# HW-1

- Subject Name: Deep Learning
- Subject Code: CS 7150
- Professor Name: Jiaji Liu
- Student Name: Varun Guttikonda
- NUID: 002697400

## Problem 1

Consider a 2 dimensional random Gaussian Vector:

$$x \equiv \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim \mathcal{N}(\mu, \Sigma)$$

where

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_1 & \sigma_2^2 \end{bmatrix}$$

and $-1 < \rho < 1$

Before answering the questions below, I would like to formulate the probability density function of the random vector $x$. Let $k$ be the dimension of the random vector $x$, then the probability density function of $x$ is given by:

$$p(x) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

where $|\Sigma|$ is the determinant of the covariance matrix $\Sigma$, and $\mu$ is the mean of the random vector $x$ which is of the dimension $k$.

In our case, the dimension of the random vector $x$ is 2, and the covariance matrix $\Sigma$ is given by:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_1 & \sigma_2^2 \end{bmatrix}$$

The determinant of the covariance matrix $\Sigma$ is given by:

$$|\Sigma| = \sigma_1^2\sigma_2^2(1-\rho^2)$$

## Question 1

Derive the joint entropy $H(x)$

Answer:

The general formula for the entropy is given by

$$H(x) = -\sum_{i=1}^{n} p(x_i)\log p(x_i)$$

For now, lets take $x$ to be our vector, so

$$H(\vec{x}) = -\sum_{i=1}^{n} p(\vec{x_i})\log p(\vec{x_i})$$

I would like to use the vector notation for calculations as the final result will still be a vector. We can take the norm or the sum of the vector to get the final result as a scalar.

$$H(\vec{x}) = -p(\vec{x})\log p(\vec{x})$$

We have the probability density function of the random vector $x$ as:

$$p(x) = \frac{1}{\sqrt{(2\pi)^k|\Sigma|}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

Applying a log on both sides, we get:

$$\log p(x) = \log \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} + \log \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

We can simpiify this further as:

$$\log p(x) = \log \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} - \frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)$$

Now substituting this in the entropy equation, we get:

$$H(\vec{x}) = -p(\vec{x}) \log p(\vec{x})$$

$$H(\vec{x}) = -\frac{1}{\sqrt{(2\pi)^k |\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right) \left(\log \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} - \frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

We can further simplify this but I would like to leave it as it is for now as we don't have the values of $\mu$ and $\Sigma$ and this is very easy to read and compute in this form.

## Question 2

Derive the mutual information $I(x_1; x_2)$

Answer:

The mutual information is given by:

$$I(x_1; x_2) = H(x_1) + H(x_2) - H(x_1, x_2)$$

We have already derived the joint entropy $H(x)$ in the previous question. We can use that to derive the mutual information.

$$I(x_1; x_2) = H(x_1) + H(x_2) - H(x_1, x_2)$$

$$I(x_1; x_2) = -p(x_1) \log p(x_1) - p(x_2) \log p(x_2) + p(x_1, x_2) \log p(x_1, x_2)$$

Let's leave the equation here and not simplify it further. We don't know if the components of the vector are independent or not. If they are independent, then the joint probability will be the product of the individual probabilities. If they are not independent, then we will have to use the joint probability function to compute the mutual information. In either case, we can use the equation in question 1 to compute the mutual information.

### Question 3

If $\rho$ can be varied, when is the joint entropy maximized? What is the mutual information $x_1$ and $x_2$ in this case?

Answer:

The joint entropy is maximized when the covariance matrix $\Sigma$ is a diagonal matrix. This is because the determinant of the covariance matrix $\Sigma$ is given by:

$$|\Sigma| = \sigma_1^2 \sigma_2^2 (1 - \rho^2)$$

If $\Sigma$ is a diagonal matrix, then $\rho = 0$ and the determinant of the covariance matrix $\Sigma$ is given by:

$$|\Sigma| = \sigma_1^2 \sigma_2^2$$

## Problem 2

We talked about cross-entropy in the class. That is

$$H(p,\ q) = -\mathbb{E}_{z \sim p}[\log q(z)]$$

In a $C$-class image classification problem, we have $N$ samples. Denote the $i$-th sample as $(x_i, y_i)$ where $x_i$ is the image and $y_i \in \{0, 1, \ldots, C-1\}$ is its class label. Suppose our model predicts class probability as $\hat{p}(y)$ where $y \in \{0, 1, \ldots, C-1\}$. The cross-entropy loss for the $i$-th sample is

$$L_i = -\log \hat{p}(y_i)$$

### Question 1

Show that an empirical estimate of cross entropy is

$$\frac{1}{N}\sum_{i=1}^{N} -\log \hat{p}(y_i)$$

Answer:

For each sample, we have the cross entropy loss as:

$$L_i = -\log \hat{p}(y_i)$$

For an empirical estimate of cross entropy, we need to take the average of the cross entropy loss for all the samples. This is given by:

$$\frac{1}{N}\sum_{i=1}^{N} -\log \hat{p}(y_i)$$

So, we have the empirical estimate of cross entropy as:

$$\frac{1}{N}\sum_{i=1}^{N} -\log \hat{p}(y_i)$$

## Question 2

Let us denote the above cross-entropy loss as $l$. We often report $l$ as an indicator of model quality. A model with lower $l$ is more accurate. However, in some applications, we also report a related metric, called *perplexity** (PPL), defined as

$$PPL = e^l$$

Show that

1. A perfect model has $PPL = 1$

Answer:

For any perfect model, the cross entropy loss will be 0. So, we have:

$$PPL = e^l = e^0 = 1$$

1. Any reasonable model has $PPL < C$

Answer:

I am assuming that the $C$ in the equation is a constant and not the number of classes. With that out of the way, let's take a logarithm on both sides of the equation:

$$PPL = e^l$$

$$\log PPL = \log e^l$$

$$\log PPL = l$$

$$\log PPL = \frac{1}{N} \sum_{i=1}^{N} - \log \hat{p}(y_i)$$

Now we know that the values of $\log PPL$ and $l$ are always negative. So, we can take the absolute value of both sides of the equation. This will not change the inequality. And since we know that the values of $p(y_i)$ are bounded, the value of the log of $p(y_i)$ will also be bounded. So, the entire value of the summation will be bounded and doesn't grow with the number of samples. So, we can say that the value of $l$ is bounded. This means that the value of $PPL$ is also bounded. So, we can say that any reasonable model has $PPL < C$.

# Problem 3

Install `numpy`, `matplotlib`, and `pytorch` or `tensorflow` in your laptop or desktop. Attach your code for sub-problems below.

## Question 1

Plot the following function on a mesh grid defined upon $-1 \leq x_1, x_2 \leq 1$

$$f(x) = \frac{1}{2}\left\| \begin{bmatrix} 2 & 1 \\ -1 & 3 \\ 0 & -2 \end{bmatrix} x - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right\|_2^2$$

## Answer 1

Before we can plot the function, let's simplify the function so that it can be represented as a simple equation. We can do this by expanding the norm and simplifying the equation.

$$f(x) = \frac{1}{2}\left\| \begin{bmatrix} 2 & 1 \\ -1 & 3 \\ 0 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right\|_2^2$$

$$f(x) = \frac{1}{2}\left\| \begin{bmatrix} 2x_1 + x_2 \\ -x_1 + 3x_2 \\ 0x_1 - 2x_2 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \right\|_2^2$$

$$f(x) = \frac{1}{2}\left\| \begin{bmatrix} 2x_1 + x_2 - 1 \\ -x_1 + 3x_2 \\ -2x_2 \end{bmatrix} \right\|_2^2$$

$$f(x) = \frac{1}{2}(\sqrt{(2x_1 + x_2 - 1)^2 + (-x_1 + 3x_2)^2 + (-2x_2)^2})^2$$

$$f(x) = \frac{1}{2}((2x_1 + x_2 - 1)^2 + (-x_1 + 3x_2)^2 + (-2x_2)^2)$$

which simplifies to

$$f(x) = \frac{1}{2}(5x_1^2 + 14x_2^2 - 4x_1 - 2x_2 - 2x_1x_2 + 1)$$

In [ ]:
```
import warnings
```

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
from tensorflow import keras

warnings.filterwarnings("ignore")
```

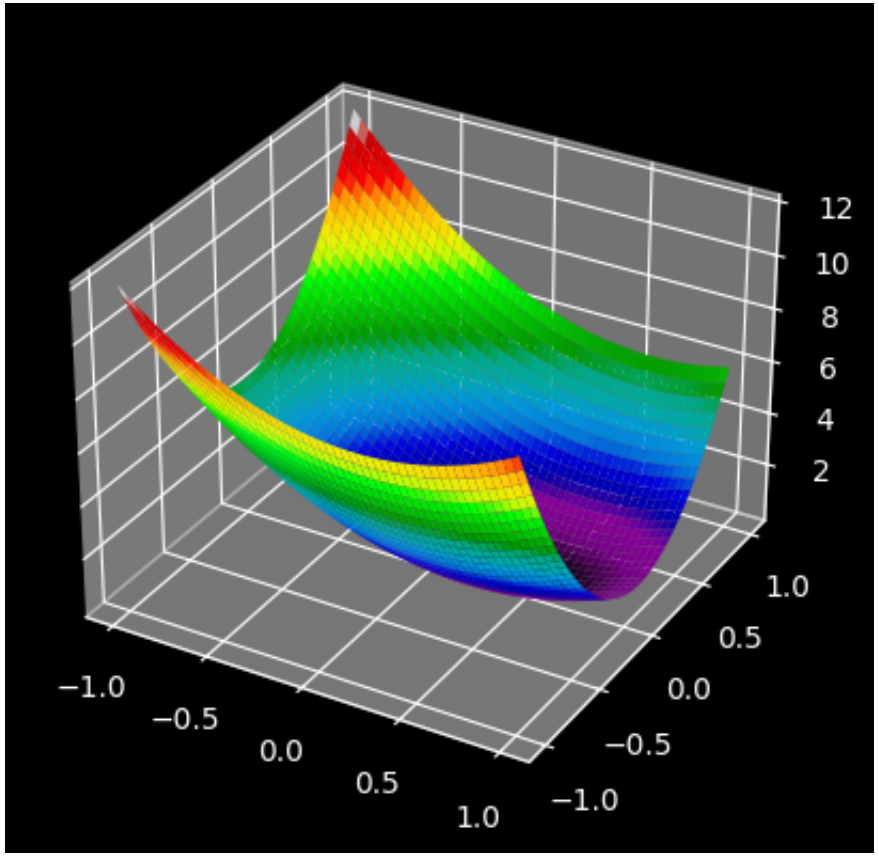
In [ ]:

```
# Create values for x1 and x2
x1 = np.linspace(-1, 1, 1000)
x2 = np.linspace(-1, 1, 1000)

# Create a meshgrid for evaluation and plotting
X1, X2 = np.meshgrid(x1, x2)

# Create the function to be evaluated
f_x = ((5 * X1**2) + (14 * X2**2) - (4 * X1) - (2 * X2) - (2 * X1 * X2) + 1) / 2

# Plot the function
fig, axs = plt.subplots(subplot_kw={"projection": "3d"})
axs.plot_surface(X1, X2, f_x, cmap="nipy_spectral", edgecolor="none")
plt.show()
```




## Question 2

Use Pytorch or Tensorflow autograd ability to find the gradient at $x = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$

In [ ]:
```
# Create variables
x1 = tf.Variable(1.0)
x2 = tf.Variable(-1.0)

# Create a gradient tape and start watching the variables
with tf.GradientTape(persistent=True) as tape:
    tape.watch(x1)
    tape.watch(x2)
```

```
    # Create the function to be evaluated
    f_x = ((5 * x1**2) + (14 * x2**2) - (4 * x1) - (2 * x2) - (2 * x1 * x2) + 1) / 2

# Calculate the gradients
variables = {"x1": x1, "x2": x2}

gradient = {x: y.numpy() for x, y in tape.gradient(f_x, variables).items()}

# Print the gradients
print("Gradient of x1: {}".format(gradient["x1"]))
print("Gradient of x2: {}".format(gradient["x2"]))
```

```
Gradient of x1: 4.0
Gradient of x2: -16.0
```

## Question 3

Write a program using Pytorch or Tensorflow's autograd ability to search for the minimmer of $f(x)$. Call it $x^*$. Also report the corresponding $f(x^*)$.

In [ ]:
```
# Create variables
x1 = tf.Variable(1.0)
x2 = tf.Variable(-1.0)

# Create a gradient tape and start watching the variables
with tf.GradientTape(persistent=True) as tape:
    tape.watch(x1)
    tape.watch(x2)

    # Create the function to be evaluated
    f_x = ((5 * x1**2) + (14 * x2**2) - (4 * x1) - (2 * x2) - (2 * x1 * x2) + 1) / 2

# Print the old values
print("Old value of x1: {}".format(x1.numpy()))
print("Old value of x2: {}".format(x2.numpy()))
print("Old value of f_x: {}".format(f_x.numpy()))

# Create an optimizer and minimize the function
optimizer = keras.optimizers.SGD(learning_rate=0.1)
```

```
optimizer.minimize(f_x, [x1, x2], tape=tape)

f_x_star = (
    (5 * x1**2) + (14 * x2**2) - (4 * x1) - (2 * x2) - (2 * x1 * x2) + 1
) / 2

# Print the new values
print("New value of x1: {}".format(x1.numpy()))
print("New value of x2: {}".format(x2.numpy()))
print("New value of f_x: {}".format(f_x_star.numpy()))
```

```
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, plea
se use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.SGD`.
Old value of x1: 1.0
Old value of x2: -1.0
Old value of f_x: 10.0
New value of x1: 0.6000000238418579
New value of x2: 0.6000000238418579
New value of f_x: 1.7599999904632568
```