Perrot
tristanp@kth.se

November 2023

Contents

1 CAVI for Earth quakes

1.1 Question 1.1

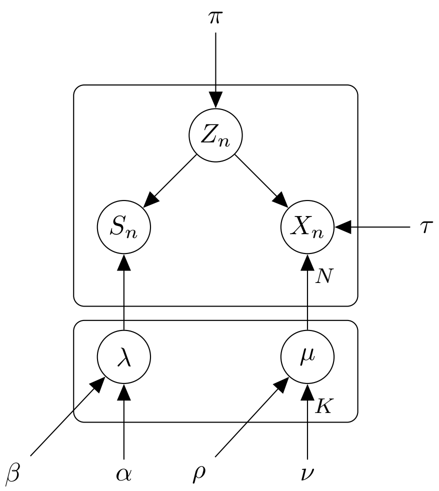


Figure 1: Directed Graphical Model for the Earthquake problem

1.2 Question 1.2

Let us take the Alternative 1 in 2D. Here, we know these distributions:

- $p(Z_n|\pi) = Categorical(\pi)$
- $p(S_n|Z_n = k, \lambda_k) = Poisson(\lambda_k)$
- $p(X_n|Z_n = k, \mu_k, \tau) = Normal(\mu_k, \tau \cdot I)$
- $p(\mu_k|\nu, \rho) = Normal(\nu, \rho \cdot I)$
- $p(\lambda_k|\alpha, \beta) = Gamma(\alpha, \beta)$

Where, ρ and τ define precision and not standard variation. Then we have:

$$\begin{aligned}
\log p(X, S, Z, \lambda, \mu|\pi, \tau, \alpha, \beta, \nu, \rho) &= \log p(X|S, Z, \lambda, \mu, \pi, \tau, \alpha, \beta, \nu, \rho) \\
&\quad + \log p(S, Z, \lambda, \mu|\pi, \alpha, \beta, \nu, \rho) \\
&= \log p(X|Z, \mu, \tau) + \log p(S|Z, \lambda, \mu, \pi, \alpha, \beta, \nu, \rho) \\
&\quad + \log p(Z, \lambda, \mu|\pi\alpha, \beta, \nu, \rho) \\
&= \log p(X|Z, \mu, \tau) + \log p(S|Z, \lambda) + \log p(Z|\pi) \\
&\quad + \log p(\lambda, \mu|\alpha, \beta, \nu, \rho) \\
\log p(X, S, Z, \lambda, \mu|\pi, \tau, \alpha, \beta, \nu, \rho) &= \log p(X|Z, \mu, \tau) + \log p(S|Z, \lambda) + \log p(Z|\pi) \\
&\quad + \log p(\mu|\nu, \rho) + \log p(\lambda|\alpha, \beta)
\end{aligned} \tag{1}$$

Where:

$$\begin{aligned}
\log p(X|Z,\mu,\tau) &= \sum_{n=1}^{N}\sum_{k=1}^{K}\log p(X_n|Z_n=k,\mu_k,\tau) \\
\log p(S|Z,\lambda) &= \sum_{n=1}^{N}\sum_{k=1}^{K}\log p(S_n|Z_n=k,\lambda_k) \\
\log p(Z|\pi) &= \sum_{n=1}^{N}\log p(Z_n|\pi) \\
\log p(\mu|\nu,\rho) &= \sum_{k=1}^{K}\log p(\mu_k|\nu,\rho) \\
\log p(\lambda|\alpha,\beta) &= \sum_{k=1}^{K}\log p(\lambda_k|\alpha,\beta)
\end{aligned} \tag{2}$$

1.3 Question 1.3

Here, the mean field approximation is not an approximation but an equality because Z, μ, λ are independent. Therefore we have:

$$\begin{aligned}
\log q^*(Z_n) &\stackrel{\pm}{=} \mathbb{E}_{\mu,\lambda}[\log p(X_n, S_n, Z_n, \lambda, \mu|\pi,\tau,\alpha,\beta,\nu,\rho)] \\
&\stackrel{\pm}{=} \mathbb{E}_{\mu,\lambda}[\log p(X_n|Z_n,\mu,\tau) + \log p(S_n|Z_n,\lambda) + \log p(Z_n|\pi)] \\
&= \mathbb{E}_{\mu}\left[\sum_{k=1}^{K}\mathbb{1}_{\{Z_n=k\}}\left(\log\left(\frac{\tau}{2\pi}\right) - \frac{\tau}{2}\left((x_n-\mu_k)^T(x_n-\mu_k)\right)\right)\right] \\
&\qquad + \mathbb{E}_{\lambda}\left[\sum_{k=1}^{K}\mathbb{1}_{\{Z_n=k\}}\left(\log(\pi_k) - \lambda_k + S_n\log(\lambda_k) - \log(S_n!)\right)\right] \\
&\stackrel{\pm}{=} \sum_{k=1}^{K}\mathbb{1}_{\{Z_n=k\}}\left(\log\left(\frac{\tau}{2\pi}\right) - \frac{\tau}{2}\mathbb{E}_{\mu}\left[(x_n-\mu_k)^T(x_n-\mu_k)\right]\right. \\
&\qquad \left. + \log(\pi_k) + \mathbb{E}_{\lambda}\left[-\lambda_k + S_n\log(\lambda_k)\right] - \log(S_n!)\right)
\end{aligned} \tag{3}$$

