Name: Prithul Sarker

Course: CS 687 Fundamentals of Deep Learning

Homework 2

1.

f(x) =

Jacobian of f(x) =

=

2. (a)

Z1 = W1\* x + b1

Z = W2\* Z1 + b2

2. (b)

Z1 = [ -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 = first class| X = xi) = = 0.000911

Softmax loss = - ln (0.000911) = 7

3. (a)

Final Output

W2

b2

b1

W1

x

3. (b)

W1

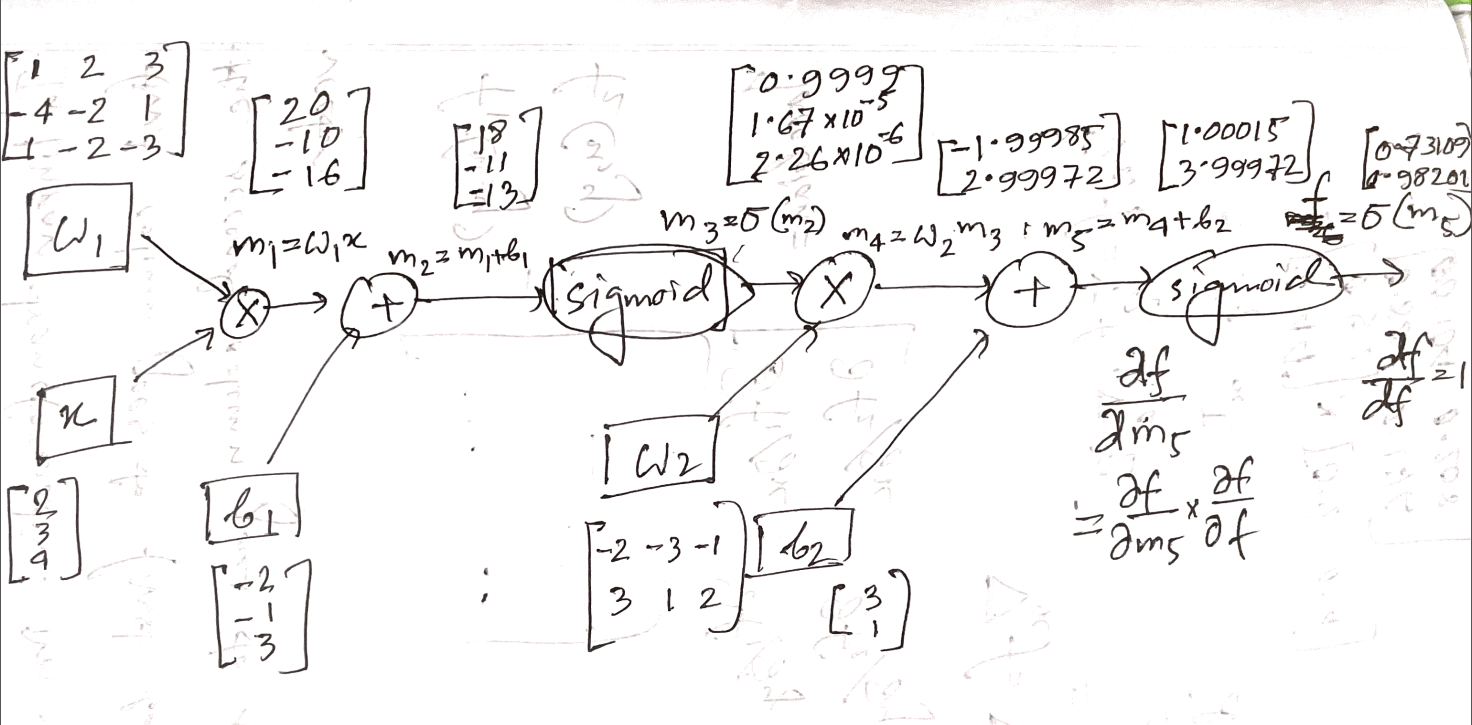
x

Final Output

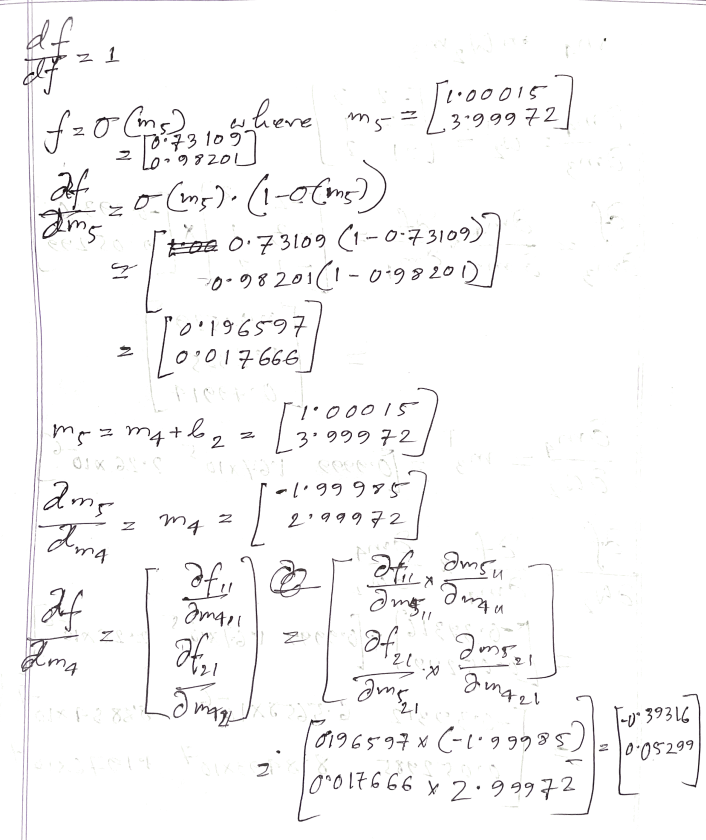
b1

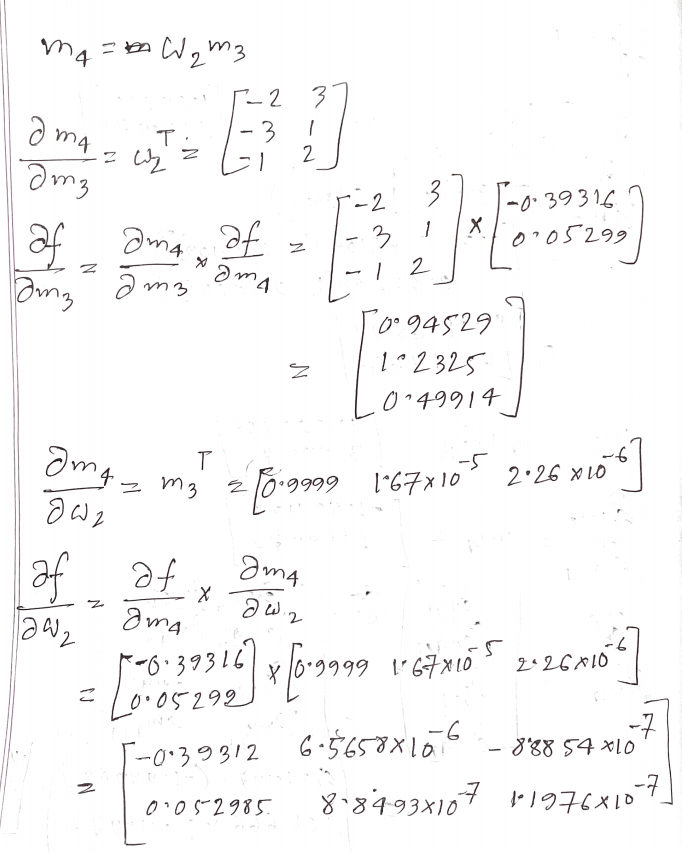
