Name: Prithul Sarker

Course: CS 687 Fundamentals of Deep Learning

Homework 2

1.

f(x) = $\begin{bmatrix} x_0 \sin(x_2) + x_1 \sin(x_0) + x_2 \sin(x_1) \\ \sin(x_0)\cos(x_1) + \sin(x_1)\cos(x_2) + \sin(x_2)\cos(x_0) \end{bmatrix}$

Jacobian of f(x) =

$$\begin{bmatrix} \frac{\partial(x_0*\sin(x_2)+x_1*\sin(x_0)+x_2*\sin(x_1))}{\partial x_0} & \frac{\partial(x_0*\sin(x_2)+x_1*\sin(x_0)+x_2*\sin(x_1))}{\partial x_1} & \frac{\partial(x_0*\sin(x_2)+x_1*\sin(x_0)+x_2*\sin(x_1))}{\partial x_2} \\ \frac{\partial(\sin(x_0)*\cos(x_1)+\sin(x_1)*\cos(x_2)+\sin(x_2)*\cos(x_0))}{\partial x_0} & \frac{\partial(\sin(x_0)*\cos(x_1)+\sin(x_1)*\cos(x_2)+\sin(x_2)*\cos(x_0))}{\partial x_1} & \frac{\partial(\sin(x_0)*\cos(x_1)+\sin(x_1)*\cos(x_2)+\sin(x_2)*\cos(x_0))}{\partial x_2} \end{bmatrix}$$

=

$$\begin{bmatrix} \sin(x_2) + x_1\cos(x_0) & \sin(x_0) + x_2\cos(x_1) & x_0\cos(x_2) + \sin(x_1) \\ \cos(x_0)\cos(x_1) - \sin(x_2)\sin(x_0) & \cos(x_1)\cos(x_2) - \sin(x_0)\sin(x_1) & \cos(x_0)\cos(x_2) - \sin(x_1)\sin(x_2) \end{bmatrix}$$

2. (a)

$Z_1 = W_1 * x + b_1$

$Z = W_2 * Z_1 + b_2$

2. (b)

$Z_1 = [\ -7 \ \ 2 \ \ 12]^T$

$Z = [\ -1 \ \ 6]^T$

2. (c)

Since the second row has higher value, the classifier should label the input as second class.

2. (d)

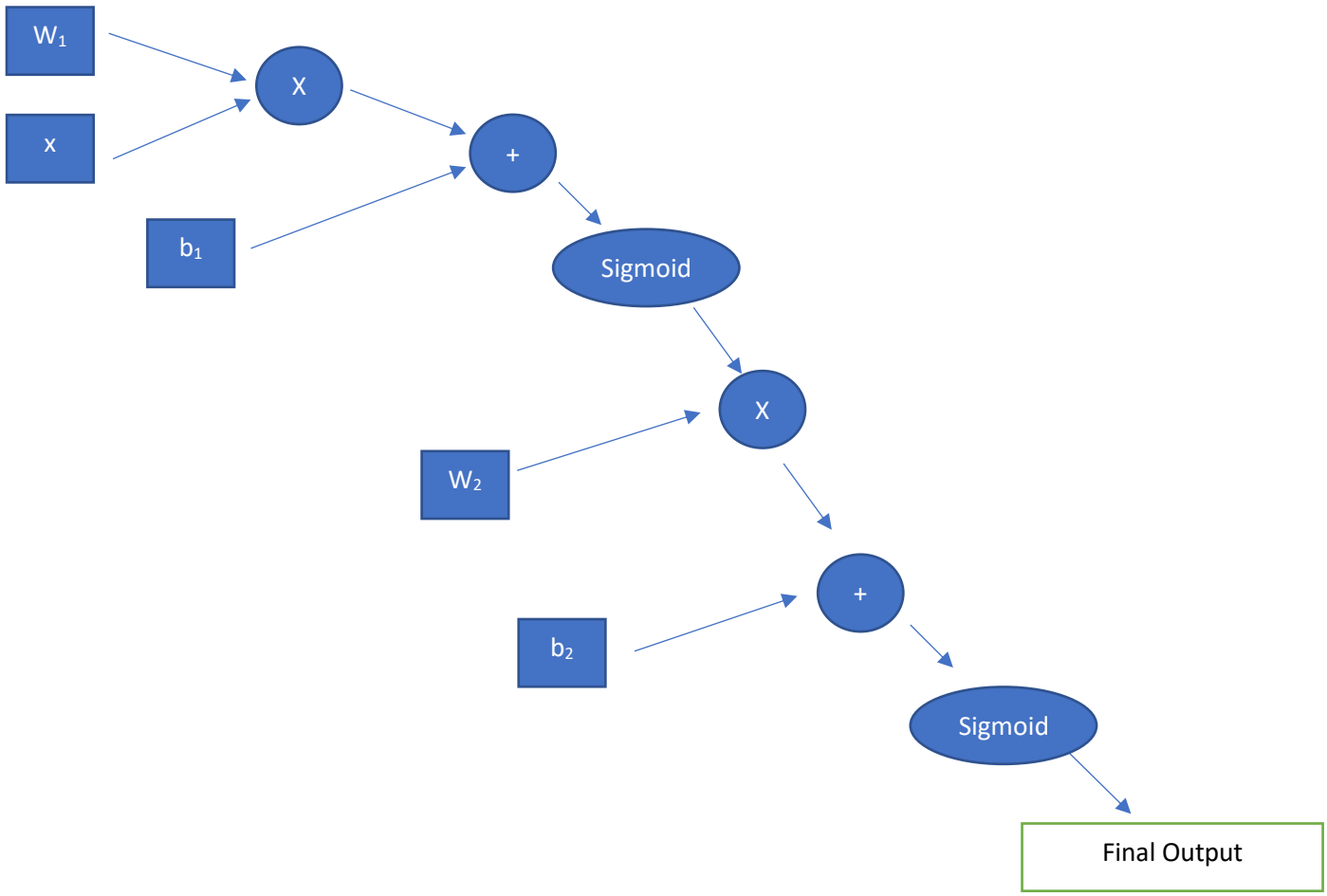
Hinge loss = max(0, 6 -(-1) + 1) = max(0, 8) = 8