Now, if we take the entire expression that is multiplied by $\mathbb{1}_{\{Z_n=k\}}$ and we call it $u_{n,k}$, we have:

$$q^*(Z_n) \propto \prod_{k=1}^{K} u_{n,k}^{\mathbb{1}_{\{Z_n=k\}}} \tag{4}$$

And if we normalize by taking $r_{n,k} = \frac{u_{n,k}}{\sum_{i=1}^{K} u_{n,i}}$ we get:

$$q^*(Z_n) = \prod_{k=1}^{K} r_{n,k}^{\mathbb{1}_{\{Z_n=k\}}} \tag{5}$$

Wich means that $q^*(Z_n)$ is a categorical distribution with parameters $r_{n,k}$. There for we have the expectation of Z_n easily because $\mathbb{E}[z_{n,k}] = r_{n,k}$ where $z_{n,k} = \mathbb{1}_{\{S_n=k\}}$. Note that $r_{n,k}$ depends of

the expected value of μ_k, μ_k^2, λ_k and $\log \lambda_k$. We will be able to compute these expected values by finding $q^*(\mu_k)$ and $q^*(\lambda_k)$.
Let us compute $q^*(\mu_k)$:

$$\begin{aligned}
\log q^*(\mu_k) &\stackrel{+}{=} \mathbb{E}_{Z,\lambda}[\log p(X, S, Z = k, \lambda_k, \mu_k | \pi, \tau, \alpha, \beta, \nu, \rho)] \\
&\stackrel{+}{=} \mathbb{E}_{Z,\lambda}[\log p(X|Z = k, \mu_k, \tau) + \log p(\mu_k|\nu, \rho)] \\
&= \mathbb{E}_{Z,\lambda}\left[\sum_{n=1}^{N} \mathbb{1}_{\{Z_n=k\}} \left(\log\left(\frac{\tau}{2\pi}\right) - \frac{\tau}{2}\left((x_n - \mu_k)^T (x_n - \mu_k)\right)\right)\right] \\
&\qquad + \log\left(\frac{\rho}{2\pi}\right) - \frac{\rho}{2}\left((\mu_k - \nu)^T(\mu_k - \nu)\right) \\
&\stackrel{+}{=} \sum_{n=1}^{N} r_{n,k}\left(\log\left(\frac{\tau}{2\pi}\right) - \frac{\tau}{2}\left((x_n - \mu_k)^T (x_n - \mu_k)\right)\right) - \frac{\rho}{2}\left((\mu_k - \nu)^T(\mu_k - \nu)\right) \\
&\stackrel{+}{=} \sum_{n=1}^{N} r_{n,k}\left(-\frac{\tau}{2}\left((x_n - \mu_k)^T (x_n - \mu_k)\right)\right) - \frac{\rho}{2}\left((\mu_k - \nu)^T(\mu_k - \nu)\right) \\
&\stackrel{+}{=} -\frac{\tau\sum_{n=1}^{N} r_{n,k}}{2}\left(-2\mu_{k,0}x_{n,0} - 2\mu_{k,1}x_{n,1} + \mu_{k,0}^2 + \mu_{k,1}^2\right) \\
&\qquad - \frac{\rho}{2}\left(-2\mu_{k,0}\nu_0 - 2\mu_{k,1}\nu_1 + \mu_{k,0}^2 + \mu_{k,1}^2\right)
\end{aligned} \tag{6}$$

We define $S = \frac{\rho}{\tau\sum_{n=1}^{N} r_{n,k}}$. Then we have:

$$\begin{aligned}
\log q^*(\mu_k) &\stackrel{+}{=} -\frac{\tau\sum_{n=1}^{N} r_{n,k}}{2}\Big[(S + N)\mu_{k,0}^2 + (S + N)\mu_{k,1}^2 \\
&\qquad -2\mu_{k,0}(S\nu_0 + \sum_{n=1}^{N} x_{n,0}) - 2\mu_{k,1}(S\nu_1 + \sum_{n=1}^{N} x_{n,1})\Big] \\
&\stackrel{+}{=} -\frac{\tau\sum_{n=1}^{N} r_{n,k}}{2(S + N)}\left[\left(\mu_k - \frac{S\nu + \sum_{n=1}^{N} x_n}{S + N}\right)^T \left(\mu_k - \frac{S\nu + \sum_{n=1}^{N} x_n}{S + N}\right)\right]
\end{aligned} \tag{7}$$

Therefore, we have $q^*(\mu_k) = Normal\left(\mu^*, \rho^* \cdot I\right)$. And we can compute the expected value of μ_k and μ_k^2 easily.

$$\begin{aligned}
\mu^* &= \frac{S\nu + \sum_{n=1}^{N} x_n}{S + N} = \frac{\rho\nu + \tau\sum_{n=1}^{N} r_{n,k}x_n}{\rho + N\tau\sum_{n=1}^{N} r_{n,k}} \\
\rho^* &= \frac{\tau\sum_{n=1}^{N} r_{n,k}}{S + N} = \frac{(\tau\sum_{n=1}^{N} r_{n,k})^2}{\rho + N\tau\sum_{n=1}^{N} r_{n,k}}
\end{aligned} \tag{8}$$

And therefore:

$$\begin{aligned}
\mathbb{E}[\mu_k] &= \mu^* \\
\mathbb{E}[\mu_k^2] &= \frac{1}{\rho^*} + \mu^{*T}\mu^*
\end{aligned} \tag{9}$$

