Probabilistic Machine Learning
Assignment 1

Group 12

01/12/2024

Bishop 2.8

We will start by showing that

$$\mathbb{E}[x] = \mathbb{E}_y[\mathbb{E}_x[x \mid y]] \tag{1}$$

We assume that $p(x, y)$ is well defined and the expectations are integrateable.
We then have

$$\begin{aligned}
\mathbb{E}[x] &= \int_x x p(x) dx && \text{(By definition)} \\
&= \int_x x \left(\int_y p(x, y) dy \right) dx \\
&= \int_y \left(\int_x x p(x, y) dx \right) dy \\
&= \int_y p(y) \left(\int_x \frac{x p(x, y)}{p(y)} dx \right) dy \\
&= \int_y p(y) \mathbb{E}_x[x \mid y] dy && \text{(Def. of conditional expectation)} \\
&= \mathbb{E}_y[\mathbb{E}_x[x \mid y]] && \text{(Def. of expectation)}
\end{aligned}$$

We will now show that

$$\text{var}[x] = \mathbb{E}_y \left[\text{var}_x[x \mid y] \right] + \text{var}_y \left[\mathbb{E}_x[x \mid y] \right]$$

We have that

$$\begin{aligned}
\text{var}[x] &= \mathbb{E}[x^2] - (\mathbb{E}[x])^2 && \text{(By definition)} \\
&= \mathbb{E}_y[\mathbb{E}_x[x^2 \mid y]] - (\mathbb{E}_y[\mathbb{E}_x[x \mid y]])^2 && \text{(Eq. 1)} \\
&= \mathbb{E}_y[\text{var}_x[x \mid y] + \mathbb{E}_x[x \mid y]] - (\mathbb{E}_y[\mathbb{E}_x[x \mid y]])^2 && \text{(Def. of variance)} \\
&= \mathbb{E}_y[\text{var}_x[x \mid y]] + \mathbb{E}_y[x \mid y] - (\mathbb{E}_y[\mathbb{E}_x[x \mid y]])^2 && \text{(Linearity of expectation)} \\
&= \mathbb{E}_y[\text{var}_x[x \mid y]] + \text{var}_y[\mathbb{E}_x[x \mid y]] && \text{(Def. of conditional variance)}
\end{aligned}$$

■

Bishop 8.9

We denote the Markov Blanket for some node x as $MB(x)$. A path from some node $v \in \setminus MB(x) \cup x$ to x must pass through either a parent p, or a child, c of x.

If it passes through p the path will either meet head-to-tail or tail-to-tail at p and will therefore be blocked since p is conditioned on.

If the path passes through c it will either pass through some child of c or through some co-parent of c. In the case where it passes through a child, it will meet head-to-tail where the path then will be blocked since we condition on c. In the case it passes through a co-parent it won't be blocked by c since it then will meet head-to-head. Let denote this co-parent of c as cp. The path will then meet cp either head-to-tail or tail-to-tail. Since we condition on cp the path will therefore be blocked.

■

Bishop 8.11

First we want to calculate $P(F = 0 \mid D = 0)$. We have from the book that $P(G = 0 \mid F = 0) = 0.81$. We start by calculating the numerator in Bayes' Theorem by

$$P(F = 0)P(D = 0 \mid F = 0) = P(F = 0) \sum_{G \in \{0,1\}} P(D = 0 \mid G)P(G \mid F = 0) = 0.0748$$

We have also from the book that $P(G = 0) = 0.315$. We can therefore calculate the denominator by

$$P(D = 0) = \sum_{G \in \{0,1\}} P(D = 0 \mid G)P(G) = 0.352$$

We therefore have

$$P(F = 0 \mid D = 0) = \frac{P(D = 0 \mid F = 0)P(F = 0)}{P(D = 0)} = 0.2125$$

We now observe that $B = 0$. We then have

$$\begin{aligned} P(F = 0 \mid D = 0, B = 0) &= \frac{P(D = 0 \mid F = 0, B = 0)P(F = 0)}{P(D = 0 \mid B = 0)} \\ &= \frac{\sum_{G \in \{0,1\}} P(D = 0 \mid G)P(G \mid B = 0, F = 0)P(F = 0)}{\sum_{G \in \{0,1\}} \sum_{F \in \{0,1\}} P(D = 0 \mid G)P(G \mid B = 0, F)P(F)} \\ &\simeq 0.0587 \end{aligned}$$

We therefore see that the probability that tank is empty has decreased from 0.2125 to 0.0587. This agrees with our intuition since we now know that the battery is flat which explains away the observation. The graphical model of the problem equates to the one we see in Fig 8.54. We also see how observing a descendant of G unblocks the path between B and F making them conditional dependent on D.