2. (e)

$P(Y = \text{first class} \mid X = x_i) = \frac{e^{-1}}{e^{-1}+e^{6}} = 0.000911$

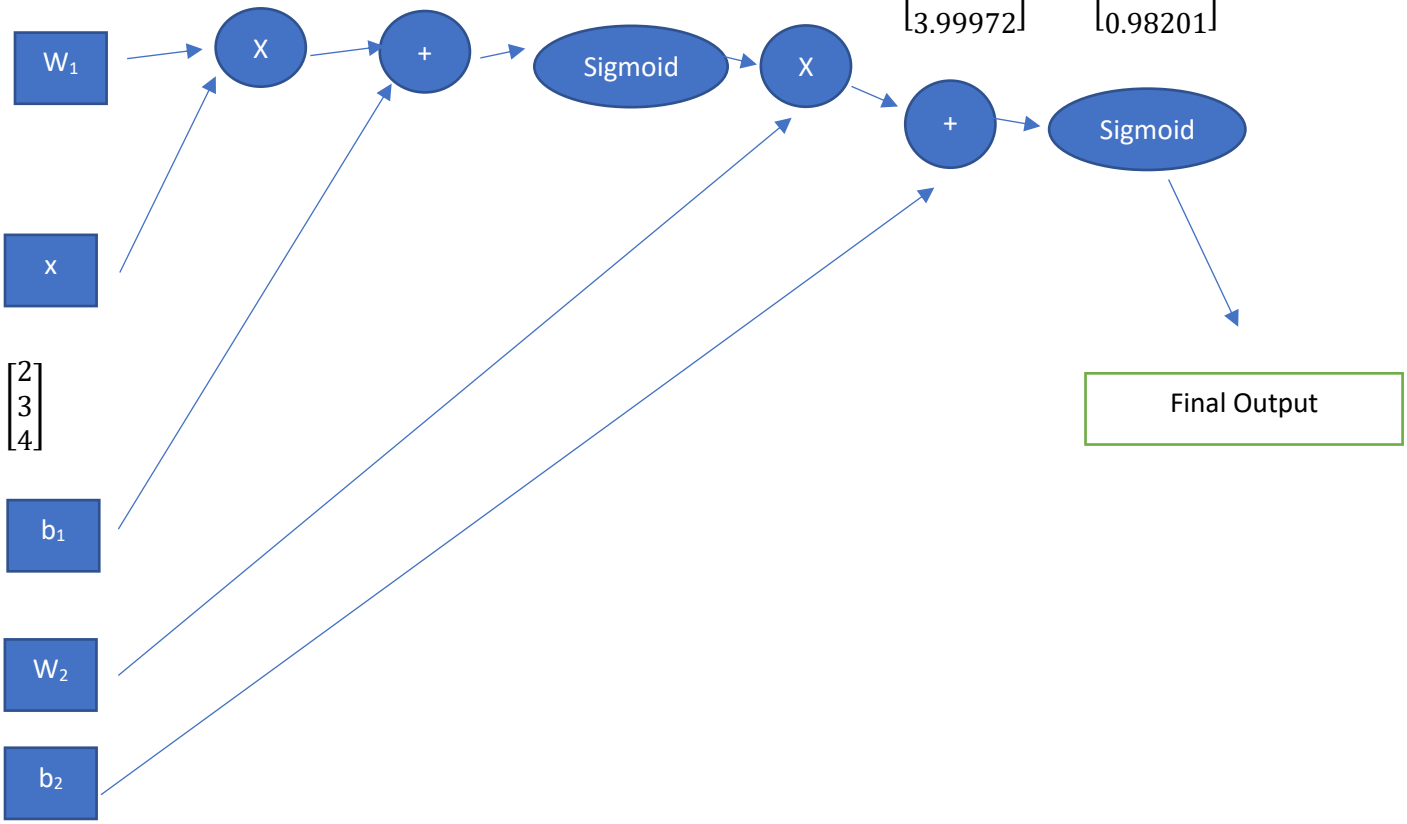
Softmax loss = - ln (0.000911) = 7

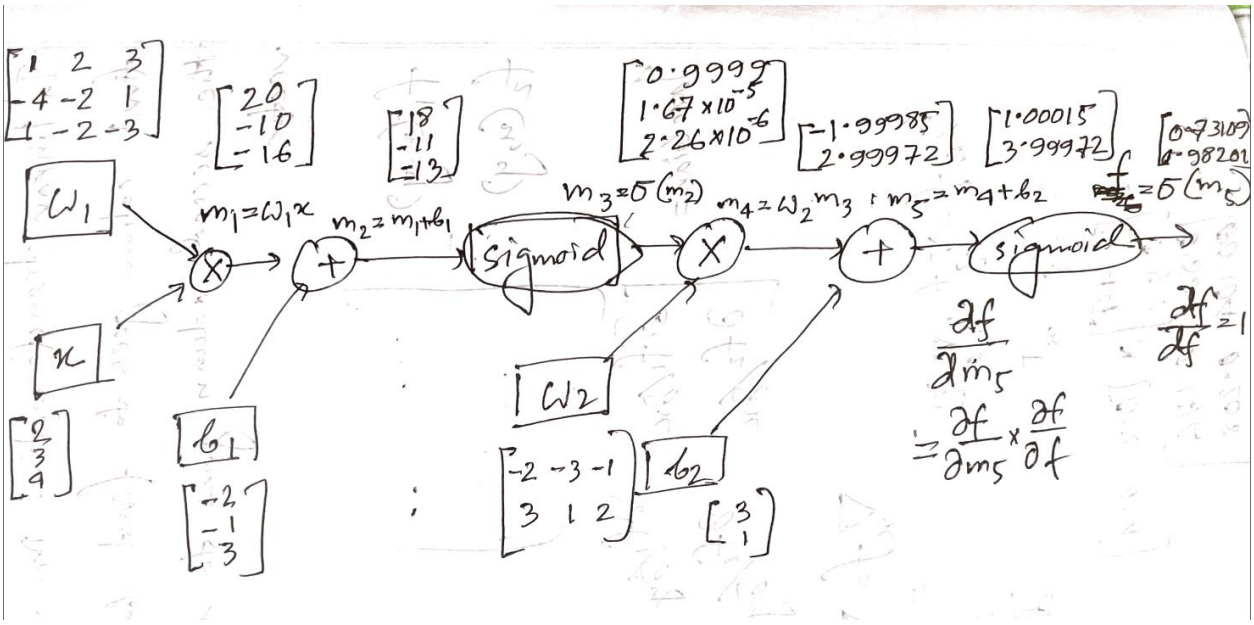
3. (a)