W2

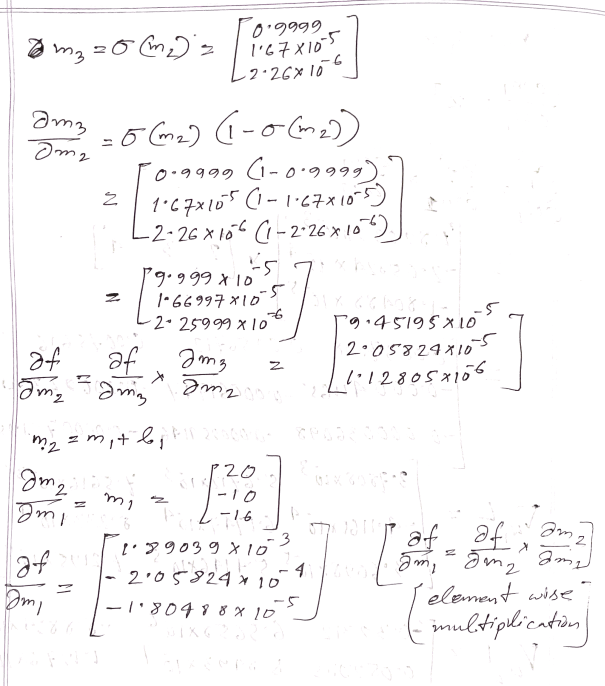
b2

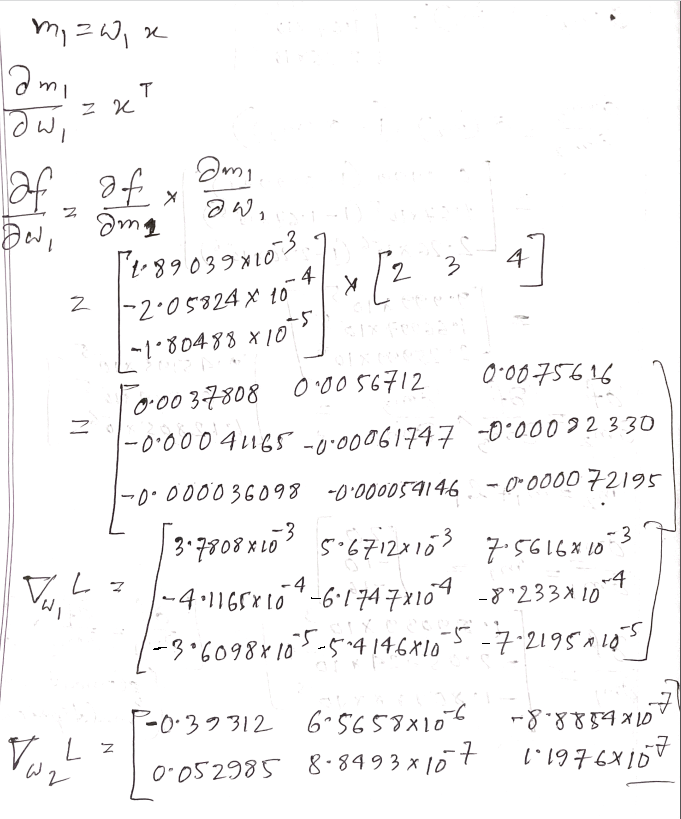


3. (c)









w1L =

w2L =

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, verbose= 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')