Let us compute $q^*(\lambda_k)$:

$$\begin{aligned}\log q^*(\lambda_k) &\stackrel{\pm}{=} \mathbb{E}_{Z,\mu}[\log p(X, S, Z = k, \lambda_k, \mu_k|\pi, \tau, \alpha, \beta, \nu, \rho)] \\ &\stackrel{\pm}{=} \mathbb{E}_{Z,\mu}[\log p(S|Z = k, \lambda_k) + \log p(\lambda_k|\alpha, \beta)] \\ &= \mathbb{E}_Z\left[\sum_{n=1}^{N} \mathbb{1}_{\{Z_n=k\}}\left(-\lambda_k + S_n \log(\lambda_k) - \log(S_n!)\right)\right] \\ &\qquad + \log\left(\frac{\beta^\alpha}{\Gamma(\alpha)}\right) + (\alpha - 1)\log(\lambda_k) - \beta\lambda_k \\ &\stackrel{\pm}{=} \sum_{n=1}^{N} r_{n,k}\left(-\lambda_k + S_n \log(\lambda_k)\right) + (\alpha - 1)\log(\lambda_k) - \beta\lambda_k \\ &= \left(\alpha + \sum_{n=1}^{N} S_n r_{n,k} - 1\right)\log(\lambda_k) - \left(\beta + \sum_{n=1}^{N} r_{n,k}\right)\lambda_k\end{aligned} \tag{10}$$

Therefore, we have $q^*(\lambda_k) = Gamma\left(\alpha + \sum_{n=1}^{N} S_n r_{n,k}, \beta + \sum_{n=1}^{N} r_{n,k}\right)$. And we can compute the expected value of λ_k and $\log \lambda_k$ easily.

$$\begin{aligned}\mathbb{E}[\lambda_k] &= \frac{\alpha + \sum_{n=1}^{N} S_n r_{n,k}}{\beta + \sum_{n=1}^{N} r_{n,k}} \\ \mathbb{E}[\log \lambda_k] &= \psi(\alpha + \sum_{n=1}^{N} S_n r_{n,k}) - \log(\beta + \sum_{n=1}^{N} r_{n,k})\end{aligned} \tag{11}$$

2 VAE image generation

Question 5.1 (in the notebook)

Our objective function is ELBO: $E_{q(z|x)}\big[\log \frac{p(x,z)}{q(z|x)}\big]$
We will show that ELBO can be rewritten as $E_{q(z|x)}\big(\log p(x|z)\big) - D_{KL}\big(q(z|x)||p(z)\big)$ We have:

$$\begin{aligned}E_{q(z|x)}\big[\log \frac{p(x,z)}{q(z|x)}\big] &= E_{q(z|x)}\big[\log p(x,z) - \log q(z|x)\big] \\ &= E_{q(z|x)}\big[\log p(x|z) + \log p(z) - \log q(z|x)\big] \\ &= E_{q(z|x)}\big[\log p(x|z)\big] + E_{q(z|x)}\big[\log p(z)\big] - E_{q(z|x)}\big[\log q(z|x)\big] \\ &= E_{q(z|x)}\big[\log p(x|z)\big] - E_{q(z|x)}\big[\log \frac{q(z|x)}{p(z)}\big] \\ &= E_{q(z|x)}\big[\log p(x|z)\big] - D_{KL}\big(q(z|x)||p(z)\big)\end{aligned}$$

Question 5.2 (in the notebook)

Consider the second term: $-D_{KL}\big(q(z|x)||p(z)\big)$

Question : *Kullback–Leibler divergence can be computed using the closed-form analytic expression when both the variational and the prior distributions are Gaussian. Write down this KL* *divergence in terms of the parameters of the prior and the variational distributions. Your solution should consider a generic case where the latent space is K-dimensional.*

We have:

$$D_{KL}\left(q(z|x)||p(z)\right) = \int q(z|x)\log\frac{q(z|x)}{p(z)}\,dz$$

And we also have:

$$\begin{aligned} q(z|x) = \mathcal{N}(z|\mu(x),\sigma(x)) &= \prod_{i=1}^{K}\left(\frac{1}{\sigma_i\sqrt{2\pi}}\exp\left(-\frac{(z_i-\mu_i)^2}{2\sigma^2}\right)\right) \\ p(z) = \mathcal{N}(z|0,I) &= \left(\frac{1}{\sqrt{2\pi}}\right)^K \exp\left(-\frac{1}{2}z^T z\right) = \prod_{i=1}^{K}\left(\frac{1}{\sqrt{2\pi}}\right)\exp\left(-\frac{1}{2}z_i^2\right) \end{aligned} \tag{12}$$

Therefore:

$$\begin{aligned} D_{KL}\left(q(z|x)||p(z)\right) &= \int q(z|x)\log\frac{q(z|x)}{p(z)}\,dz \\ &= \int q(z|x)\log\frac{\prod_{i=1}^{K}\left(2\pi\sigma_i^2\right)^{-\frac{1}{2}}\exp\left(-\frac{(z_i-\mu_i)^2}{2\sigma_i^2}\right)}{\prod_{i=1}^{K}\left(2\pi\right)^{-\frac{1}{2}}\exp\left(-\frac{z_i^2}{2}\right)}\,dz \\ &= \int q(z|x)\left(\sum_{i=1}^{K}-\log(\sigma_i)-\frac{(z_i-\mu_i)^2}{2\sigma_i^2}+\frac{z_i^2}{2}\right)dz \\ &= \mathbb{E}_{q(z|x)}\left[\sum_{i=1}^{K}-\log(\sigma_i)-\frac{(z_i-\mu_i)^2}{2\sigma_i^2}+\frac{z_i^2}{2}\right] \\ &= \sum_{i=1}^{K}-\log(\sigma_i)-\frac{\mathbb{E}_{q(z|x)}\left[(z_i-\mu_i)^2\right]}{2\sigma_i^2}+\frac{\mathbb{E}_{q(z|x)}\left[z_i^2\right]}{2} \\ &= \sum_{i=1}^{K}-\log(\sigma_i)-\frac{\sigma_i^2}{2\sigma_i^2}+\frac{\mathbb{E}_{q(z|x)}[z_i^2]}{2} \\ &= \sum_{i=1}^{K}-\log(\sigma_i)-\frac{1}{2}+\frac{\sigma_i^2+\mu_i^2}{2} \\ D_{KL}\left(q(z|x)||p(z)\right) &= \frac{1}{2}\sum_{i=1}^{K}\left(\sigma_i^2+\mu_i^2-\log(\sigma_i^2)-1\right) \end{aligned} \tag{13}$$

The rest of tht implementation could be find in the appendix A.1.