3. (b)

$\begin{bmatrix} 1 & 2 & 3 \\ -4 & -2 & 1 \\ 1 & -2 & -3 \end{bmatrix}$ $\begin{bmatrix} 20 \\ -10 \\ -16 \end{bmatrix}$ $\begin{bmatrix} 18 \\ -11 \\ -13 \end{bmatrix}$ $\begin{bmatrix} 0.9999 \\ 1.67\ e-5 \\ 2.26\ e-6 \end{bmatrix}$ $\begin{bmatrix} -1.99985 \\ 2.99972 \end{bmatrix}$ $\begin{bmatrix} 1.00015 \\ 3.99972 \end{bmatrix}$ $\begin{bmatrix} 0.73109 \\ 0.98201 \end{bmatrix}$





3. (c)

$$\frac{df}{df} = 1$$

$f = \sigma(m_5)$ where $m_5 = \begin{bmatrix} 1.00015 \\ 3.99972 \end{bmatrix}$

$$= \begin{bmatrix} 0.73109 \\ 0.98201 \end{bmatrix}$$

$$\frac{\partial f}{\partial m_5} = \sigma(m_5) \cdot (1 - \sigma(m_5))$$

$$= \begin{bmatrix} 0.73109\,(1 - 0.73109) \\ 0.98201\,(1 - 0.98201) \end{bmatrix}$$

$$= \begin{bmatrix} 0.196597 \\ 0.017666 \end{bmatrix}$$

$$m_5 = m_4 + b_2 = \begin{bmatrix} 1.00015 \\ 3.99972 \end{bmatrix}$$

$$\frac{\partial m_5}{\partial m_4} = m_4 = \begin{bmatrix} -1.99985 \\ 2.99972 \end{bmatrix}$$

$$\frac{\partial f}{\partial m_4} = \begin{bmatrix} \frac{\partial f_{11}}{\partial m_{4,1}} \\ \frac{\partial f_{21}}{\partial m_{42}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_{11}}{\partial m_{5_{11}}} \times \frac{\partial m_{5_{11}}}{\partial m_{4\,11}} \\ \frac{\partial f_{21}}{\partial m_{5_{21}}} \times \frac{\partial m_{5_{21}}}{\partial m_{4\,21}} \end{bmatrix}$$

$$= \begin{bmatrix} 0.196597 \times (-1.99985) \\ 0.017666 \times 2.99972 \end{bmatrix} = \begin{bmatrix} -0.39316 \\ 0.05299 \end{bmatrix}$$

$$m_4 = W_2 m_3$$

$$\frac{\partial m_4}{\partial m_3} = W_2^T = \begin{bmatrix} -2 & 3 \\ -3 & 1 \\ -1 & 2 \end{bmatrix}$$

$$\frac{\partial f}{\partial m_3} = \frac{\partial m_4}{\partial m_3} * \frac{\partial f}{\partial m_4} = \begin{bmatrix} -2 & 3 \\ -3 & 1 \\ -1 & 2 \end{bmatrix} \times \begin{bmatrix} -0.39316 \\ 0.05299 \end{bmatrix}$$

$$= \begin{bmatrix} 0.94529 \\ 1.2325 \\ 0.49914 \end{bmatrix}$$

$$\frac{\partial m_4}{\partial W_2} = m_3^T = \begin{bmatrix} 0.9999 & 1.67 \times 10^{-5} & 2.26 \times 10^{-6} \end{bmatrix}$$

$$\frac{\partial f}{\partial W_2} = \frac{\partial f}{\partial m_4} \times \frac{\partial m_4}{\partial W_2}$$

$$= \begin{bmatrix} -0.39316 \\ 0.05299 \end{bmatrix} \times \begin{bmatrix} 0.9999 & 1.67 \times 10^{-5} & 2.26 \times 10^{-6} \end{bmatrix}$$

$$= \begin{bmatrix} -0.39312 & 6.5658 \times 10^{-6} & -8.8854 \times 10^{-7} \\ 0.052985 & 8.8493 \times 10^{-7} & 1.1976 \times 10^{-7} \end{bmatrix}$$

$$m_3 = \sigma(m_2) = \begin{bmatrix} 0.9999 \\ 1.67 \times 10^{-5} \\ 2.26 \times 10^{-6} \end{bmatrix}$$

$$\frac{\partial m_3}{\partial m_2} = \sigma(m_2)\left(1 - \sigma(m_2)\right)$$

$$= \begin{bmatrix} 0.9999\,(1-0.9999) \\ 1.67\times10^{-5}\,(1-1.67\times10^{-5}) \\ 2.26\times10^{-6}\,(1-2.26\times10^{-6}) \end{bmatrix}$$

$$= \begin{bmatrix} 9.999 \times 10^{-5} \\ 1.66997 \times 10^{-5} \\ 2.25999 \times 10^{-6} \end{bmatrix}$$