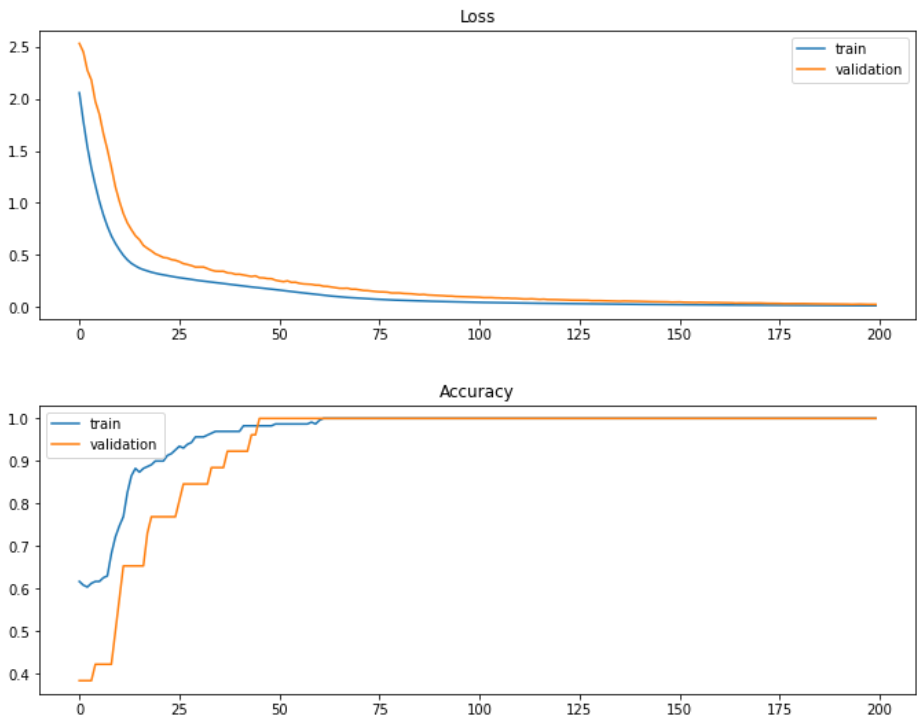
plt.plot(history.history['val\_accuracy'], label = 'validation')

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(

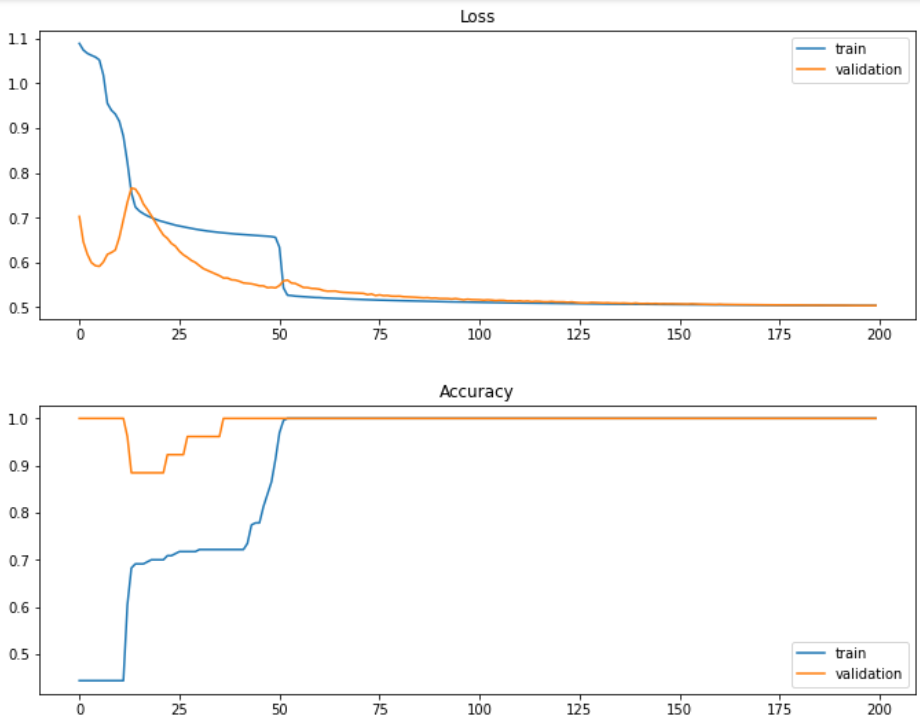
optimizer="adam",

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:

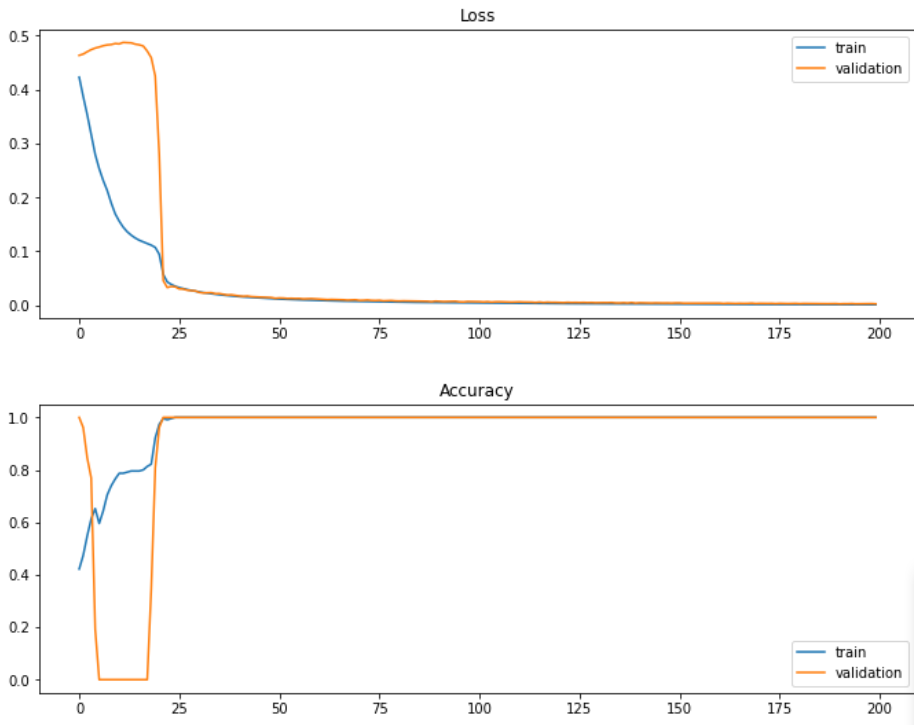
model.compile(

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

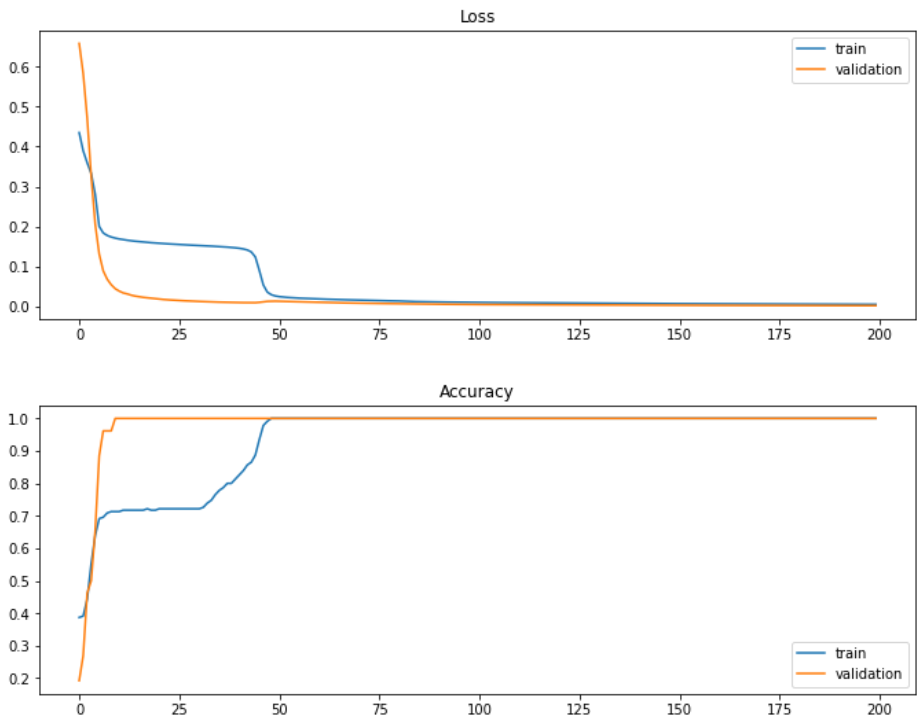
model.compile(

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

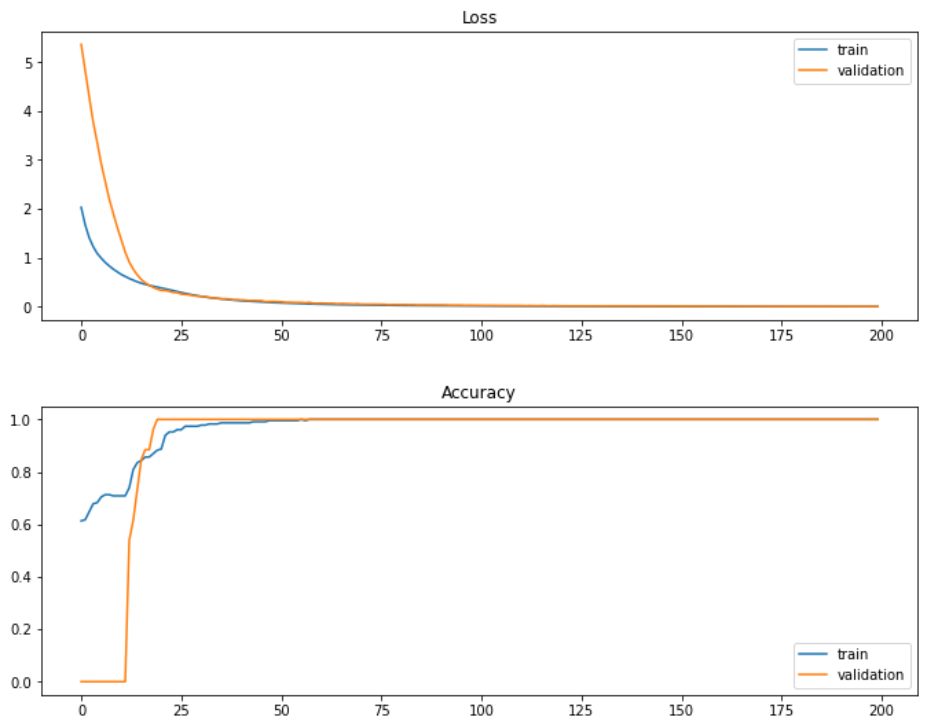
model.add(Dense(8, activation='relu'))

# 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

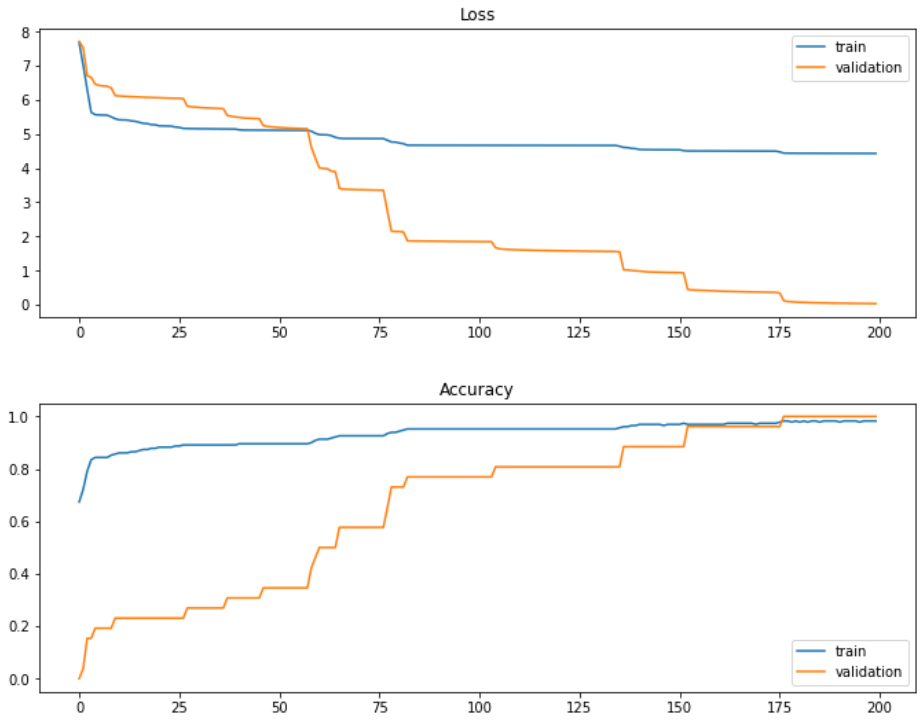
model.add(tf.keras.Input(shape=(2,)))

# 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:

