Statistical and Machine Learning (01.113)
Homework 1

February 19, 2019.

1 Problem 1

Let $\theta \in \mathbb{R}^d$ be a fixed vector and θ_0 be a constant. Let $x \in \mathbb{R}^d$ be variable. Consider the hyperplane in $\mathbb{R}^d$ whose equation is given by $\langle \theta, x \rangle + \theta_0 = 0$. Given a point $y \in \mathbb{R}^d$, find the shortest distance from y to the hyperplane.
Hint: Normalize θ, i.e. let $n = \frac{\theta}{\|\theta\|}$ and rewrite the equation of the hyperplane in terms of n.

Solution. We can rewrite the equation of the hyperplane as $\langle n, x \rangle + \frac{\theta_0}{\|\theta\|} = 0$. Let x' be the point on the hyperplane which is closest to y, and note that

$$y = x' + dn,$$

where d is the displacement between y and x'; i.e. $d \geq 0$ if $\langle y, n \rangle \geq 0$ and $d < 0$ if $\langle y, n \rangle < 0$. Then we have

$$\begin{aligned} \langle y, n \rangle &= \langle x', n \rangle + d \langle n, n \rangle \\ \implies d &= \langle y, n \rangle - \langle x', n \rangle \\ \implies d &= \frac{\langle y, \theta \rangle + \theta_0}{\|\theta\|}. \end{aligned}$$

Thus, the shortest distance from y to the hyperplane is $\left| \frac{\langle y, \theta \rangle + \theta_0}{\|\theta\|} \right|$. ■

2 Problem 2

A continuous random variable X is said to have the *standard normal distribution*, with mean $\mu = 0$ and variance $\sigma^2 = 1$, that is $X \sim N(0, 1)$, if it has a probability density function (pdf) defined by

$$f_X(x) = \frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}, \qquad x \in \mathbb{R}.$$

Prove that

$$\int_{\mathbb{R}} f_X(x)\, \mathrm{d}x = 1.$$

Hint: Let $I = \int_{\mathbb{R}} e^{\frac{-x^2}{2}}\, \mathrm{d}x$. Express I^2 as a double integral over $\mathbb{R}^2$ and convert to polar coordinates.

Solution. Denoting $I = \int_{\mathbb{R}} e^{\frac{-y^2}{2}} \, \mathrm{d}y$, we obtain

$$\begin{aligned} I^2 &= \int_{\mathbb{R}} e^{\frac{-x^2}{2}} \, \mathrm{d}x \int_{\mathbb{R}} e^{\frac{-y^2}{2}} \, \mathrm{d}y \\ &= \int_{\mathbb{R}^2} e^{\frac{-(x^2+y^2)}{2}} \, \mathrm{d}x \, \mathrm{d}y. \end{aligned}$$

With $x = r\cos\theta$ and $y = r\sin\theta$, we have

$$\begin{aligned} I^2 &= \int_0^\infty \int_0^{2\pi} e^{\frac{-r^2}{2}} r \, \mathrm{d}\theta \, \mathrm{d}r \\ &= 2\pi \int_0^\infty r e^{\frac{-r^2}{2}} \, \mathrm{d}r \\ &= 2\pi \left[-e^{-\frac{r^2}{2}} \right]_0^\infty \\ &= 2\pi. \end{aligned}$$

∎

3 Problem 3

Let X and Y be random variables with a joint normal distribution such that $\mathbb{E}[X] = 0 = \mathbb{E}[Y]$, $\mathbb{E}\left[X^2\right] = 1 = \mathbb{E}\left[Y^2\right]$, and the covariance $\mathbb{E}[XY] = \rho$ where $0 < |\rho| < 1$.

(a) Write down the joint probability distribution $p(x, y)$ of X and Y.

(b) Let B denote the inverse of the covariance matrix of $[X, Y]^T$. Perform the decomposition $B = PDP^{-1}$, where D is a diagonal matrix and P is an orthogonal matrix.

(c) Use the result above to transform $(x, y) \to (u, v)$ such that under the new coordinates, the joint distribution can be factorized; i.e. $q(u, v) = q_1(u)q_2(v)$.

Hint: For (b), compute the eigenvalues and eigenvectors of B. For (c), recall the definition of the adjoint of a matrix.

Solution.

(a) $p(x, y) = \frac{1}{2\pi} e^{-\frac{1}{2(1-\rho^2)} \left\langle \begin{bmatrix} x \\ y \end{bmatrix}, \frac{1}{\sqrt{1-\rho^2}} \begin{bmatrix} 1 & -\rho \\ -\rho & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \right\rangle} = \frac{1}{2\pi\sqrt{1-\rho^2}} e^{-\frac{(x^2 - 2\rho xy + y^2)}{2(1-\rho^2)}}$

(b) We have

$$B = \frac{1}{1-\rho^2} \begin{bmatrix} 1 & -\rho \\ -\rho & 1 \end{bmatrix}.$$

Since B is symmetric, and has eigenvalues $\frac{1+\rho}{1-\rho^2} = \frac{1}{1-\rho}$ and $\frac{1-\rho}{1-\rho^2} = \frac{1}{1+\rho}$ with corresponding orthonormal eigenvectors $\left[-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]^T$ and $\left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]^T$, we can decompose B as

$$B = \begin{bmatrix} \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} \frac{1}{1-\rho} & 0 \\ 0 & \frac{1}{1+\rho} \end{bmatrix} \begin{bmatrix} \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}^{-1} =: PDP^{-1}.$$

(c) Hence

$$\begin{aligned} e^{-\frac{1}{2}\langle \mathbf{x}, B\mathbf{x}\rangle} &= e^{-\frac{1}{2}\left\langle \mathbf{x}, PDP^{-1}\mathbf{x}\right\rangle} \\ &= e^{-\frac{1}{2}\left\langle P^T\mathbf{x}, DP^{-1}\mathbf{x}\right\rangle} \\ &= e^{-\frac{1}{2}\langle \mathbf{u}, D\mathbf{u}\rangle}, \end{aligned}$$

where $\mathbf{x} = [x, y]^T$ and $\mathbf{u} = [u, v]^T = P^T\mathbf{x} = P^{-1}\mathbf{x}$.
Thus under the change of coordinates $\mathbf{u} = P^T\mathbf{x}$, the probability distribution becomes

$$\begin{aligned} q(u, v) &= \frac{1}{2\pi\sqrt{1-\rho^2}} e^{-\frac{1}{2}\langle \mathbf{u}, D\mathbf{u}\rangle} \\ &= \left(\frac{1}{\sqrt{2\pi}\sqrt{1-\rho}}\right)\left(\frac{1}{\sqrt{2\pi}\sqrt{1+\rho}}\right) e^{-\frac{1}{2}\langle \mathbf{u}, D\mathbf{u}\rangle} \\ &= \left(\frac{1}{\sqrt{2\pi}\sqrt{1-\rho}}\right) e^{-\frac{u^2}{2(1-\rho)}} \left(\frac{1}{\sqrt{2\pi}\sqrt{1+\rho}}\right) e^{-\frac{v^2}{2(1+\rho)}} \\ &= q_1(u)q_2(v). \end{aligned}$$

■

4 Problem 4

We will now use PyTorch to perform linear regression using gradient descent. Import the Boston data from *sklearn* datasets to generate a linear model that predicts the prices of houses (*MEDV*) using three inputs:

(i) average number of rooms per dwelling (*RM*);

(ii) index of accessibility to radial highways (*RAD*);

(iii) per capita crime rate by town (*CRIM*).

You can access the selected inputs and target variables using the following code:

```
import matplotlib.pyplot as plt    #To generate the plots of question (d)
import numpy
csv = 'https://www.dropbox.com/s/0rjqoaygjbk3sp8/boston_house_prices_3features.txt?dl=1'
data = numpy.genfromtxt(csv, delimiter=',', skip_header=1)
```

The data contains 506 observations on housing prices for Boston suburbs. The first three columns corresponds to the inputs RM, RAD and $CRIM$, respectively. The last column is the target $MEDV$.

Import PyTorch and format the data as follows:

```
import torch
# Convert inputs and target to tensors
inputs = data[:, [0,1,2]]
inputs = inputs.astype(numpy.float32)
inputs = torch.from_numpy(inputs)
```

```
target =  data[:,3]
target = target.astype(numpy.float32)
target = torch.from_numpy(target)
```

(a) Write the code to generate (random) weights $w_{\text{RM}}, w_{\text{RAD}}, w_{\text{CRIM}}$ and bias b. After that, write a function to compute the linear model.

(b) Write a function that computes the mean squared error (MSE).

(c) Complete the loop below to update the weights and bias using a fixed learning rate (try different values from 0.01 to 0.0001) over 200 iterations/epochs.

```
for i in range(200):
        print("Epoch", i, ":")

#       compute the model predictions
#       compute the loss and its gradient

        print("Loss=", loss)

        with torch.no_grad():

#               update the weights
#               update the bias

                w.grad.zero_()
                b.grad.zero_()
```

(Note that we reset the gradients to zero by using *w.grad.zero_()* and *b.grad.zero_()* because PyTorch accumulates gradients.)

(d) Use the matplotlib library to plot the evolution of MSE at every iteration.

Upload the final script in your Dropbox folder and name it as "**HW1.py**".

Solution.

```
## Import pacakges and data set
import torch
import numpy
import matplotlib.pyplot as plt
##########
# Convert inputs and targets in tensors
csv = 'https://www.dropbox.com/s/0rjqoaygjbk3sp8/boston_house_prices_3features.txt?dl=1'
data = numpy.genfromtxt(csv,delimiter=',', skip_header=1)

inputs = data[:, [0,1,2]]
inputs = inputs.astype(numpy.float32)
inputs = torch.from_numpy(inputs)

target =  data[:,3]
target = target.astype(numpy.float32)
```

```
target = torch.from_numpy(target)

# A) Start with the given weights
torch.manual_seed(2)     # We set a seed to  make the computations reproducible
w = torch.randn(1, 3, requires_grad=True)
b = torch.randn(1, requires_grad=True)
print(w)
print(b)
# Define the linear model and the loss function (mean squared error)
def linearModel(x):
        return torch.mm(x, w.t()) + b
# B) Function that computs MSE
def mse(x, y):
        diff = x - y
        return torch.sum(diff * diff) / diff.numel()

# Predict the target variable
print("Before training: ")
prediction = linearModel(inputs)
print("Prediction: ", prediction)

# Calculate the loss function and the values of the weights
loss = mse(prediction, target)
print(loss)
print(w)
print(b)

########## C)

# Step 1. Define the lerning rate, train for 200 epochs and compute the loss w.r.t. to the weights
lrate = 0.0001
for i in range(200):
        print("Epoch ", i, ":")
        preds = linearModel(inputs)
        loss = mse(preds, target)
        print("Loss =  ", loss)
        loss.backward()
        with torch.no_grad():
                w -= lrate * w.grad
                b -= lrate * b.grad
                w.grad.zero_()
                b.grad.zero_()

# Step 2. Predict the target variable
print("After training: ")
prediction = linearModel(inputs)
print("Prediction: ", prediction)

#Step 3. Calculate the new loss function and the new values of the weights
loss = mse(prediction, target)
print(loss)
print(w)
print(b)

#D) Generate the plots
loss_history = []
for i in range(200):
    print("Epoch", i, ":")
    preds =  linearModel(inputs)
    loss = mse(preds,target)
```

```
    loss.backward()
    loss_history.append(loss)
    print ("Loss =", loss)
    with torch.no_grad():
        w -= lrate* w.grad
        b -= lrate* b.grad
        w.grad.zero_()
        b.grad.zero_()

plt.plot(loss_history)
plt.xlabel('Epoch')
plt.ylabel('Loss (MSE)')
```

■