$$\frac{\partial f}{\partial m_2} = \frac{\partial f}{\partial m_3} \times \frac{\partial m_3}{\partial m_2} = \begin{bmatrix} 9.45195 \times 10^{-5} \\ 2.05824 \times 10^{-5} \\ 1.12805 \times 10^{-6} \end{bmatrix}$$

$$m_2 = m_1 + b_1$$

$$\frac{\partial m_2}{\partial m_1} = m_1 = \begin{bmatrix} 20 \\ -10 \\ -16 \end{bmatrix}$$

$$\frac{\partial f}{\partial m_1} = \begin{bmatrix} 1.89039 \times 10^{-3} \\ -2.05824 \times 10^{-4} \\ -1.80488 \times 10^{-5} \end{bmatrix}$$

$$\left[\frac{\partial f}{\partial m_1} = \frac{\partial f}{\partial m_2} \times \frac{\partial m_2}{\partial m_1}\right]$$

[element wise multiplication]

$$m_1 = w_1 x$$

$$\frac{\partial m_1}{\partial w_1} = x^T$$

$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial m_1} \times \frac{\partial m_1}{\partial w_1}$$

$$= \begin{bmatrix} 1.89039 \times 10^{-3} \\ -2.05824 \times 10^{-4} \\ -1.80488 \times 10^{-5} \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 4 \end{bmatrix}$$

$$= \begin{bmatrix} 0.0037808 & 0.0056712 & 0.0075616 \\ -0.00041165 & -0.00061747 & -0.00082330 \\ -0.000036098 & -0.000054146 & -0.000072195 \end{bmatrix}$$

$$\nabla_{w_1} L = \begin{bmatrix} 3.7808 \times 10^{-3} & 5.6712 \times 10^{-3} & 7.5616 \times 10^{-3} \\ -4.1165 \times 10^{-4} & -6.1747 \times 10^{-4} & -8.233 \times 10^{-4} \\ -3.6098 \times 10^{-5} & -5.4146 \times 10^{-5} & -7.2195 \times 10^{-5} \end{bmatrix}$$

$$\nabla_{w_2} L = \begin{bmatrix} -0.39312 & 6.5658 \times 10^{-6} & -8.8854 \times 10^{-7} \\ 0.052985 & 8.8493 \times 10^{-7} & 1.1976 \times 10^{-7} \end{bmatrix}$$

$$\nabla_{w1} L = \begin{bmatrix} 3.7808x10^{-3} & 5.6712x10^{-3} & 7.5616x10^{-3} \\ -4.1165x10^{-4} & -6.1747x10^{-4} & -8.233x10^{-4} \\ -3.6098x10^{-5} & -5.4146x10^{-5} & -7.2195x10^{-5} \end{bmatrix}$$

$$\nabla_{w2} L = \begin{bmatrix} -0.39312 & 6.5658x10^{-6} & -8.8854x10^{-7} \\ 0.052985 & 8.8493x10^{-7} & 1.1976x10^{-7} \end{bmatrix}$$

4.

```
# Importing necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import tensorflow as tf
```

4. (a)

```
# Answer to the question no 4(a)
# Set up number of training examples
N = 400
N_each = round(N/40)  # For setting the range

# Set up upper and lower bound on input probabilites
Uh, Ul = 20, -1 # Upper and Lower Limit

# Setting up the O-class
O_first = np.asarray([(random.uniform(-Ul, Uh), random.uniform(-
Ul, Uh)) for x in range(N_each) for y in range(N_each)])
O_third = np.asarray([(random.uniform(-Uh, Ul), random.uniform(-
Uh, Ul)) for x in range(N_each) for y in range(N_each)])
O = np.concatenate((O_first, O_third))

# Setting up the X-class
X_fourth = np.asarray([(random.uniform(-Ul, Uh), random.uniform(-
Uh, Ul)) for x in range(N_each) for y in range(N_each)])
X_second = np.asarray([(random.uniform(-Uh, Ul), random.uniform(-
Ul, Uh)) for x in range(N_each) for y in range(N_each)])
X = np.concatenate((X_fourth, X_second))

# Plotting the X-class and O-class
plt.scatter(O[:,0], O[:,1], marker= 'o', c= 'red', edgecolors= 'none', label= 'O-class')
plt.scatter(X[:,0], X[:,1], marker= '+', c= 'blue', label= 'X-class')
plt.legend()
plt.grid(True)
```

4. (b)

```
# Answer to the question no 4(b)

# Setting up training examples
X_train = np.concatenate((X, O))

# Setting up labels
y_x = np.asarray([(1, 0) for x in range(N_each*N_each*2)])
y_o = np.asarray([(0, 1) for x in range(N_each*N_each*2)])
y_train = np.concatenate((y_x, y_o))
```

4.(c)

```
# Answer to the question no 4(c)

# Setting up model as sequential
model = Sequential()

# Setting up the input shape
model.add(tf.keras.Input(shape=(2,)))

# Adding first layer with 8 nodes and activation function as relu
model.add(Dense(8, activation='relu'))

# Adding second layer with activation function as sigmoid
model.add(Dense(2, activation='sigmoid'))

print(model.summary())
```

4. (d)

```
# Answer to the question no 4(d)
# Setting up loss as binary cross entropy, optimizer as adam, and metrics as accuracy
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=['accuracy'],
)

# Training your model
model.fit(X_train, y_train, batch_size= 10, epochs= 200, verbose= 1)
```