3 Reparameterization and the score function

3.1 Question 3.4

According to the Sticking the Landing paper, we can do the reparameterization by sampling z from a parametric distribution $q_\phi(z)$ and by sampling ϵ from a fixed distribution $p(\epsilon)$ and applying a deterministic transformation $t(\epsilon, \phi) = z$ Therefore, we have for the total derivative of the ELBO term:

$$\begin{aligned}\hat{\nabla}_{\text{TD}}(\epsilon, \phi) &= \nabla_\phi[\log p(z|x) + \log p(x) - \log q_\phi(z|x)] \\ &= \nabla_z[\log p(z|x) - \log q_\phi(z|x)]\nabla_\phi z - \nabla_\phi \log q_\phi(z|x) \\ &= \nabla_z[\log p(z|x) - \log q_\phi(z|x)]\nabla_\phi t(\epsilon, \phi) - \nabla_\phi \log q_\phi(z|x)\end{aligned} \tag{14}$$

Where the left term is the score function.

3.2 Question 3.5

Now, we will show that the expectation of the score function is zero.

$$\begin{aligned}\mathbb{E}_{q_\phi(z|x)}\left[\nabla_\phi \log q_\phi(z|x)\right] &= \int q_\phi(z|x)\nabla_\phi \log q_\phi(z|x)\, dz \\ &= \int \nabla_\phi q_\phi(z|x)\, dz \\ &= \nabla_\phi \int q_\phi(z|x)\, dz \\ &= \nabla_\phi 1 \\ &= 0\end{aligned} \tag{15}$$

3.3 Question 3.6

In the Sticking the Landing paper, the author handle this score function by removing it and it results in an unbiased gradient estimator with variance that approaches zero as the approximate posterior approaches the exact posterior.

3.4 Question 3.7

For particular cases, the score function may actually decrease the variance. The name of the concept that describes how the score function acts in this situation is **control variate**.

4 Reparameterization of common distributions

4.1 Question 4.8 - Exponential distribution

The exponential distribution is defined as:

$$p(x|\lambda) = \lambda \exp(-\lambda x) \tag{16}$$

Here, we can use the inverse of the cumulative distribution function to sample from this distribution. The cumulative distribution function is:

$$F(x) = 1 - \exp(-\lambda x) \tag{17}$$

Therefore, the inverse of the cumulative distribution function is:

$$F^{-1}(x) = -\frac{1}{\lambda}\log(1 - x) \tag{18}$$

And as stated in the Sticking the Landing paper, we can sample u from the uniform distribution $U(0,1)$ and apply the inverse of the cumulative distribution function to get a sample from the exponential distribution. Therefore, we have:

$$z = -\frac{1}{\lambda}\log(1-u) \tag{19}$$

The implementation of the reparameterization of the exponential distribution can be found in the appendix A.2.

4.2 Question 4.9 - Categorical distribution

4.2.1 Approximation by the Gumbel-Sofmax distribution

Using the Categorical Reparameterization with Gumbel-Softmax paper, we can describe how to reparameterize the categorical distribution. We have z categorical variable with class probabilities $\pi_1, \ldots, \pi_z$. We can sample z from the categorical distribution by sampling z from the Gumbel-Max distribution. The Gumbel-Max distribution is defined as:

$$G_i = -\log(-\log(u_i)) \tag{20}$$

Where u_i are sampled from the uniform distribution $U(0,1)$. And then sample y_i with Softmax:

$$y_i = \frac{\exp((\log(\pi_i) + G_i)/\tau)}{\sum_{j=1}^{z}\exp((\log(\pi_j) + G_j)/\tau)} \tag{21}$$

Where τ is the temperature parameter.

4.2.2 Using the argmax function

For evaluation purposes, we can use the argmax function to get the most probable class. Therefore, we have:

$$z = \text{one hot}\left(\arg\max_i [G_i + \log \pi_i]\right) \tag{22}$$

The implementation of the reparameterization of the categorical distribution can be found in the appendix A.2.

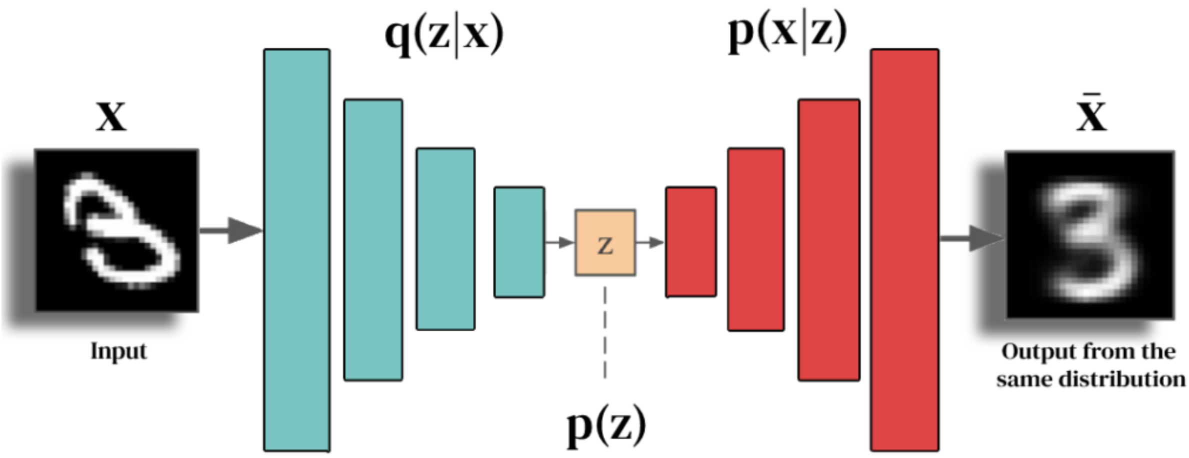
A Appendix

A.1 VAE image generation

VAE for image generation

Consider VAE model from *Auto-Encoding Variational Bayes (2014, D.P. Kingma et. al.)*.

We will implement a VAE model using Torch and apply it to the MNIST dataset.



Generative model: We model each pixel value $\in \{0, 1\}$ as a sample drawn from a Bernoulli distribution. Through a decoder, the latent random variable z_n associated with an image n is mapped to the success parameters of the Bernoulli distributions associated with the pixels of that image. Our generative model is described as follows:

$z_n \sim N(0, I)$

$\theta_n = g(z_n)$

$x_n \sim Bern(\theta_n)$

where g is the decoder. We choose the prior on z_n to be the standard multivariate normal distribution, for computational convenience.

Inference model: We infer the posterior distribution of z_n via variational inference. The variational distribution $q(z_n|x_n)$ is chosen to be multivariate Gaussian with a diagonal covariance matrix. The mean and covariance of this distribution are obtained by applying an encoder to x_n.

$q(z_n|x_n) \sim q(\mu_n, \sigma_n^2)$

where $\mu_n, \sigma_n^2 = f(x_n)$ and f is the encoder.

Implementation: Let's start with importing Torch and other necessary libraries:

In [ ]:
```
import torch
import torch.nn as nn

import numpy as np

from tqdm import tqdm
from torchvision.utils import save_image, make_grid
```

Step1: Model Hyperparameters

In [ ]:
```python
dataset_path = '~/datasets'

batch_size = 100

# Dimensions of the input, the hidden layer, and the latent space.
x_dim = 784
hidden_dim = 400
latent_dim = 200

# Learning rate
lr = 1e-3

# Number of epoch
epochs = 15  # can try something greater if you are not satisfied with the results
```

Step2: Load Dataset

In [ ]:
```python
from torchvision.datasets import MNIST
import torchvision.transforms as transforms
from torch.utils.data import DataLoader

mnist_transform = transforms.Compose([
    transforms.ToTensor(),
])


train_dataset = MNIST(
    dataset_path, transform=mnist_transform, train=True, download=True)
test_dataset = MNIST(dataset_path, transform=mnist_transform,
                     train=False, download=True)

train_loader = DataLoader(dataset=train_dataset,
                          batch_size=batch_size, shuffle=True)
test_loader = DataLoader(dataset=test_dataset,
                         batch_size=batch_size, shuffle=False)
```

Step3: Define the model

In [ ]:
```python
class Encoder(nn.Module):
    # encoder outputs the parameters of variational distribution "q"
    def __init__(self, input_dim, hidden_dim, latent_dim):
        super(Encoder, self).__init__()

        # FC stands for a fully connected layer
        self.FC_enc1 = nn.Linear(input_dim, hidden_dim)
        self.FC_enc2 = nn.Linear(hidden_dim, hidden_dim)
        self.FC_mean = nn.Linear(hidden_dim, latent_dim)
        self.FC_var = nn.Linear(hidden_dim, latent_dim)

        # will use this to add non-linearity to our model
        self.LeakyReLU = nn.LeakyReLU(0.2)

        self.training = True

    def forward(self, x):
        h_1 = self.LeakyReLU(self.FC_enc1(x))
        h_2 = self.LeakyReLU(self.FC_enc2(h_1))
        mean = self.FC_mean(h_2)  # mean
        log_var = self.FC_var(h_2)   # log of variance

        return mean, log_var
```

In [ ]:
```python
class Decoder(nn.Module):
    # decoder generates the success parameter of each pixel
    def __init__(self, latent_dim, hidden_dim, output_dim):
        super(Decoder, self).__init__()
        self.FC_dec1 = nn.Linear(latent_dim, hidden_dim)
        self.FC_dec2 = nn.Linear(hidden_dim, hidden_dim)
        self.FC_output = nn.Linear(hidden_dim, output_dim)

        self.LeakyReLU = nn.LeakyReLU(0.2)  # again for non-linearity

    def forward(self, z):
        h_out_1 = self.LeakyReLU(self.FC_dec1(z))
        h_out_2 = self.LeakyReLU(self.FC_dec2(h_out_1))

        theta = torch.sigmoid(self.FC_output(h_out_2))
        return theta
```

Q3.1 (2 points) Below implement the reparameterization function.

In [ ]:
```python
class Model(nn.Module):
    def __init__(self, Encoder, Decoder):
        super(Model, self).__init__()
        self.Encoder = Encoder
        self.Decoder = Decoder
```

```
    def reparameterization(self, mean, var):
        # insert your code here
        std = torch.sqrt(var + 1e-10)
        eps = torch.randn_like(std)
        z = mean + std * eps

        return z

    def forward(self, x):
        mean, log_var = self.Encoder(x)
        # takes exponential function (log var -> var)
        z = self.reparameterization(mean, torch.exp(log_var))

        theta = self.Decoder(z)

        return theta, mean, log_var
```

Step4: Model initialization

In [ ]:
```
encoder = Encoder(input_dim=x_dim, hidden_dim=hidden_dim,
                  latent_dim=latent_dim)


decoder = Decoder(latent_dim=latent_dim,
                  hidden_dim=hidden_dim, output_dim=x_dim)



model = Model(Encoder=encoder, Decoder=decoder)
```

Step5: Loss function and optimizer

Our objective function is ELBO: $E_{q(z|x)}\big[\log\frac{p(x,z)}{q(z|x)}\big]$

- **Q5.1 (1 point)** Show that ELBO can be rewritten as :

  $E_{q(z|x)}\big(\log p(x|z)\big) - D_{KL}\big(q(z|x)||p(z)\big)$

5.1 Your answer

$$
\begin{aligned}
E_{q(z|x)}\Big[\log\frac{p(x,z)}{q(z|x)}\Big] &= E_{q(z|x)}\big[\log p(x,z) - \log q(z|x)\big] \\
&= E_{q(z|x)}\big[\log p(x|z) + \log p(z) - \log q(z|x)\big] \\
&= E_{q(z|x)}\big[\log p(x|z)\big] + E_{q(z|x)}\big[\log p(z)\big] - E_{q(z|x)}\big[\log q(z|x)\big] \\
&= E_{q(z|x)}\big[\log p(x|z)\big] - E_{q(z|x)}\Big[\log\frac{q(z|x)}{p(z)}\Big] \\
&= E_{q(z|x)}\big[\log p(x|z)\big] - D_{KL}\big(q(z|x)||p(z)\big)
\end{aligned}
$$

Consider the first term: $E_{q(z|x)}\big(\log p(x|z)\big)$

$E\_q(z|x)\big(\log p(x|z)\big) = \int q(z|x)\log p(x|z)dz$

We can approximate this integral by Monte Carlo integration as following:

$\approx \frac{1}{L}\sum_{l=1}^{L}\log p(x|z_l)$, where $z_l \sim q(z|x)$.

Now we can compute this term using the analytic expression for $p(x|z)$. (Remember we model each pixel as a sample drawn from a Bernoulli distribution).

Consider the second term: $-D_{KL}\big(q(z|x)||p(z)\big)$

- **Q5.2 (2 points)** Kullback–Leibler divergence can be computed using the closed-form analytic expression when both the variational and the prior distributions are Gaussian. Write down this KL divergence in terms of the parameters of the prior and the variational distributions. Your solution should consider a generic case where the latent space is K-dimensional.

5.2 Your answer

$$
\begin{aligned}
D_{KL}\left(q(z|x)||p(z)\right) &= \int q(z|x)\log\frac{q(z|x)}{p(z)}\,dz \\
&= \int q(z|x)\log\frac{\prod_{i=1}^{K}\left(2\pi\sigma_i^2\right)^{-\frac{1}{2}}\exp\left(-\frac{(z_i-\mu_i)^2}{2\sigma_i^2}\right)}{\prod_{i=1}^{K}\left(2\pi\right)^{-\frac{1}{2}}\exp\left(-\frac{z_i^2}{2}\right)}\,dz \\
&= \int q(z|x)\left(\sum_{i=1}^{K}-\log(\sigma_i)-\frac{(z_i-\mu_i)^2}{2\sigma_i^2}+\frac{z_i^2}{2}\right)\,dz \\
&= \mathbb{E}_{q(z|x)}\left[\sum_{i=1}^{K}-\log(\sigma_i)-\frac{(z_i-\mu_i)^2}{2\sigma_i^2}+\frac{z_i^2}{2}\right] \\
&= \sum_{i=1}^{K}-\log(\sigma_i)-\frac{\mathbb{E}_{q(z|x)}\left[(z_i-\mu_i)^2\right]}{2\sigma_i^2}+\frac{\mathbb{E}_{q(z|x)}\left[z_i^2\right]}{2} \\
&= \sum_{i=1}^{K}-\log(\sigma_i)-\frac{\sigma_i^2}{2\sigma_i^2}+\frac{\mathbb{E}_{q(z|x)}[z_i^2]}{2} \\
&= \sum_{i=1}^{K}-\log(\sigma_i)-\frac{1}{2}+\frac{\sigma_i^2+\mu_i^2}{2} \\
D_{KL}\left(q(z|x)||p(z)\right) &= \frac{1}{2}\sum_{i=1}^{K}\left(\sigma_i^2+\mu_i^2-\log(\sigma_i^2)-1\right)
\end{aligned}
$$

Q5.3 (5 points) Now use your findings to implement the loss function, which is the negative of ELBO:

```
In [ ]: from torch.optim import Adam


        def loss_function(x, theta, mean, log_var):  # should return the loss function (- ELBO)
            # insert your code here
            # expected log-likelihood
            recon_loss = -torch.sum(x * torch.log(theta + 1e-10) +
                                    (1 - x) * torch.log(1 - theta + 1e-10))

            # KL Divergence
            kl_div = -0.5 * torch.sum(1 + log_var - mean.pow(2) - log_var.exp())

            loss = recon_loss + kl_div

            return loss


        # optimizer
        optimizer = Adam(model.parameters(), lr=lr)
```

Step6: Train the model

```
In [ ]: print("Start training VAE...")
        model.train()

        for epoch in range(epochs):
            overall_loss = 0
            for batch_idx, (x, _) in enumerate(train_loader):
                x = x.view(batch_size, x_dim)
                x = torch.round(x)

                optimizer.zero_grad()

                theta, mean, log_var = model(x)
                loss = loss_function(x, theta, mean, log_var)
                overall_loss += loss.item()

                loss.backward()
                optimizer.step()

            print("\tEpoch", epoch + 1, "complete!", "\tAverage Loss: ",
                  overall_loss / (batch_idx*batch_size))

        print("Finish!!")
```

```
Start training VAE...
	Epoch 1 complete! 	Average Loss:  171.05750705929154
	Epoch 2 complete! 	Average Loss:  120.2330909262834
	Epoch 3 complete! 	Average Loss:  103.97579276006887
	Epoch 4 complete! 	Average Loss:  98.17505150185204
	Epoch 5 complete! 	Average Loss:  94.57102987400876
	Epoch 6 complete! 	Average Loss:  91.87520890938022
	Epoch 7 complete! 	Average Loss:  90.07733892424875
	Epoch 8 complete! 	Average Loss:  88.65414684467602
	Epoch 9 complete! 	Average Loss:  87.63794590919763
	Epoch 10 complete! 	Average Loss:  86.61679169872966
	Epoch 11 complete! 	Average Loss:  85.82079130093123
	Epoch 12 complete! 	Average Loss:  85.06231359720628
	Epoch 13 complete! 	Average Loss:  84.45940288664701
	Epoch 14 complete! 	Average Loss:  84.02014981023059
	Epoch 15 complete! 	Average Loss:  83.48515547559735
Finish!!
```

Step7: Generate images from test dataset

With our model trained, now we can start generating images.

First, we will generate images from the latent representations of test data.

Basically, we will sample z from $q(z|x)$ and give it to the generative model (i.e., decoder) $p(x|z)$. The output of the decoder will be displayed as the generated image.

Q7.1 (2 points) Write a code to get the reconstructions of test data, and then display them using the show_image function

In [ ]:
```python
model.eval()
# below we get decoder outputs for test data
with torch.no_grad():
    for batch_idx, (x, _) in enumerate(tqdm(test_loader)):
        x = x.view(batch_size, x_dim)
        # insert your code below to generate theta from x

        # Pass the test images through the encoder and decoder
        mean, log_var = model.Encoder(x)
        # reparameterize to get latent variable
        z = model.reparameterization(mean, torch.exp(log_var))
        # decode the latent variable to get reconstructed image
        theta = model.Decoder(z)
```

```
100%|████████████████████████████████████████████████████████████| 100/100 [00:06<00:00, 14.64it/s]
```

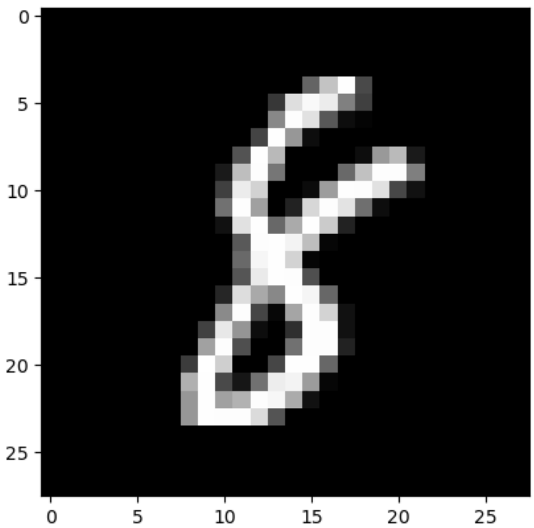
A helper function to display images:

In [ ]:
```python
import matplotlib.pyplot as plt


def show_image(theta, idx):
    x_hat = theta.view(batch_size, 28, 28)
    # x_hat = Bernoulli(x_hat).sample() # sample pixel values (you can also try this, and observe how the generated images look)
    fig = plt.figure()
    plt.imshow(x_hat[idx].cpu().numpy(), cmap='gray')
```

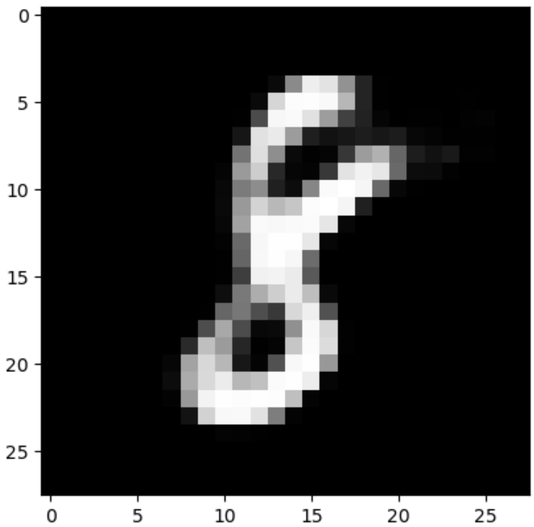
First display an image from the test dataset,

In [ ]:
```python
show_image(x, idx=0)  # try different indices as well
```



Now display its reconstruction and compare:

In [ ]:
```python
show_image(theta, idx=0)
```



Step8: Generate images from noise

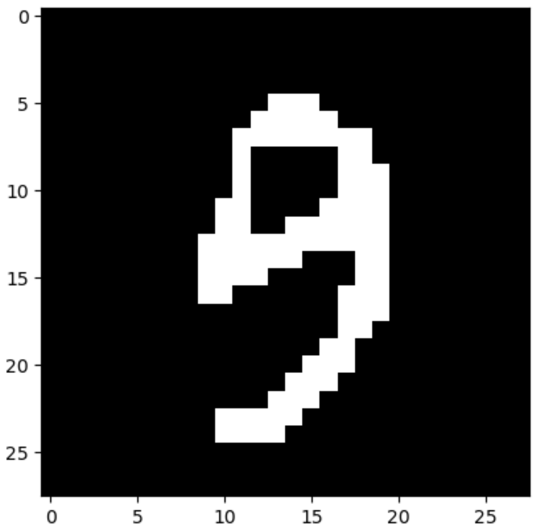
In the previous step, we sampled latent vector z from $q(z|x)$. However, we know that the KL term in our loss function enforced $q(z|x)$ to be close to $N(0, I)$. Therefore, we can sample z directly from noise $N(0, I)$, and pass it to the decoder $p(x|z)$.

Q8.1 (3 points) Create images from noise and display.

In [ ]:
```
with torch.no_grad():
    # insert your code here to create images from noise (it is enough to create theta value for each pixel)
    #
    #
    # generated_images = ....  # should be a matrix ( batch_size-by-x_dim )
    generated_images = torch.round(
        model.Decoder(torch.randn(batch_size, latent_dim)))
```

Display a couple of generated images:

In [ ]:
```
show_image(generated_images, idx=0)
```



A.2 Reparameterization of common distributions

Reparameterization of common distributions

We will work with Torch throughout this notebook.

In [ ]:
```
import torch
from torch.distributions import Exponential, Categorical #, ...  import the distributions you need here
from torch.nn import functional as F
```

A helper function to visualize the generated samples:

In [ ]:
```
import matplotlib.pyplot as plt
def compare_samples (samples_1, samples_2, bins=100, range=None):
  fig = plt.figure()
  if range is not None:
    plt.hist(samples_1, bins=bins, range=range, alpha=0.5)
    plt.hist(samples_2, bins=bins, range=range, alpha=0.5)
  else:
    plt.hist(samples_1, bins=bins, alpha=0.5)
    plt.hist(samples_2, bins=bins, alpha=0.5)
  plt.xlabel('value')
  plt.ylabel('number of samples')
  plt.legend(['direct','via reparameterization'])
  plt.show()
```

Q1. Exponential Distribution

Below write a function that generates N samples from $Exp(\lambda)$.

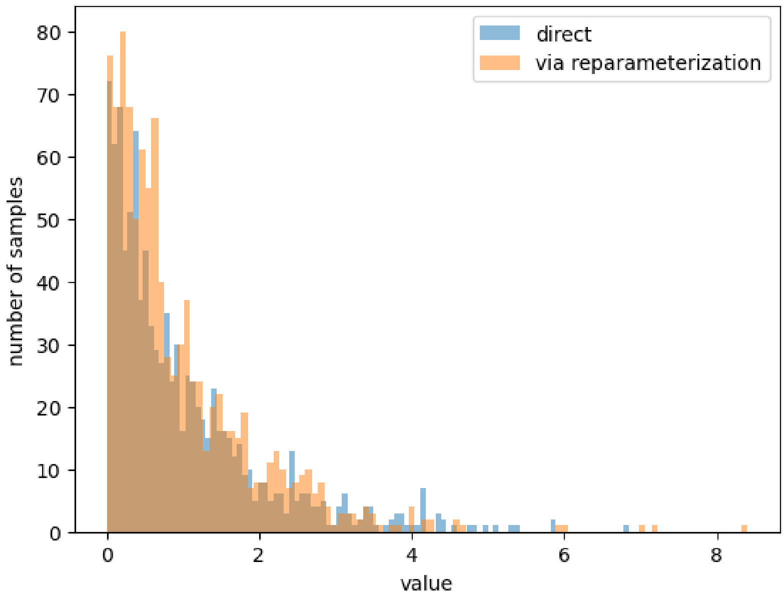
In [ ]:
```
def exp_sampler(l, N):
  # insert your code
  samples = Exponential(l).sample((N,))
  return samples # should be N-by-1
```

Now, implement the reparameterization trick:

In [ ]:
```
def exp_reparametrize(l,N):
  # this function should return N samples via reparametrization,
  # insert your code
  samples = -1/l *torch.log(1-torch.rand(N,))
  return samples
```

Generate samples for $\lambda = 1$ and compare:

In [ ]:
```
l = 1    #lambda
N = 1000
direct_samples = exp_sampler(l, N)
reparametrized_samples = exp_reparametrize(l, N)
direct_samples = direct_samples.detach().numpy()
reparametrized_samples = reparametrized_samples.detach().numpy()
compare_samples(direct_samples, reparametrized_samples)
```



Q2. Categorical Distribution

Below write a function that generates N samples from Categorical (**a**), where **a** = $[a_0, a_1, a_2, a_3]$.

In [ ]:
```
def categorical_sampler(a, N):
  # insert your code
  samples = Categorical(a).sample((N,))
  return samples  # should be N-by-1
```

Now write a function that generates samples from Categorical (**a**) via reparameterization:

In [ ]:
```
# Hint: approximate the Categorical distribution with the Gumbel-Softmax distribution
def categorical_reparametrize(a, N, temp=0.1, eps=1e-20):  # temp and eps are hyperparameters for Gumbel-Softmax
  # insert your code
  samples = torch.zeros(N, a.shape[0])
  for i in range(N):
    samples[i] = F.gumbel_softmax(a, tau=temp, hard=True, eps=eps)

  return samples # make sure that your implementation allows the gradient to backpropagate
```

Generate samples when $a = [0.1, 0.2, 0.5, 0.2]$ and visualize them:

In [ ]:
```
a = torch.tensor([0.1,0.2,0.5,0.2])
N = 1000
direct_samples = categorical_sampler(a, N)
reparametrized_samples = categorical_reparametrize(a, N, temp=0.1, eps=1e-20)
direct_samples = direct_samples.detach().numpy()
reparametrized_samples = reparametrized_samples.detach().numpy()
compare_samples(direct_samples, reparametrized_samples)
```