Week 1 Programming exercise

We start by sampling 20 points from $x \sim N(0, I), x \in \mathbb{R}^2$ and respective labels according to the distribution $p(y \mid x) = \mathcal{N}\left(y; x^T\theta, 0.1\right), \theta = [-1, 1]^T$. We can then compute the mean and covariance of the posterior by

$$\mu_{\text{posterior}} = \Sigma_{\text{posterior}} \left(X^T y/\sigma_y\right)$$

and

$$\Sigma_{\text{posterior}} = \left(\Sigma_{\text{prior}}^{-1} + \frac{1}{\sigma_y} X^T X\right)^{-1}$$

Where a plot of this distribution can be seen in Fig. 1. We can then calculate

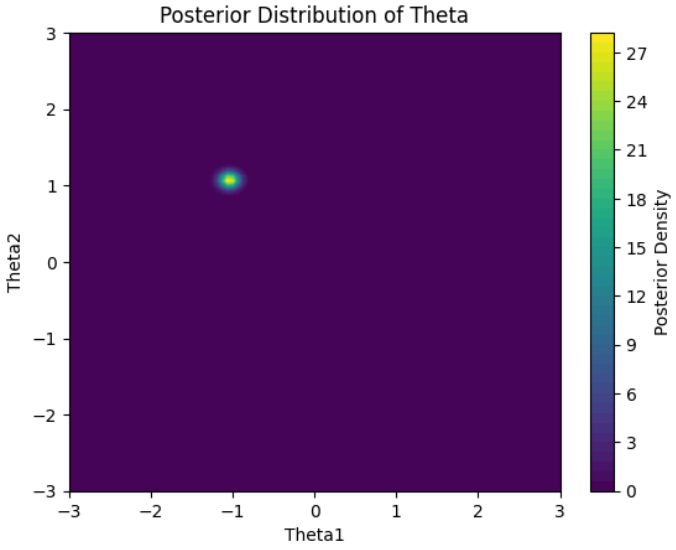

Figure 1: Posterior distribution for θ

the variance of the posterior predictive by $x^T\Sigma_\theta x + \sigma^2$. A plot of the variance can be seen on Fig. 2. . We can now repeat this but where we sample from $x \sim \mathcal{N}\left(0, \Sigma_x\right), \Sigma_X = \begin{bmatrix} 0.1 & 0 \\ 0 & 1 \end{bmatrix}$. A plot of the posterior distribution and the

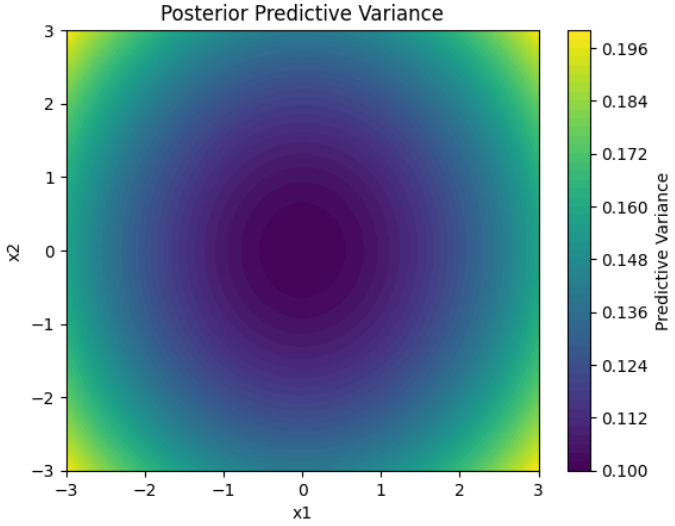

Figure 2: The variance of the posterior predictive

variance of the posterior predictive can be seen in Fig. 3. We can see that the posterior distribution now is wider than before. This is due to sampling with less variance along the x_1-axis leading to less information about θ_1 and therefore a more uncertain (wider) estimate. If we where to introduce less noise for the labels the posterior distribution would become more narrow reflecting a higher certainty in parameter estimation. The predictive variance would also become reduced. .

Bishop 9.10

$$
\begin{aligned}
\mathrm{p}(x_b|x_a) &= \frac{p(x_b, x_a)}{p(x_a)} && \text{Bayes theoreme} \\
&= \frac{\sum_{k=1}^{K} \pi_k p(x_b, x_a|k)}{p(x_a)} && \text{using } p(x) = \sum_{k=1}^{K} p(x|k) \\
&= \frac{\sum_{k=1}^{K} \pi_k p(x_b|x_a, k) p(x_a|k)}{\sum_{j=1}^{K} \pi_j p(x_a|k)} && \text{chain rule} \\
&= \sum_{k=1}^{K} \frac{\pi_k p(x_a|k)}{\sum_{j=1}^{K} \pi_j p(x_a|k)} p(x_b|x_a, k) && \text{rearranging} \\
&= \sum_{k=1}^{K} \pi'_k p(x_b|x_a, k) && \pi'_k = \frac{\pi_k p(x_a|k)}{\sum_{j=1}^{K} \pi_j p(x_a|k)}
\end{aligned}
$$

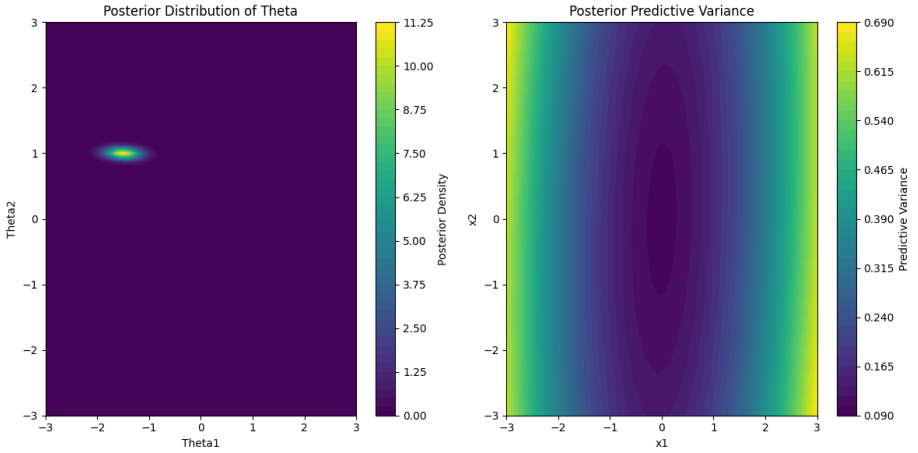

Figure 3: The variance of the posterior predictive

Hence we have shown expressions for mixing coefficients π'_k and component densities $p(x_b|x_a, k) = \frac{p(x_b,x_a|k)}{p(x_a|k)}$

Bishop 10.4

We can find the minimization of the KL divergence between p and q by doing the derivative wrt μ and Σ and setting equal to zero. This works since $p(x)$ is fixed which implies $\mathrm{KL}(p||q)$ is convex meaning local minimum is global minimum. Formally we want to find these two quantities i.e. $\frac{d}{d\mu}\mathrm{KL}(p||q) = 0 \wedge \frac{d}{d\Sigma}\mathrm{KL}(p||q) = 0$. We start by deriving an expression usable for both expressions and then find $\frac{\partial}{\partial\mu}\log q(x) \wedge \frac{\partial}{\partial\Sigma}\log q(x)$

$$\begin{aligned}
\mathrm{KL}(p||q) &= \int p(x)\log(\frac{p(x)}{q(x)})dx && \text{definition of } KL(p||q)\\
&= \int p(x)\log p(x) - p(x)\log q(x)dx && \frac{\log p}{\log q} = \log p - \log q
\end{aligned}$$

$$\begin{aligned}
\frac{\partial}{\partial\mu}\log q(x) &= \frac{\partial}{\partial\mu}\log[(2\pi)^{-k/2}\det(\Sigma)^{-1/2}\exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))]\\
&= \frac{\partial}{\partial\mu}[-\frac{k}{2}\log(2\pi) - \frac{1}{2}\log(\det(\Sigma) - \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))] && \text{log rules}\\
&= \frac{\partial}{\partial\mu}[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)] && \text{non } \mu \text{ terms are zero}\\
&= -\frac{1}{2}[\frac{\partial}{\partial\mu}((x-\mu)^T\Sigma^{-1}(x-\mu))]\\
&= -\frac{1}{2}[\frac{\partial(x-\mu)^T}{\partial\mu}\Sigma^{-1}(x-\mu) + (x-\mu)^T\Sigma^{-1}\frac{\partial(x-\mu)}{\partial\mu}] && \text{chain rule}\\
&= -\frac{1}{2}[-\mathrm{I}\,\Sigma^{-1}(x-\mu) - (x-\mu)^T\Sigma^{-1}\,\mathrm{I}]\\
&= -\frac{1}{2}[-\Sigma^{-1}(x-\mu) - (x-\mu)^T\Sigma^{-1}] && \mathrm{I}\Sigma^{-1} = \Sigma^{-1}\mathrm{I}{=}\Sigma^{-1}\\
&= -\frac{1}{2}[-2\Sigma^{-1}(x-\mu)] && (AB)^T = B^TA^T\\
&= \Sigma^{-1}(x-\mu)
\end{aligned}$$

From this point we find the derivative w.r.t μ

$$
\begin{aligned}
\frac{d}{d\mu}\,\mathrm{KL}(p||q) = 0 &= \frac{d}{d\mu}\int p(x)\log p(x) - p(x)\log q(x)dx \\
&= \int \frac{d}{d\mu}p(x)\log p(x) - \frac{d}{d\mu}p(x)\log q(x)dx \\
&= -\int p(x)\frac{d}{d\mu}\log q(x)dx \\
&= -\int p(x)\Sigma^{-1}(x-\mu)dx \\
&= -\Sigma^{-1}\int p(x)(x-\mu)dx \\
&= -\Sigma^{-1}[\int p(x)xdx - \mu\int p(x)dx] \qquad\qquad \int p(x)dx = 1 \\
&= -(\mathbb{E}_{x\sim p(x)}[x] - \mu) \\
&= -\mathbb{E}_{x\sim p(x)}[x] + \mu
\end{aligned}
$$

This is equivalent to $\mu = \mathbb{E}_{x\sim p(x)}[x]$

Now we do the derivative wrt to the covariance:

$$
\begin{aligned}
\frac{\partial}{\partial\Sigma}\log q(x) &= \frac{\partial}{\partial\Sigma}\log[(2\pi)^{-k/2}\det(\Sigma)^{-1/2}\exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))] \\
&= \frac{\partial}{\partial\Sigma}[-\frac{k}{2}\log(2\pi) - \frac{1}{2}\log(\det(\Sigma) - \frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))] \quad \text{log rules} \\
&= -\frac{1}{2}\frac{\partial}{\partial\Sigma}\log(\det(\Sigma)) - \frac{1}{2}(x-\mu)^T\frac{\partial}{\partial\Sigma}\Sigma^{-1}(x-\mu) \quad \text{non sigma terms are zero} \\
&= -\frac{1}{2}\frac{1}{det(\Sigma)}det(\Sigma)(\Sigma^{-1})^T - \frac{1}{2}(x-\mu)^T(-\Sigma^{-1}\Sigma^{-1})(x-\mu) \\
&= -\frac{1}{2}\Sigma^{-1} - \frac{1}{2}(x-\mu)^T(-\Sigma^{-1}\Sigma^{-1})(x-\mu) \\
&= -\frac{1}{2}[\Sigma^{-1} + (x-\mu)^T(-\Sigma^{-1}\Sigma^{-1})(x-\mu)] \\
&= -\frac{1}{2}[\Sigma^{-1} - \Sigma^{-1}(x-\mu)(x-\mu)^T\Sigma^{-1}]
\end{aligned}
$$

66 We substitute in $\frac{\partial}{\partial \Sigma} \log q(x)$

$$
\begin{aligned}
\frac{d}{d\Sigma} \mathrm{KL}(p||q) &= 0 \\
&= \frac{d}{d\mu} \int p(x) \log p(x) - p(x) \log q(x) dx \\
&= \int \frac{d}{d\Sigma} p(x) \log p(x) - \frac{d}{d\Sigma} p(x) \log q(x) dx \\
&= - \int p(x) \frac{d}{d\Sigma} \log q(x) dx \\
&= - \int p(x) [-\frac{1}{2} (\Sigma^{-1} - \Sigma^{-1}(x-\mu)(x-\mu)^T \Sigma^{-1}] dx \\
&= \frac{1}{2} \int p(x) [(\Sigma^{-1} - \Sigma^{-1}(x-\mu)(x-\mu)^T \Sigma^{-1}] dx \\
&= \frac{1}{2} \Sigma^{-1} \int p(x) dx - \frac{1}{2} \int p(x) \Sigma^{-1}(x-\mu)(x-\mu)^T \Sigma^{-1} dx \\
&= \frac{1}{2} \Sigma^{-1} - \frac{1}{2} \Sigma^{-1} \int p(x)(x-\mu)(x-\mu)^T dx \Sigma^{-1} \\
&= \frac{1}{2} \Sigma^{-1} - \frac{1}{2} \Sigma^{-1} \mathbb{E}_{x \sim p(x)} [(x-\mu)(x-\mu)^T] \Sigma^{-1} \\
&\iff \\
&\frac{1}{2} \Sigma^{-1} = \frac{1}{2} \Sigma^{-1} \mathbb{E}_{x \sim p(x)} [(x-\mu)(x-\mu)^T] \Sigma^{-1} \\
&\Sigma^{-1} = \Sigma^{-1} \mathbb{E}_{x \sim p(x)} [(x-\mu)(x-\mu)^T] \Sigma^{-1} \\
&\Sigma \Sigma^{-1} = \Sigma \Sigma^{-1} \mathbb{E}_{x \sim p(x)} [(x-\mu)(x-\mu)^T] \Sigma^{-1} \\
&I = I \mathbb{E}_{x \sim p(x)} [(x-\mu)(x-\mu)^T] \Sigma^{-1} \\
&I\Sigma = I \mathbb{E}_{x \sim p(x)} [(x-\mu)(x-\mu)^T] \Sigma^{-1} \Sigma \\
&\Sigma = \mathbb{E}_{x \sim p(x)} [(x-\mu)(x-\mu)^T]
\end{aligned}
$$

67 We've now shown what was asked in the exercise.

Week 2 Programming exercise

Part 1

We trained a VAE that downsampled in the following way; $28 \times 28 \rightarrow 14 \times 14 \rightarrow 7 \times 7$ using convolutions. This is linearly projected through two seperate linear layers representing μ and Σ. We then used these pararmeters to obtain (z_1, z_2), which was projected up into 7×7. The upsampling steps are directly inverse proportional to the downsampling steps i.e. $7 \times 7 \rightarrow 14 \times 14 \rightarrow 28 \times 28$. The model was trained for 50 epochs using Adam and $lr = 3e-4$. The most important detail is we weighted KL by an arbritrarily set factor of 0.001 in order to avoid mode collpase which was the default behavior.

We notice the points approximately are within a disk around origo which is to be expected as our latent space representation tries to approximate $\mathcal{N}(0, I)$.

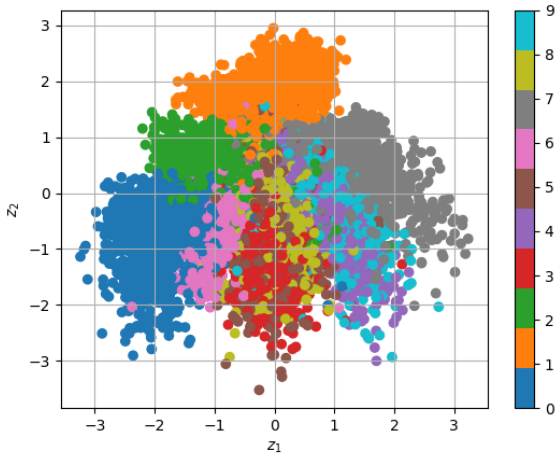

Figure 4: Encoded test samples with color-code for the different 10 digit classes.

Week 2 Programming exercise

We chose to use make a plot of $k = 15$ i.e. (15×15) Looking at the distribution $p(x|z)$, we notice it reflects the plot shown in part 1 in that members of each class are placed close to eachother. More interestingly we can better tell in this plot that classes similar to eachother are generally closer to eachother. This can be explained by the fact that similar classes share constructive components. By this meaning, we can generally deconstruct each class into a finite set of parts

and the classes with the biggest intersection are proportionally closer than the ones which are not.

Part 2

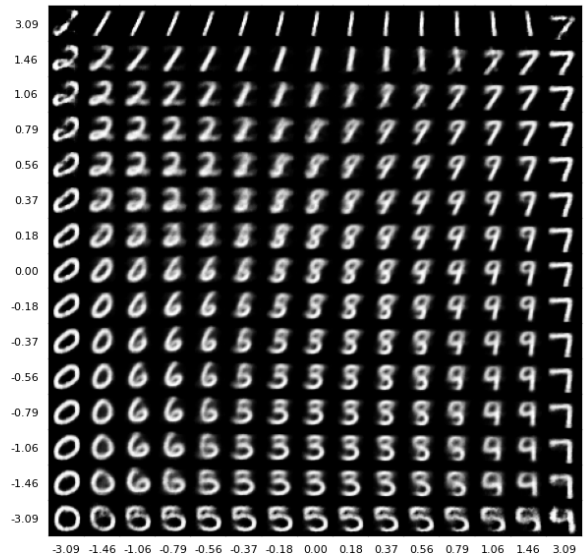

Figure 5: Visualization of latent space in terms of most likely image generations

A Week 1 Programming Exercise

```
1
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from scipy.stats import multivariate_normal
5
6  # Enforce reproducibility
7  np.random.seed(42)
8
9  # Parameters
10 theta_true = np.array([-1, 1])
11 sigma_y = 0.1  # Noise variance
12 prior_mean = np.zeros(2)
```

```
prior_cov = np.eye(2)
n_samples = 20

# Generate data
x = np.random.multivariate_normal(np.zeros(2), np.eye(2), n_samples)
y = x @ theta_true + np.random.normal(0, np.sqrt(sigma_y), n_samples)

# Plot data
plt.scatter(x[:, 0], x[:, 1], c=y, cmap='viridis', s=50, label='Data Points')
plt.colorbar(label='Labels (y)')
plt.title('Dataset for Regression')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()


posterior_cov = np.linalg.inv(np.linalg.inv(prior_cov) + (1 / sigma_y) * (x.T @ x))
posterior_mean = posterior_cov @ (x.T @ y) * (1 / sigma_y)

# Grid for plotting posterior in range [-3, 3]^2
theta_range = np.linspace(-3, 3, 100)
theta_grid = np.array(np.meshgrid(theta_range, theta_range)).reshape(2, -1).T

posterior_pdf = multivariate_normal(mean=posterior_mean, cov=posterior_cov).pdf(theta_grid)
posterior_pdf = posterior_pdf.reshape(100, 100)

# Plot posterior
plt.contourf(theta_range, theta_range, posterior_pdf, levels=50, cmap='viridis')
plt.colorbar(label='Posterior Density')
plt.title('Posterior Distribution of Theta')
plt.xlabel('Theta1')
plt.ylabel('Theta2')
plt.show()

# Grid for x in range [-3, 3]^2
x_grid = np.linspace(-3, 3, 50)
x_test = np.array(np.meshgrid(x_grid, x_grid)).reshape(2, -1).T

predictive_mean = x_test @ posterior_mean
predictive_variance = np.sum((x_test @ posterior_cov) * x_test, axis=1) + sigma_y

# Plot predictive variance
plt.contourf(x_grid, x_grid, predictive_variance.reshape(50, 50), levels=50, cmap='viridis')
plt.colorbar(label='Predictive Variance')
plt.title('Posterior Predictive Variance')
```

```
plt.xlabel('x1')
plt.ylabel('x2')
plt.show()


sigma_x = np.array([[0.1, 0], [0, 1]])
x_new = np.random.multivariate_normal(np.zeros(2), sigma_x, n_samples)
y_new = x_new @ theta_true + np.random.normal(0, np.sqrt(sigma_y), n_samples)

# Plot data
plt.scatter(x_new[:, 0], x_new[:, 1], c=y_new, cmap='viridis', s=50, label='Data Points')
plt.colorbar(label='Labels (y)')
plt.title('Dataset for Regression')
plt.xlabel('x1')
plt.ylabel('x2')
plt.legend()
plt.show()


xT_x_new = x_new.T @ x_new
posterior_cov_new = np.linalg.inv(np.linalg.inv(prior_cov) + (1 / sigma_y) * xT_x_new)
posterior_mean_new = posterior_cov_new @ (x_new.T @ y_new) * (1 / sigma_y)

# Compute density of posterior
posterior_pdf_new = multivariate_normal(mean=posterior_mean_new, cov=posterior_cov_new).pdf(
posterior_pdf_new = posterior_pdf_new.reshape(100, 100)

predictive_variance_new = np.sum((x_test @ posterior_cov_new) * x_test, axis=1) + sigma_y

fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Plot predictive variance
ax1 = axes[0]
contour1 = ax1.contourf(theta_range, theta_range, posterior_pdf_new, levels=50, cmap='viridi
fig.colorbar(contour1, ax=ax1, label='Posterior Density')
ax1.set_title('Posterior Distribution of Theta')
ax1.set_xlabel('Theta1')
ax1.set_ylabel('Theta2')


# Plot new predictive variance
ax2 = axes[1]
contour2 = ax2.contourf(x_grid, x_grid, predictive_variance_new.reshape(50, 50), levels=50,
fig.colorbar(contour2, ax=ax2, label='Predictive Variance')
ax2.set_title('Posterior Predictive Variance')
ax2.set_xlabel('x1')
```

```
ax2.set_ylabel('x2')

# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```

B Week 2 Programming exercise

```

import math
import torch
import torch.nn as nn
import numpy
import matplotlib.pyplot as plt
from torchvision import datasets, transforms
from tqdm import tqdm

train_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=True, download=True,
                   transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,)),
                       lambda x: x>0,
                       lambda x: x.float(),
            ])),
    batch_size=64, shuffle=True)
test_loader = torch.utils.data.DataLoader(
    datasets.MNIST('../data', train=False, transform=transforms.Compose([
                       transforms.ToTensor(),
                       transforms.Normalize((0.1307,), (0.3081,)),
                       lambda x: x>0,
                       lambda x: x.float(),
                   ])),
    batch_size=64, shuffle=True)


class VAE(nn.Module):
    def __init__(self, latent_dim = 2):
        super().__init__()

        self.latent_dim = latent_dim


        #The encoder serves as q(z|x)
```

```
        self.encoder = nn.Sequential(nn.Conv2d(in_channels=1, out_channels=32, kernel_size=3
                                     nn.BatchNorm2d(32),
                                     nn.ReLU(),

                                     nn.Conv2d(in_channels=32, out_channels=64, kernel_size=
                                     nn.BatchNorm2d(64),
                                     nn.ReLU(),


                                     #flatten before applying reparametrization trick
                                     nn.Flatten(start_dim=1) #bx64x7x7 -> bx128*7*7
                                    )

        self.mu = nn.Linear(64*7*7, latent_dim)
        self.log_var = nn.Linear(64*7*7, latent_dim)

        self.project_up = nn.Sequential(nn.Linear(2, 64*7*7),
                                        nn.BatchNorm1d(64*7*7),
                                        nn.ReLU()
                                       )
        self.decoder = nn.Sequential(nn.ConvTranspose2d(in_channels=64, out_channels=32, ker
                                     nn.BatchNorm2d(32),
                                     nn.ReLU(),

                                     nn.ConvTranspose2d(in_channels=32, out_channels=1, kerne
                                     nn.Sigmoid()
                                     )


    def encode(self, x):
        h = self.encoder(x) #compute hidden state
        mu = self.mu(h) #feed h into variable mu
        log_var = self.log_var(h) #feed h into variable log_var
        return mu, log_var

    def reparameterization(self, mu, log_var):
        epsilon = torch.randn_like(mu) #creating same shape as mu but drawn from unit gauss
        sigma = torch.exp(0.5*log_var)
        z = mu+sigma*epsilon
        return z

    def decode(self, z):
        z_projected = self.project_up(z).reshape(-1,64,7,7)
        x_reconstructed = self.decoder(z_projected)
        return x_reconstructed

```

```
83
84      def forward(self,x):
85
86          mu, log_var = self.encode(x)
87
88          z = self.reparameterization(mu, log_var)
89
90
91          x_reconstructed = self.decode(z)
92
93
94          return x_reconstructed, mu, log_var
95
96  def kl_divergence(mu, log_var):
97      pre_batch_average = 0.5*((torch.exp(log_var)+mu**2-1-log_var).sum(dim=1)) #summing over
98      return pre_batch_average.mean()
99
100
101 device = torch.device("cuda")
102 def train_epoch(model, device, optimizer, loss_function, lr, dataloader):
103     acc_loss = 0
104     total_samples = 1
105     model.train()
106
107
108     for (image_batch,_) in dataloader:
109             image_batch = image_batch.to(device)
110
111             prediction_batch, mu, log_var = model(image_batch)
112             reconstruction_loss = loss_function(prediction_batch, image_batch)
113             min_elbo_loss = reconstruction_loss + 0.001*kl_divergence(mu, log_var)
114
115             # print("reconstruction_loss",reconstruction_loss)
116             # print("kl_div",kl_divergence(mu, log_var), 0.001*kl_divergence(mu, log_var))
117             # print("elbo", min_elbo_loss,"\n")
118
119             acc_loss +=min_elbo_loss.item()*len(image_batch)
120             total_samples += len(image_batch)
121
122             optimizer.zero_grad()
123             min_elbo_loss.backward()
124             optimizer.step()
125
126
127     return acc_loss/total_samples
128
```

```
def val_epoch(model, device, loss_function, lr, dataloader):
    acc_loss = 0
    total_samples = 1
    model.eval()
    with torch.no_grad():
        for (image_batch,_) in dataloader:
            image_batch = image_batch.to(device)

            prediction_batch, mu, log_var = model(image_batch)

            reconstruction_loss = loss_function(prediction_batch, image_batch)
            min_elbo_loss = reconstruction_loss + kl_divergence(mu, log_var)
            acc_loss +=min_elbo_loss.item()*len(image_batch)
            total_samples += len(image_batch)

    return acc_loss/total_samples
def train_model(model, device, loss_function, lr, epochs, train_dataloader, val_dataloader):

    optimizer = torch.optim.AdamW(model.parameters(), lr = lr)
    loss_list = []
    for e in tqdm(range(epochs)):


        train_epoch_loss = train_epoch(model, device, optimizer, loss_function, lr, train_da
        val_epoch_loss = val_epoch(model, device, loss_function, lr, val_dataloader)
        print(train_epoch_loss, val_epoch_loss)

        loss_list.append((train_epoch_loss,val_epoch_loss))
    return model, loss_list


model, loss_list = train_model(VAE(2).to(device),device, nn.MSELoss(reduction="mean").to(dev

def visualize_prediction_distribution(model):
    model.eval()

    normal = torch.distributions.Normal(loc=0, scale=1)

    z_values = torch.linspace(0.001, 0.999, 15)
    z1 = normal.icdf(z_values)
    z2 = normal.icdf(z_values).sort(descending=True)[0]
    z1_grid, z2_grid = torch.meshgrid(z1,z2)
    z_pairs = torch.dstack([z1_grid,z2_grid]).transpose(0,1)

    fig, axes = plt.subplots(15,15, figsize=(7,7))
    plt.subplots_adjust(wspace=0, hspace=0)
```

```
175
176
177     z1 = z1.numpy()
178     z2 = z2.numpy()
179
180
181     for i in range(len(axes)):
182         for j in range(len(axes)):
183
184             axes[i,j].imshow(model.decode(z_pairs[i,j].to(device).unsqueeze(0)).detach().cpu
185             #ax.axis("off")
186
187             # Customize ticks only for outer subplots
188             if i == len(axes) - 1:  # Bottom row for z1 (x-axis)
189                 axes[i,j].set_xticks([14])  # Place a tick in the center (arbitrary)
190                 axes[i,j].set_xticklabels([f"{z1[j]:.2f}"], fontsize=8)
191             else:
192                 axes[i,j].set_xticks([])  # Remove ticks for inner subplots
193
194             if j == 0:  # First column for z2 (y-axis)
195                 axes[i,j].set_yticks([14])  # Place a tick in the center (arbitrary)
196                 axes[i,j].set_yticklabels([f"{z2[i]:.2f}"], fontsize=8, rotation=0)
197             else:
198                 axes[i,j].set_yticks([])  # Remove ticks for inner subplots
199
200             # Remove axis frame to match the visual style
201             axes[i,j].tick_params(left=False, bottom=False)
202
203     #plt.tight_layout()
204     plt.savefig("PML_assignment1_plot2_fpk297.png")
205     plt.show()
206
207
208 visualize_prediction_distribution(model)
209
210 normal = torch.distributions.Normal(loc=0, scale=1)
211
212 z_values = torch.linspace(0.001, 0.999, 15)
213 z1 = normal.icdf(z_values)
214 z2 = normal.icdf(z_values).sort(descending=True)[0]
215 z1_grid, z2_grid = torch.meshgrid(z1,z2)
216 z_pairs = torch.dstack([z1_grid,z2_grid]).transpose(0,1)
217 z_pairs.shape
218
219
220 def plot_latent_space(model, dataloader):
```

```
    z_points = []
    label_points = []
    for x, y in dataloader:
        x, y = x.to(device), y.to(device)

        mu, log_var = model.encode(x)
        z = model.reparameterization(mu,log_var)

        z_points.append(z)
        label_points.append(y)


    z_points = torch.cat(z_points, dim=0)
    label_points = torch.cat(label_points, dim=0)


    plt.scatter(z_points[:,0].detach().cpu().numpy(), z_points[:,1].detach().cpu().numpy(),
    plt.xlabel("$z_1$")
    plt.ylabel("$z_2$")
    plt.grid()
    plt.colorbar()
    plt.savefig("PML_assignment1_plot1_fpk297.png")
    plt.show()
plot_latent_space(model,test_loader)

```