4. (e)

```
# Answer to the question no 4(e)
N_test = 300
N_test_each = round(N_test / 4)
# Setting up the O-class
O_first_test = np.asarray([(random.uniform(-Ul, Uh), random.uniform(-
Ul, Uh)) for x in range(N_test_each) for y in range(N_test_each)])
O_third_test = np.asarray([(random.uniform(-Uh, Ul), random.uniform(-
Uh, Ul)) for x in range(N_test_each) for y in range(N_test_each)])
O_test = np.concatenate((O_first_test, O_third_test))

# Setting up the X-class
X_fourth_test = np.asarray([(random.uniform(-Ul, Uh), random.uniform(-
Uh, Ul)) for x in range(N_test_each) for y in range(N_test_each)])
X_second_test = np.asarray([(random.uniform(-Uh, Ul), random.uniform(-
Ul, Uh)) for x in range(N_test_each) for y in range(N_test_each)])
X_class_test = np.concatenate((X_fourth_test, X_second_test))

# Setting up training examples
X_test = np.concatenate((X_class_test, O_test))

# Setting up labels
y_x_test = np.asarray([(1, 0) for x in range(N_test_each*N_test_each*2)])
y_o_test = np.asarray([(0, 1) for x in range(N_test_each*N_test_each*2)])
y_test = np.concatenate((y_x_test, y_o_test))


# Evaluating model
_, score = model.evaluate(X_test, y_test, verbose= 0)

print("Accuracy :", score)
```

4. (f)

Code of the assignment is submitted in another file.

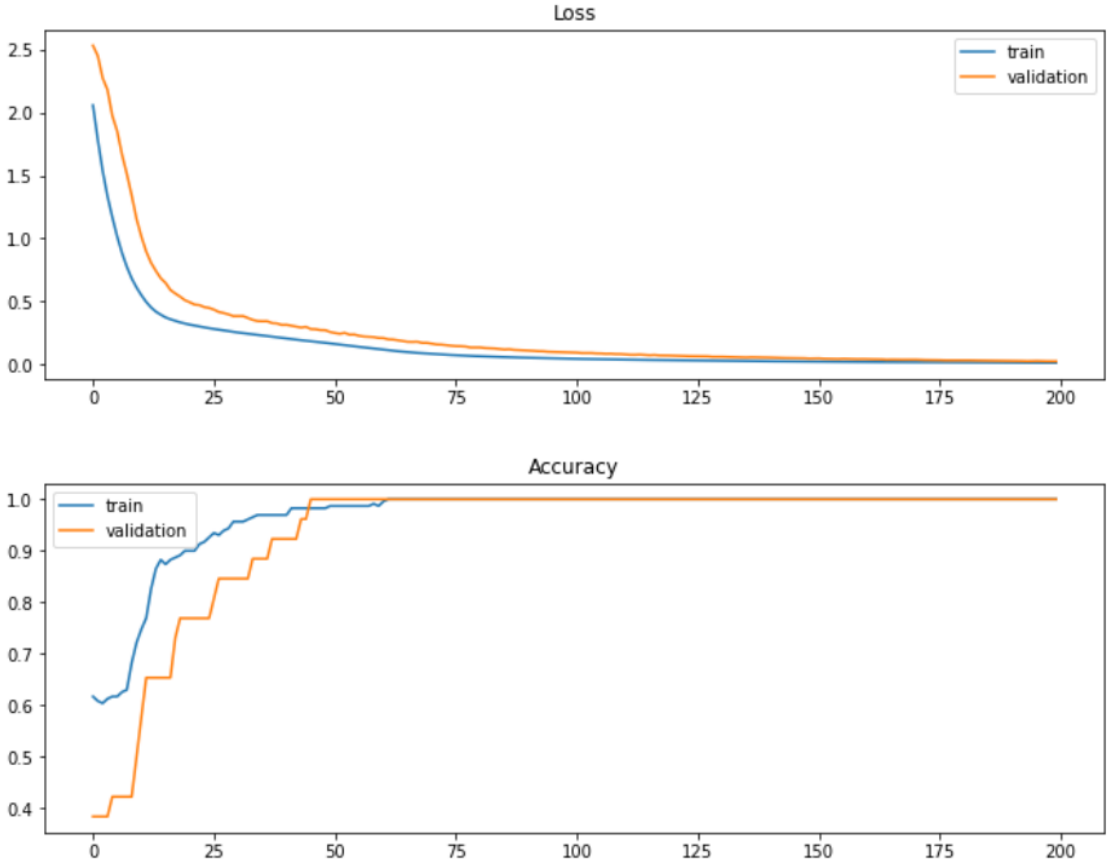
4.(g)

```
# Answer to the question no 4(g)
```

Plotting accuracy and loss for Adam optimizer and BinaryCrossentropy loss

```
# Training your model & keeping history while training
history = model.fit(X_train, y_train, validation_split = 0.1, batch_size= 10, epochs= 200, verbos
e= 1)

figure, axes = plt.subplots(nrows=2, ncols=1)

plt.subplot(211)
plt.title("Loss")
plt.plot(history.history['loss'], label = 'train')
plt.plot(history.history['val_loss'], label = 'validation')
plt.legend()

plt.subplot(212)
plt.title("Accuracy")
plt.plot(history.history['accuracy'], label = 'train')
plt.plot(history.history['val_accuracy'], label = 'validation')
plt.legend()

figure.tight_layout(pad=3.0)
plt.show()
```

If we run the above code with validation split set at 0.1, we could visualize the loss and accuracy of the model with respect to the epoch number and training and validation data. The graphs are as follows:



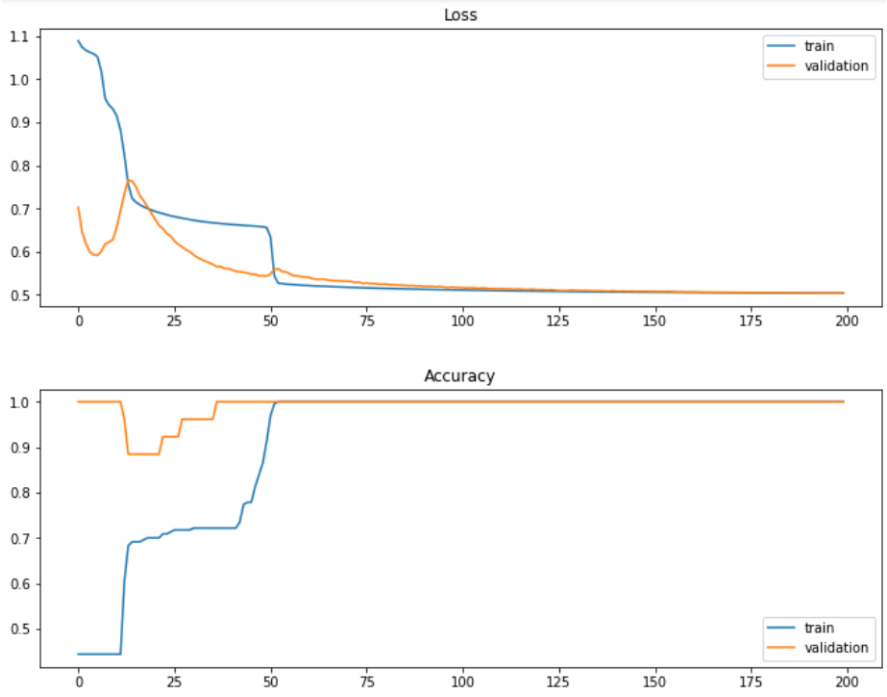
Here, we can see at initial epochs the accuracy for both validation and training set is low. With the increase of epoch, the accuracy increases, and the loss decreases. Maximum accuracy for this dataset is found after about 30 epochs.

Plotting accuracy and loss for Adam optimizer and Hinge loss

Code:

```
model.compile(
    optimizer="adam",
    loss="hinge",
    metrics=['accuracy'],
)
```

Graph:

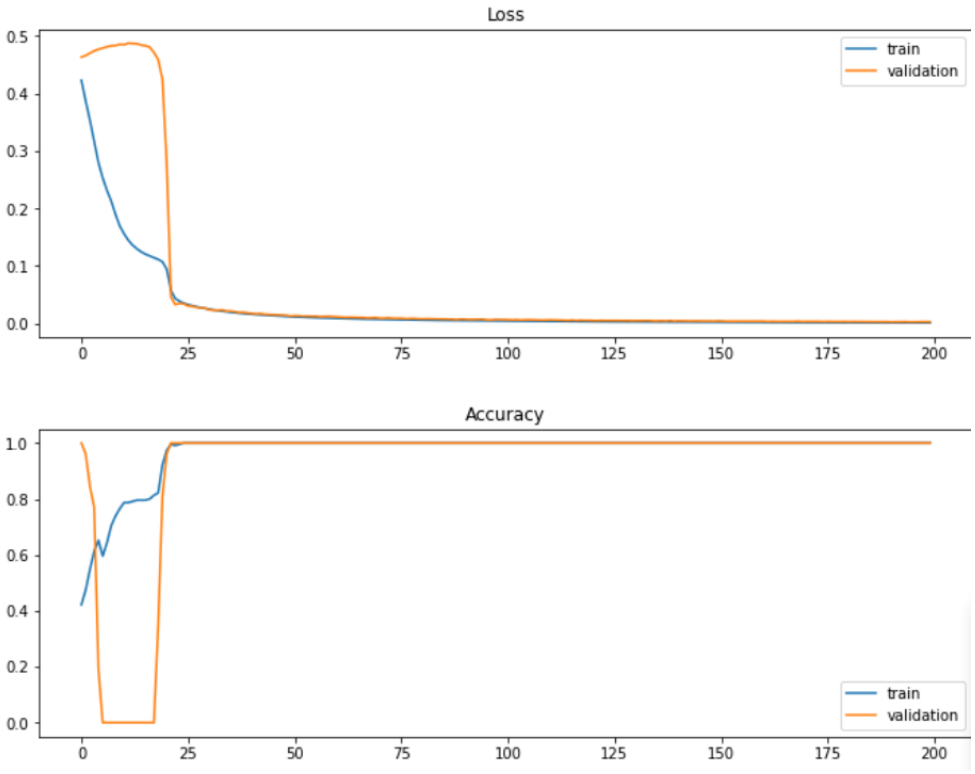


Because of the hinge loss's non-smooth nature, the maximum accuracy is found after more than 50 epochs. With comparison to BinaryCrossentropy loss, the loss for hinge loss saturates near 0.53 where the loss for BinaryCrossentropy saturates around 0.25.

Plotting accuracy and loss for Adam optimizer and MeanSquaredError loss

Code:

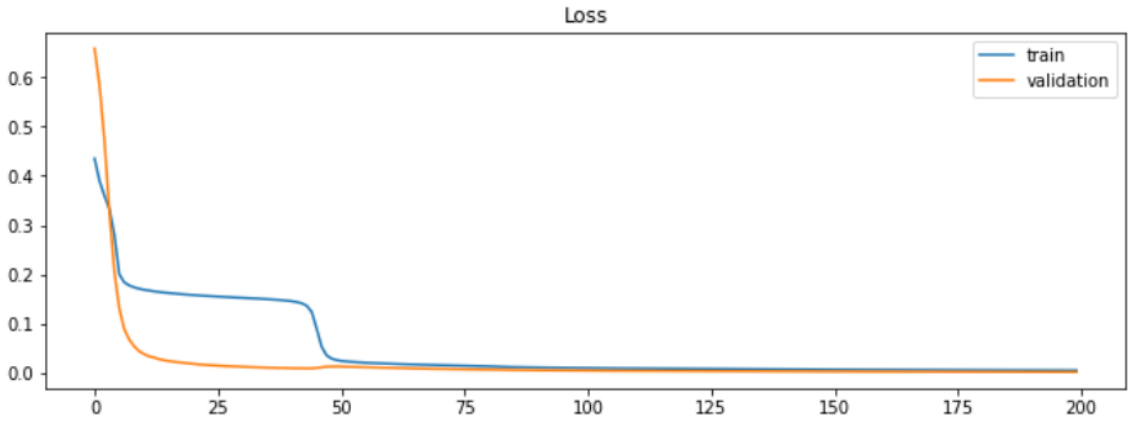
```
model.compile(
    optimizer="adam",
    loss="MeanSquaredError",
    metrics=['accuracy'],
)
```

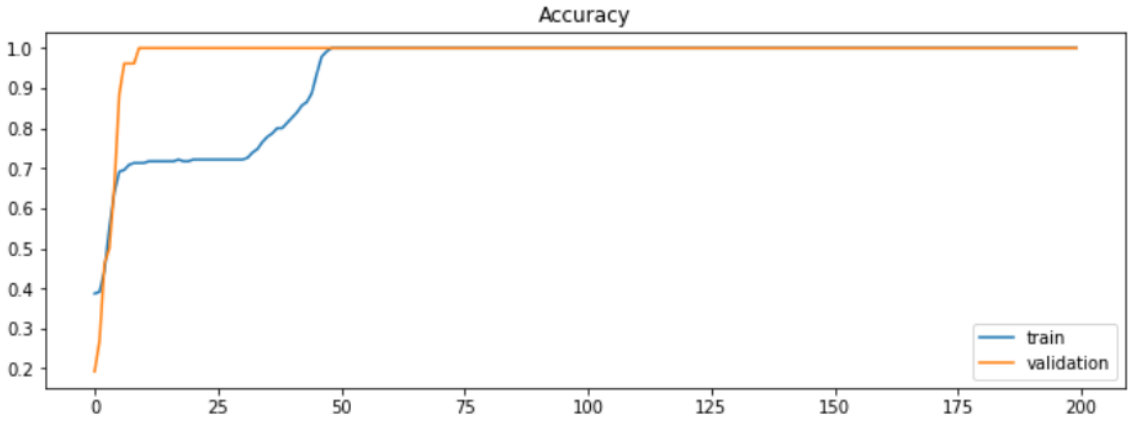


The maximum accuracy for this case is found at epoch 25. With comparison to BinaryCrossentropy and hinge loss, the loss for MeanSquaredError is less than that of other two cases. The accuracy and loss value saturates at lower epochs.

Plotting accuracy and loss for SGD optimizer and MeanSquaredError loss

```
model.compile(
    optimizer="SGD",
    loss="MeanSquaredError",
    metrics=['accuracy'],
)
```



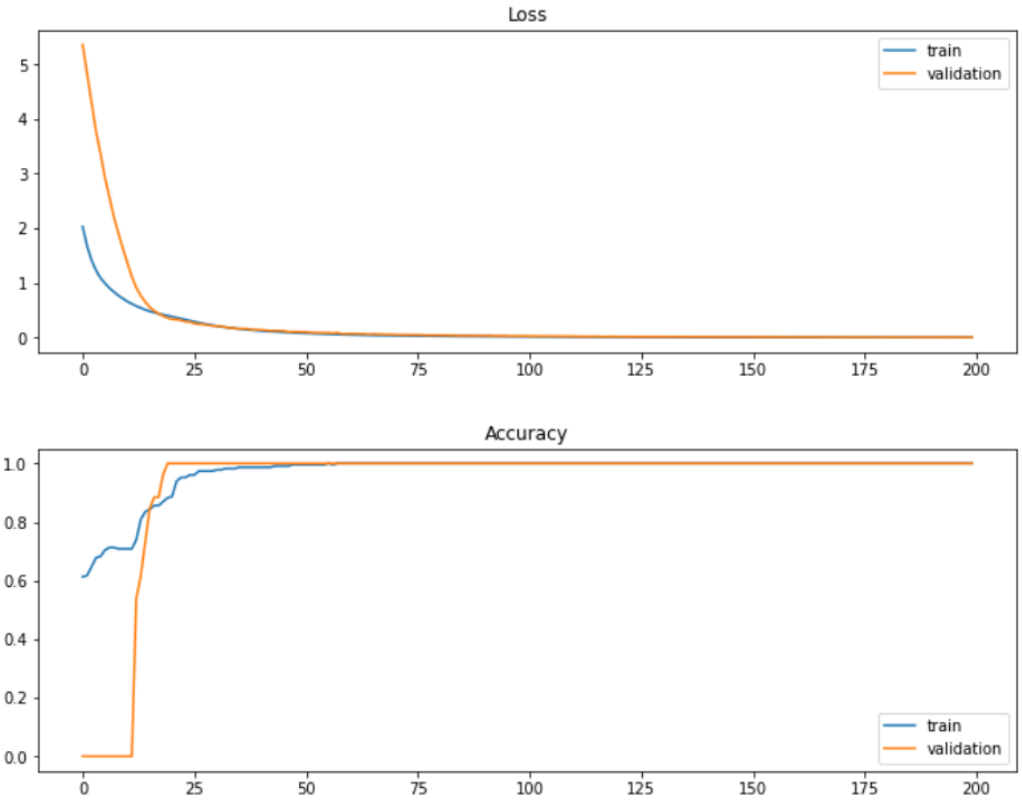


For this case, the minimum saturated loss and maximum accuracy is found after 49 epochs. Performance of Adam optimizer and MeanSquaredError loss is better in this regard.

Adding another dense layer with 4 nodes and relu activation function in between two other layers and using Adam optimizer and BinaryCrossentropy loss

Code:

```
# Setting up model as sequential
model = Sequential()
# Setting up the input shape
model.add(tf.keras.Input(shape=(2,)))
# Adding first layer
model.add(Dense(8, activation='relu'))
# Adding another layer in between
model.add(Dense(4, activation='relu'))
# Adding last layer
model.add(Dense(2, activation='sigmoid'))
```

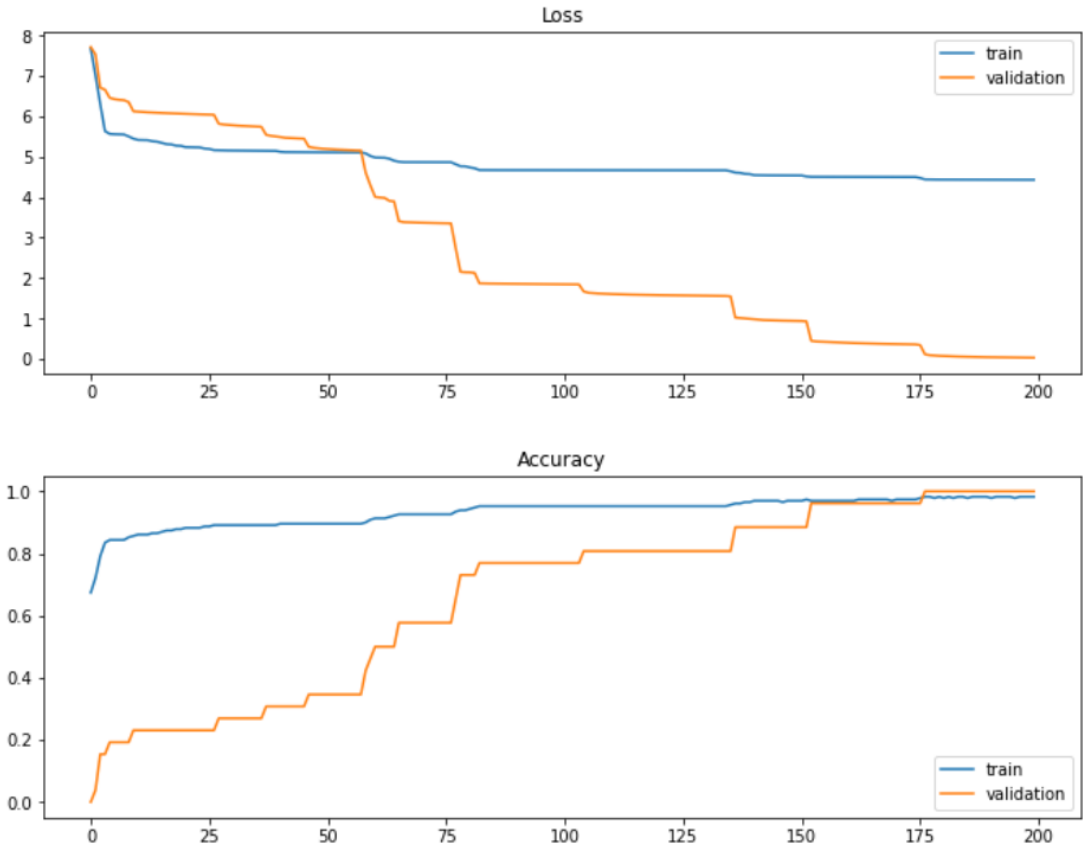


The difference between this case and the first case is the middle layer with 4 nodes and relu activation. The optimizer and loss type are same for both cases. However, we can notice that the accuracy and loss for this case reach in less epochs.

Plotting accuracy and loss for Adam optimizer and BinaryCrossentropy loss with the last layer activation as relu (similar structure with the first case)

Code:

```
# Setting up model as sequential
model = Sequential()
# Setting up the input shape
model.add(tf.keras.Input(shape=(2,)))
# Adding first layer
model.add(Dense(8, activation='relu'))
# Adding last layer
model.add(Dense(2, activation='relu'))
```



Since the last layer activation function is relu, we can see that the accuracy is yet to reach 100% after 200 epochs. As this is a binary classification problem and sigmoid function results in a value between 0 and 1, sigmoid function produces better result in terms of accuracy and loss.

4. (h)

```
# Answer to the question no 4(h)

# Making the labels from two dimension to one dimension
# For (1, 0) = 1 and for (0, 1) = 0
def oneD_from_twoD(y):
  y_oned = []
  for i in range(len(y)):
    if y[i, 0] == 1:
      y_oned.append(1)
    else: y_oned.append(0)
  return(np.asarray(y_oned))

y_test_oned = oneD_from_twoD(y_test)
y_train_oned = oneD_from_twoD(y_train)

# Setting up he model for one dimensional output
model_2 = Sequential()
model_2.add(tf.keras.Input(shape=(2,)))
model_2.add(Dense(8, activation='relu'))
model_2.add(Dense(1, activation='sigmoid'))

print(model_2.summary())

# Setting up loss as binary cross entropy, optimizer as adam, and metrics as accuracy
model_2.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=['accuracy'],
)

# Training your model
model_2.fit(X_train, y_train_oned, batch_size= 10, epochs= 200, verbose= 1)

# Evaluating model
_, score = model_2.evaluate(X_test, y_test_oned, verbose= 0)

print("Accuracy :", score)

# Plotting decision boundary
def plot_decision_boundary(X, y, model, steps=1000, cmap='Paired'):
    """
    Function to plot the decision boundary and data points of a model.
    Data points are colored based on their actual label.
    """
    cmap = get_cmap(cmap)

    # Define region of interest by data limits
    xmin, xmax = X[:,0].min() - 1, X[:,0].max() + 1
    ymin, ymax = X[:,1].min() - 1, X[:,1].max() + 1
    x_span = np.linspace(xmin, xmax, steps)
    y_span = np.linspace(ymin, ymax, steps)
    xx, yy = np.meshgrid(x_span, y_span)

    # Make predictions across region of interest
    labels = model.predict(np.c_[xx.ravel(), yy.ravel()])

    # Plot decision boundary in region of interest
    z = labels.reshape(xx.shape)

    fig, ax = plt.subplots()
    ax.contourf(xx, yy, z, cmap=cmap, alpha=0.5)

    # Get predicted labels on training data and plot
    train_labels = model.predict(X)
    ax.scatter(X[:,0], X[:,1], c=y.ravel(), cmap=cmap, lw=0)

    return fig, ax
plot_decision_boundary(X_test, y_test_oned, model_2, cmap = 'RdBu')
```

Decision boundary plot: