“Neural Networks”

Shervin Halat

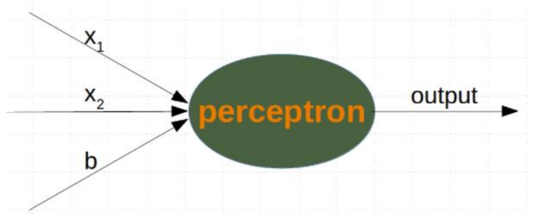
98131018

Homework 1

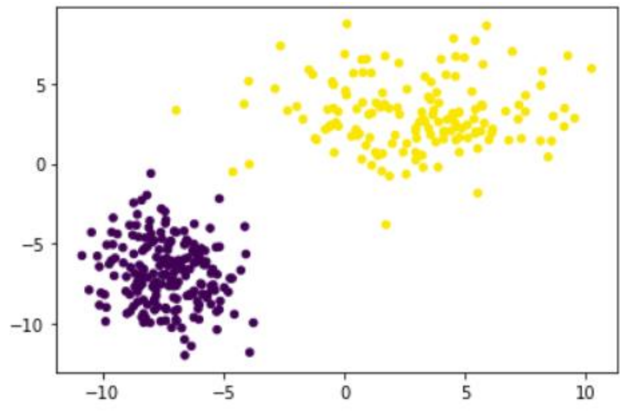
“First Part”

1.

As it can be figured out from the figure below, two classes seem to be linearly separable. Generally, there is a method to investigate linear separability of a binary data by a single perceptron called single-layer perceptron method. This method guarantees that the perceptron will converge if the data is linearly separable; In fact, a single-layer perceptron along with sigmoid activation function plus binary cross-entropy loss function, all together, construct a Logistic Regression model which is a linear model for a binary classification; Therefore, in order to investigate linear separability of our data, this model (which is shown below) will be employed:



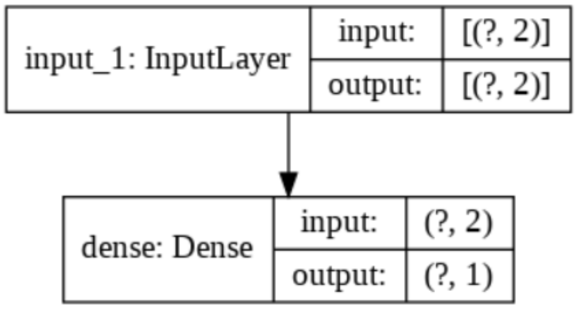
Distribution of our data is shown below:



Defined model configuration is as follows:

```python
model = keras.models.Sequential([
                                keras.layers.Input(2),
                                keras.layers.Dense(units = 1 , activation = 'sigmoid')
])
model.compile(optimizer = 'Adam' , loss = 'binary_crossentropy' , metrics = ['accuracy'])
model.fit(x, y, epochs=700)
```

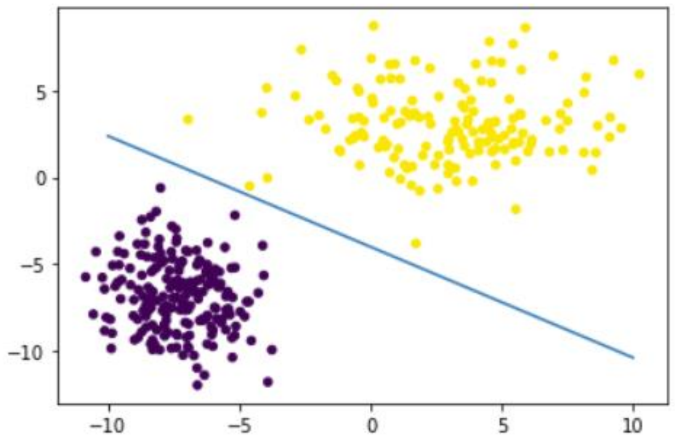
```python
keras.utils.plot_model(model,show_shapes = True,show_layer_names= True)
```



As it is shown below, the model has reached to 100% accuracy on the entire dataset after training on them. This proves that the dataset is linearly separable.

```
Epoch 800/800
11/11 [==============================] - 0s 2ms/step - loss: 0.0029 - accuracy: 1.0000
<tensorflow.python.keras.callbacks.History at 0x7fe4d12f3518>
```

The obtained separating line by the model weights after training is shown below:



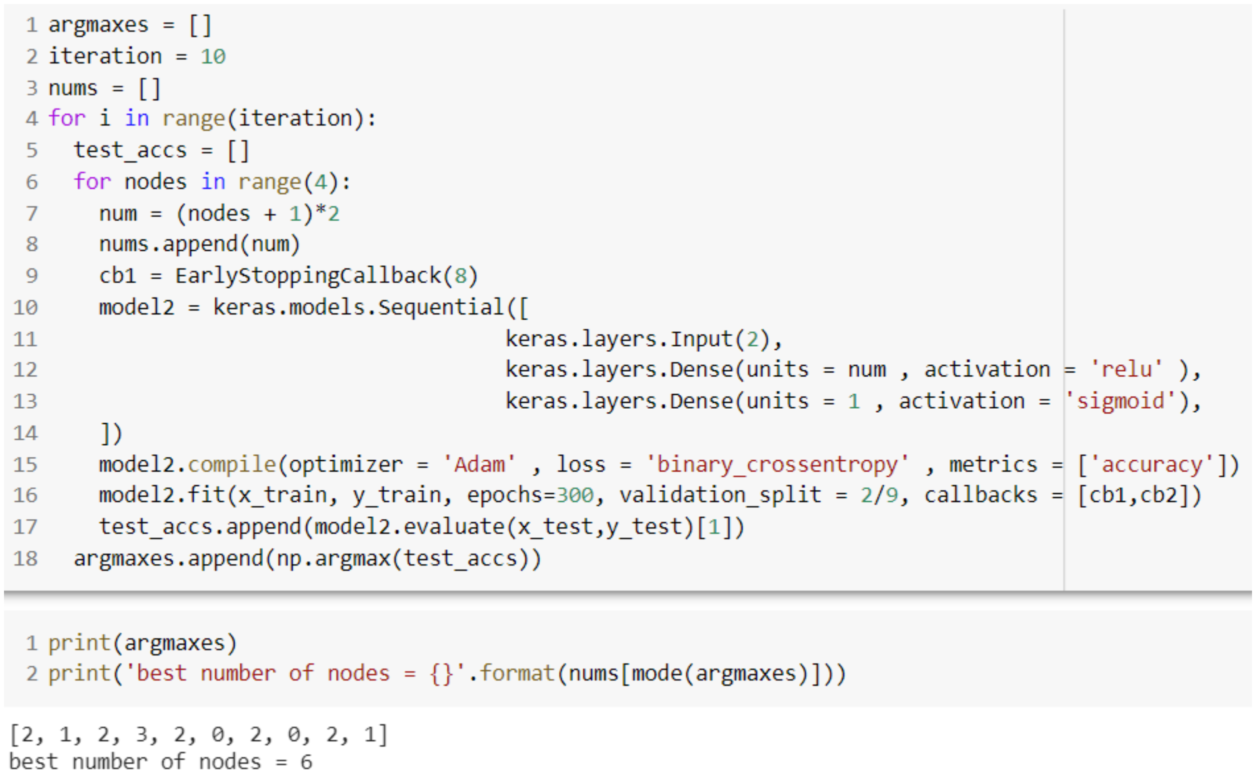
2.

The input layer of the defined model has two nodes since the input layer and data points should be of the same dimension.

The output layer of the defined model has one node since the number of output layer nodes determine number of classes of the dataset. In this case, as the dataset is a binary class, one node is satisfactory, so that, output ‘1’ determines one class and output ‘0’ determines another class.

The ‘sigmoid’ activation function was selected for this problem. The main purpose of this selection is mainly because of the fact that the sigmoid function maps any real-valued number to a probability between 0 and 1 which determines the probability of belonging to one of the classes. Therefore, it is mainly used for binary classification problems.

To obtain the best number of nodes for the hidden layer, following script was implemented. That is, with ten iterations, each time considering the hidden layer with 2, 4, 6, and 8 nodes, “the number of nodes” with best final test accuracy among obtained test accuracies of these 4 models was saved. After all, the most repeated “number of nodes” was considered as the best number of nodes. **In our case, the hidden layer with six number of nodes obtained the most votes.**



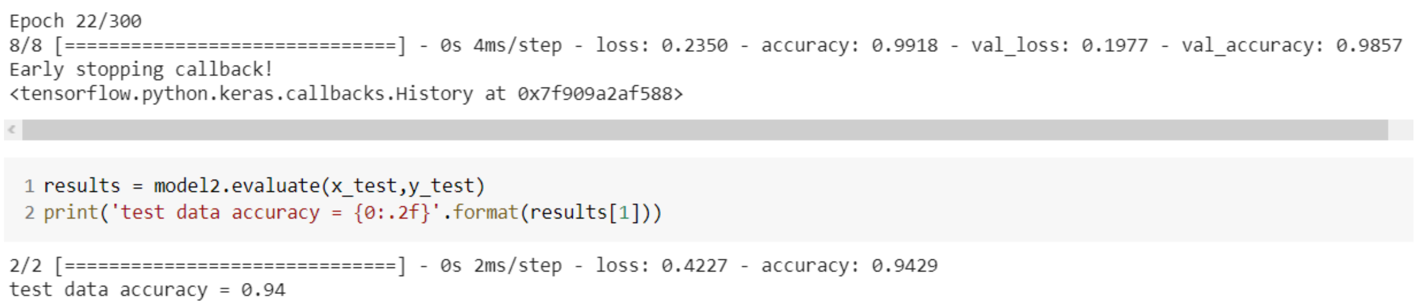
```python
argmaxes = []
iteration = 10
nums = []
for i in range(iteration):
  test_accs = []
  for nodes in range(4):
    num = (nodes + 1)*2
    nums.append(num)
    cb1 = EarlyStoppingCallback(8)
    model2 = keras.models.Sequential([
                                keras.layers.Input(2),
                                keras.layers.Dense(units = num , activation = 'relu' ),
                                keras.layers.Dense(units = 1 , activation = 'sigmoid'),
    ])
    model2.compile(optimizer = 'Adam' , loss = 'binary_crossentropy' , metrics = ['accuracy'])
    model2.fit(x_train, y_train, epochs=300, validation_split = 2/9, callbacks = [cb1,cb2])
    test_accs.append(model2.evaluate(x_test,y_test)[1])
  argmaxes.append(np.argmax(test_accs))
```

```python
print(argmaxes)
print('best number of nodes = {}'.format(nums[mode(argmaxes)]))
```

```
[2, 1, 2, 3, 2, 0, 2, 0, 2, 1]
best number of nodes = 6
```

Final best model evaluation:

Obtained best model reached to 94% accuracy on test data after 22 epochs with early stopping callback as shown below:

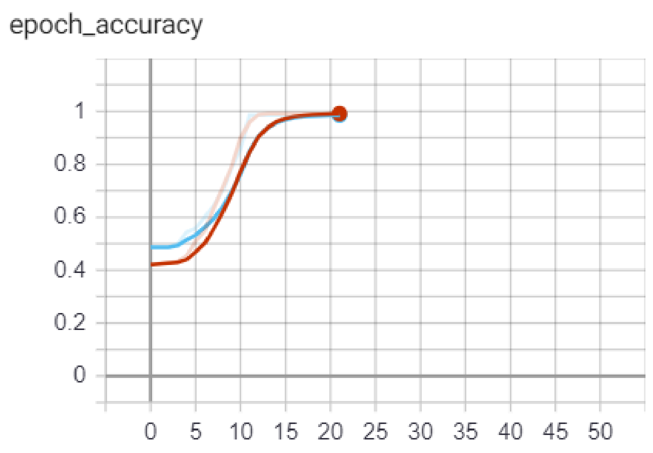


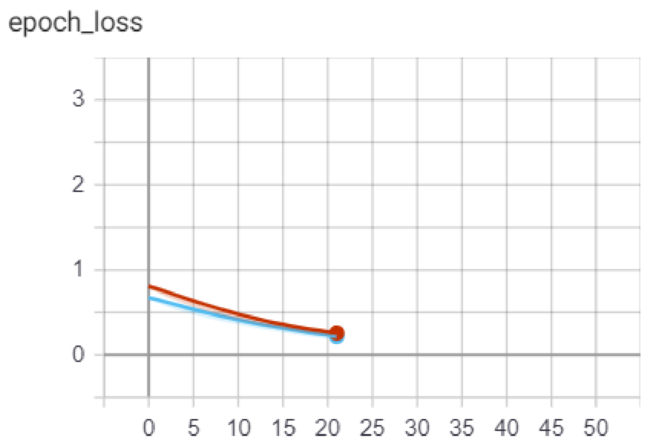
```
Epoch 22/300
8/8 [==============================] - 0s 4ms/step - loss: 0.2350 - accuracy: 0.9918 - val_loss: 0.1977 - val_accuracy: 0.9857
Early stopping callback!
<tensorflow.python.keras.callbacks.History at 0x7f909a2af588>
```

```python
results = model2.evaluate(x_test,y_test)
print('test data accuracy = {0:.2f}'.format(results[1]))
```

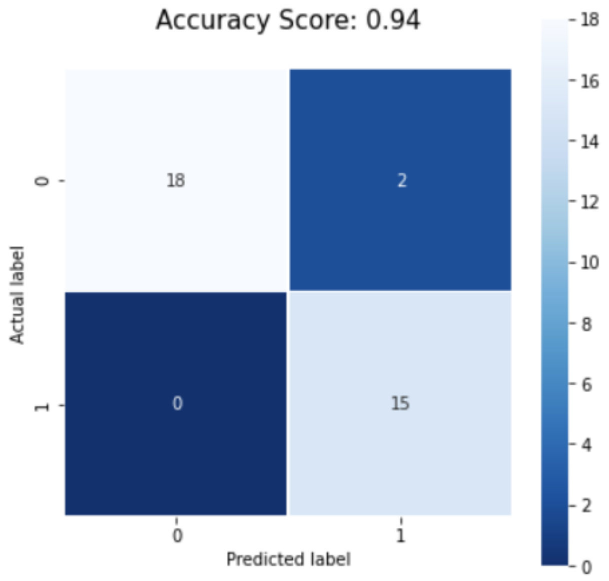
```
2/2 [==============================] - 0s 2ms/step - loss: 0.4227 - accuracy: 0.9429
test data accuracy = 0.94
```

The logs of the best model are as follows:

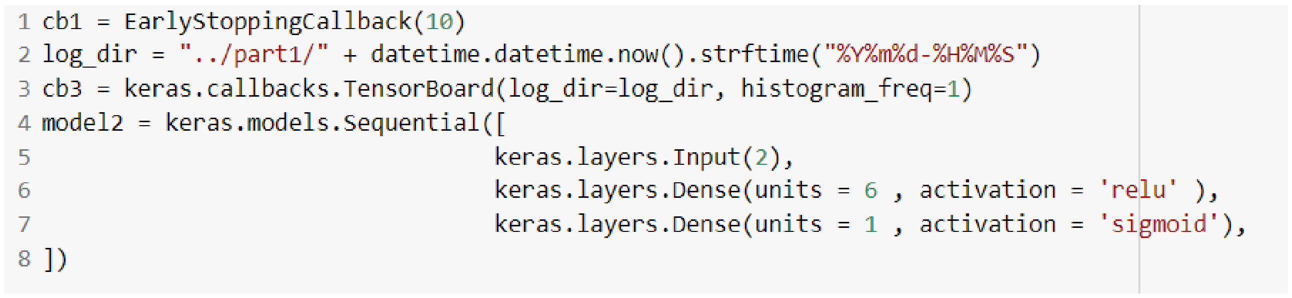




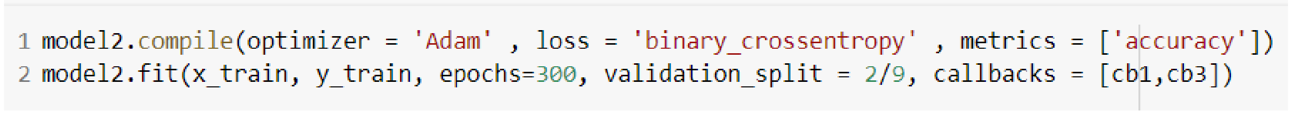
The confusion matrix of the best model is as follows:



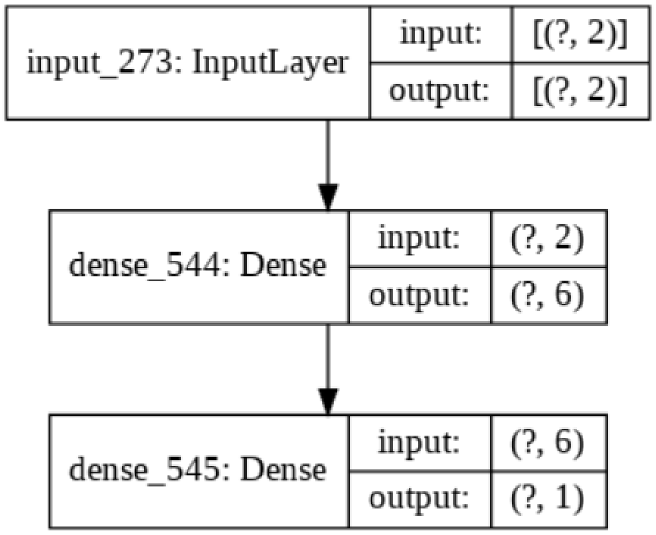
The obtained best model configuration is as follows:



```
cb1 = EarlyStoppingCallback(10)
log_dir = "../part1/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
cb3 = keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)
model2 = keras.models.Sequential([
                                  keras.layers.Input(2),
                                  keras.layers.Dense(units = 6 , activation = 'relu' ),
                                  keras.layers.Dense(units = 1 , activation = 'sigmoid'),
])
```



```
model2.compile(optimizer = 'Adam' , loss = 'binary_crossentropy' , metrics = ['accuracy'])
model2.fit(x_train, y_train, epochs=300, validation_split = 2/9, callbacks = [cb1,cb3])
```



3.

Binary Cross-entropy (BCE) loss function:

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

Binary Cross-Entropy / Log Loss

Mean Squared Error:

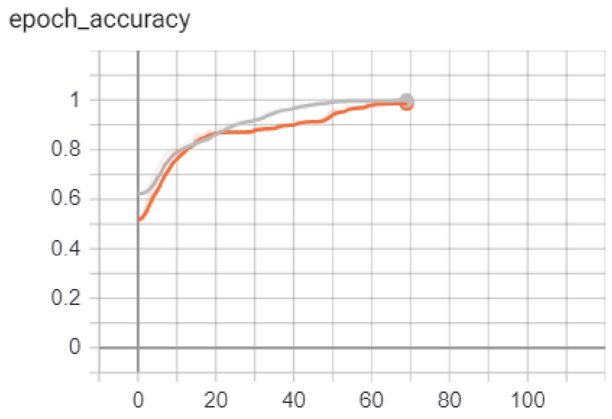
$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

MSE & BCE comparison:

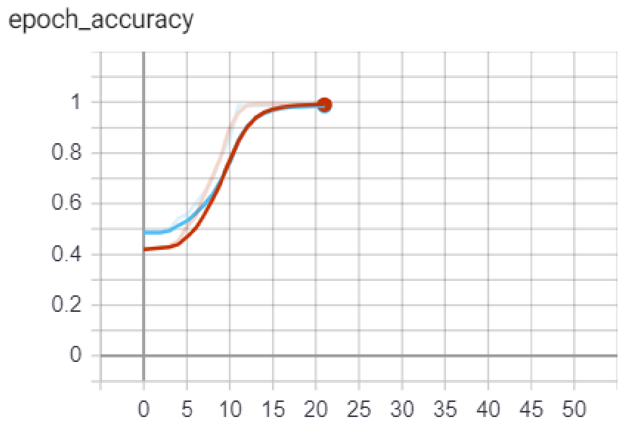
Basically, the main difference between loss functions is rooted in the way the model weights are trained. In the case of comparing MSE with BCE, the main difference in their performance is based on how each of these losses penalize the misclassification. To give an insight, by considering the two equations above, it can be figured out that the BCE loss function increases exponentially as the predicted value get closer to the wrong output as against MSE loss function, whereby, BCE loss function tends to learn the model much faster than the MSE loss function. This conclusion is in agreement with the result obtained for this problem. As it is shown in the plots below, the model with MSE (left) has converged to its peak after 70 epochs while the model with

BCE (right) has converged after about 20 epochs. Also, the BCE has much more gradient than MSE.

MSE

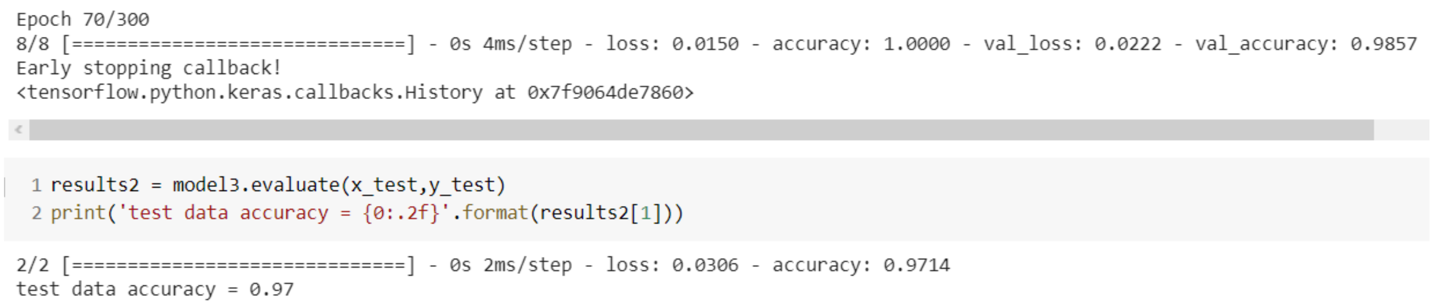


BCE



Final best model evaluation:

Obtained best model reached to 97% accuracy on test data after 70 epochs with early stopping callback as shown below:

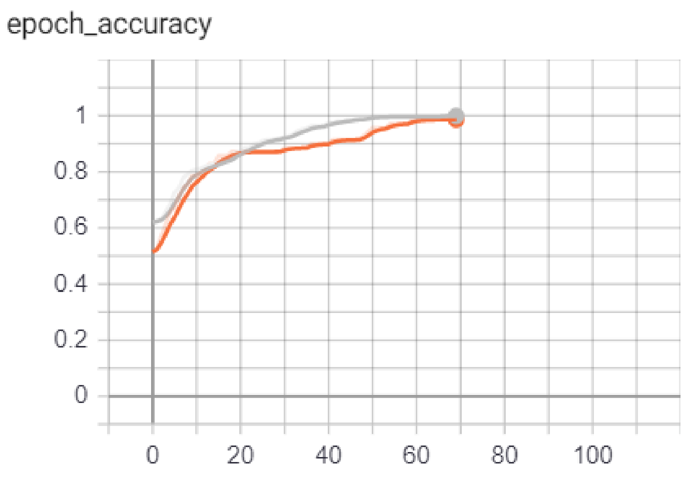


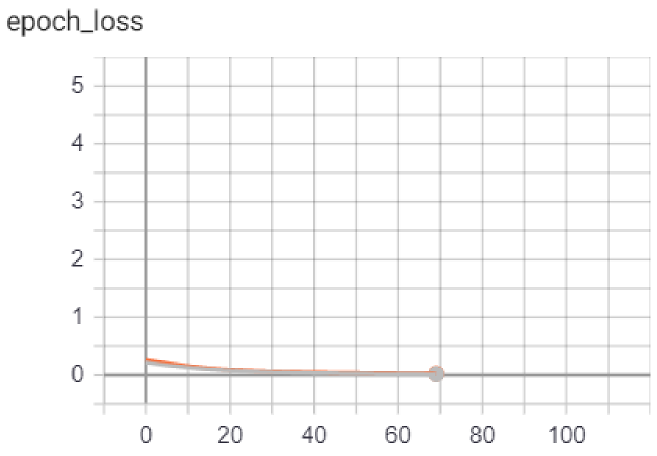
```
Epoch 70/300
8/8 [==============================] - 0s 4ms/step - loss: 0.0150 - accuracy: 1.0000 - val_loss: 0.0222 - val_accuracy: 0.9857
Early stopping callback!
<tensorflow.python.keras.callbacks.History at 0x7f9064de7860>
```

```
results2 = model3.evaluate(x_test,y_test)
print('test data accuracy = {0:.2f}'.format(results2[1]))
```

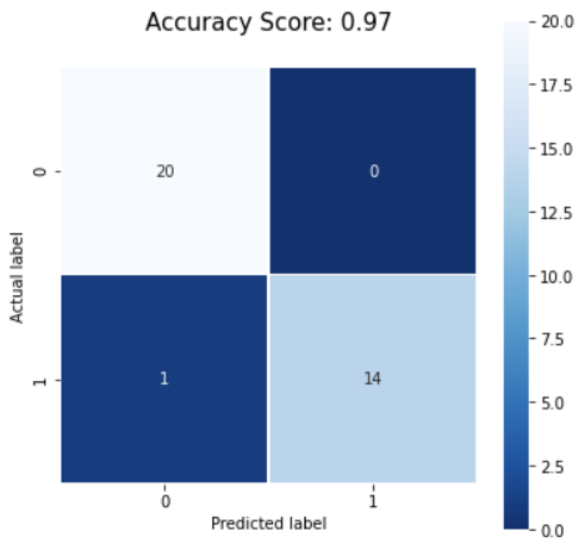
```
2/2 [==============================] - 0s 2ms/step - loss: 0.0306 - accuracy: 0.9714
test data accuracy = 0.97
```

The logs of the best model are as follows:





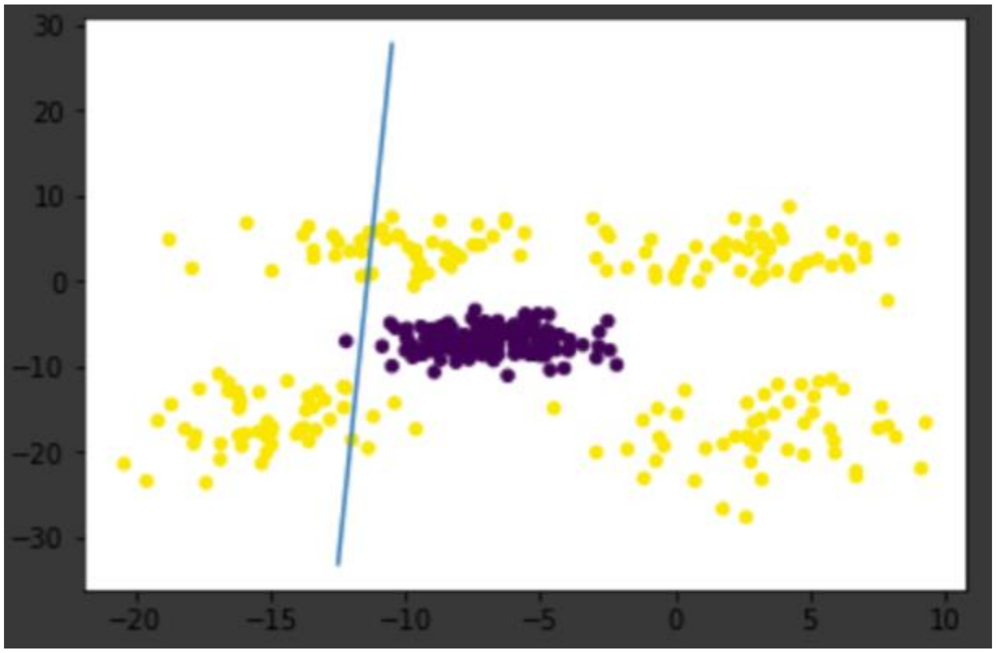
The confusion matrix of the best model is as follows:



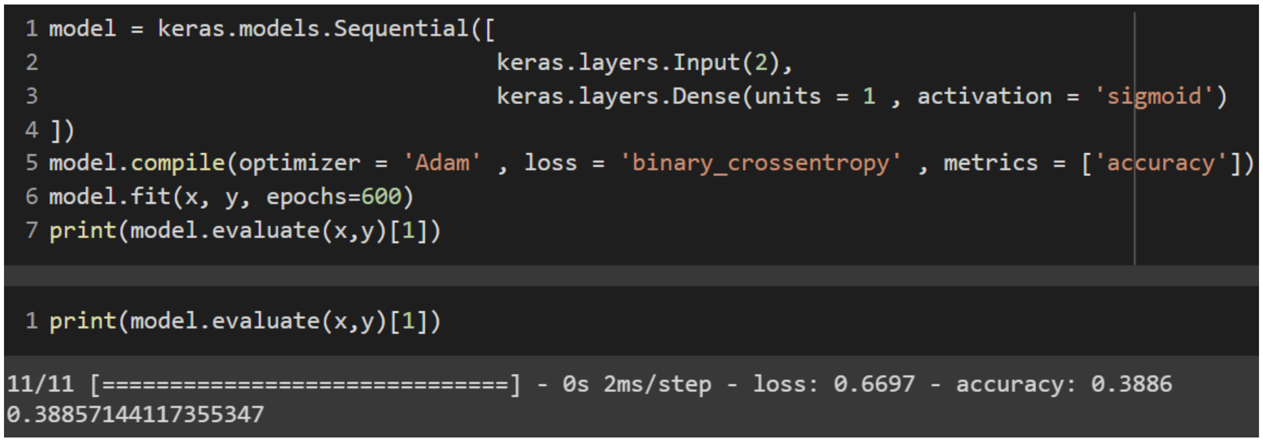
“Second Part”

1.

Again, just like the answer of question 1 from first part, converged single-layer perceptron model after 600 epochs (with accuracy of 38%) has resulted in the following separating line; therefore, the data set is not linearly separable as it is illustrated below:



Model configuration is as follows:



```
model = keras.models.Sequential([
                                keras.layers.Input(2),
                                keras.layers.Dense(units = 1 , activation = 'sigmoid')
])
model.compile(optimizer = 'Adam' , loss = 'binary_crossentropy' , metrics = ['accuracy'])
model.fit(x, y, epochs=600)
print(model.evaluate(x,y)[1])
```

```
print(model.evaluate(x,y)[1])
```

```
11/11 [==============================] - 0s 2ms/step - loss: 0.6697 - accuracy: 0.3886
0.38857144117355347
```

2.

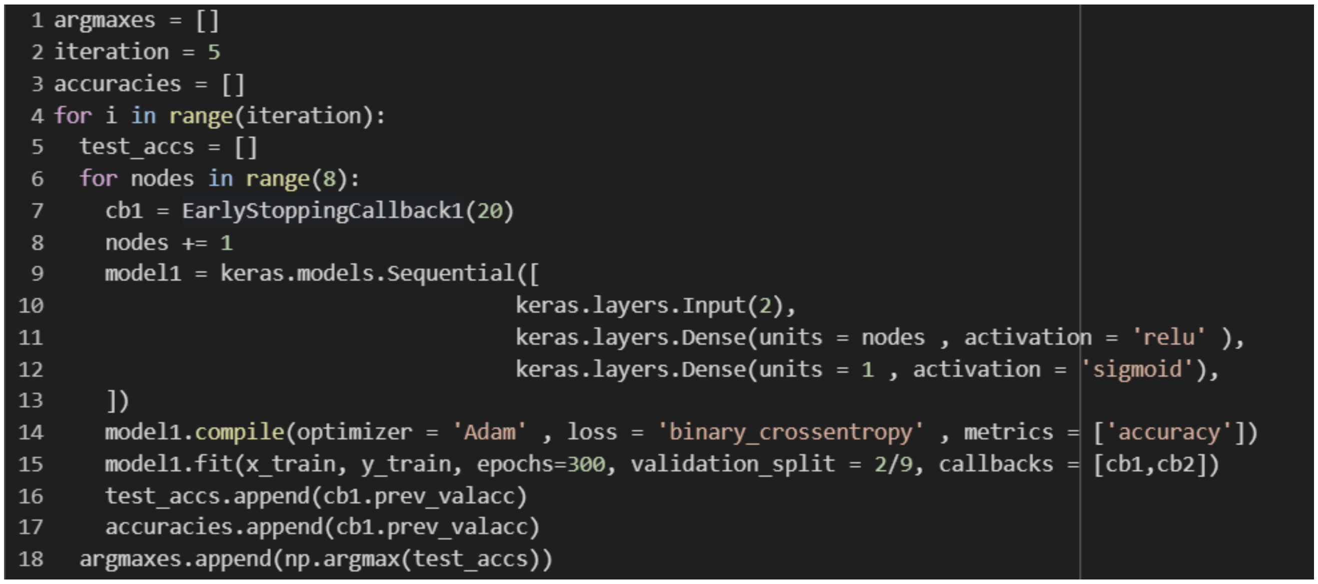
Firstly, there is no need to change the structure of the defined model in the first part for this problem for multiple reasons. First, the data of this part is a binary class data and the task is classification as well; in addition, as it can be figured out from the plot above, two classes may be separated by a single convex region which can be obtained by a network with one hidden layer and three overall layers.

Secondly, for the output layer one node with sigmoid function was selected as the activation function since the task is binary classification; hence, binary_crossentropy loss function was selected for the mutual reason discussed in previous part.

Thirdly, number of nodes in the hidden layer and the optimizer parameter were obtained by trial and error. Conducted experiments are as follows:

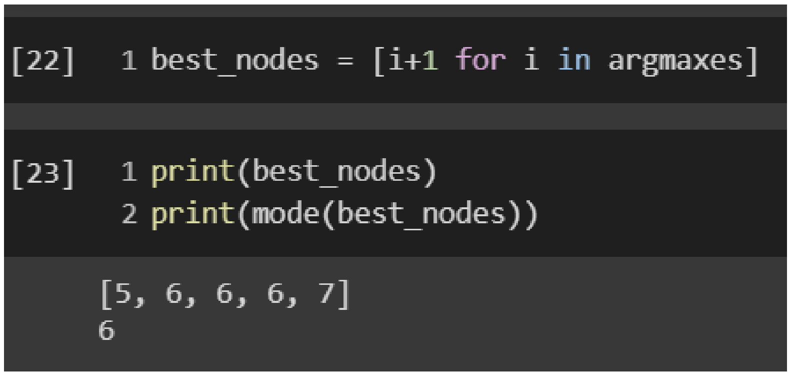
First Experiment (number of hidden layer nodes):

To obtain best number of nodes for the hidden layer, following script was implemented. That is, with five iterations, each time considering the hidden layer with 1 to 8 nodes, “the number of nodes” with best final validation accuracy among obtained validation accuracies of these 8 models was saved. After all, the most repeated “number of nodes” was considered as the best number of nodes.



```
argmaxes = []
iteration = 5
accuracies = []
for i in range(iteration):
  test_accs = []
  for nodes in range(8):
    cb1 = EarlyStoppingCallback1(20)
    nodes += 1
    model1 = keras.models.Sequential([
                                keras.layers.Input(2),
                                keras.layers.Dense(units = nodes , activation = 'relu' ),
                                keras.layers.Dense(units = 1 , activation = 'sigmoid'),
    ])
    model1.compile(optimizer = 'Adam' , loss = 'binary_crossentropy' , metrics = ['accuracy'])
    model1.fit(x_train, y_train, epochs=300, validation_split = 2/9, callbacks = [cb1,cb2])
    test_accs.append(cb1.prev_valacc)
    accuracies.append(cb1.prev_valacc)
  argmaxes.append(np.argmax(test_accs))
```

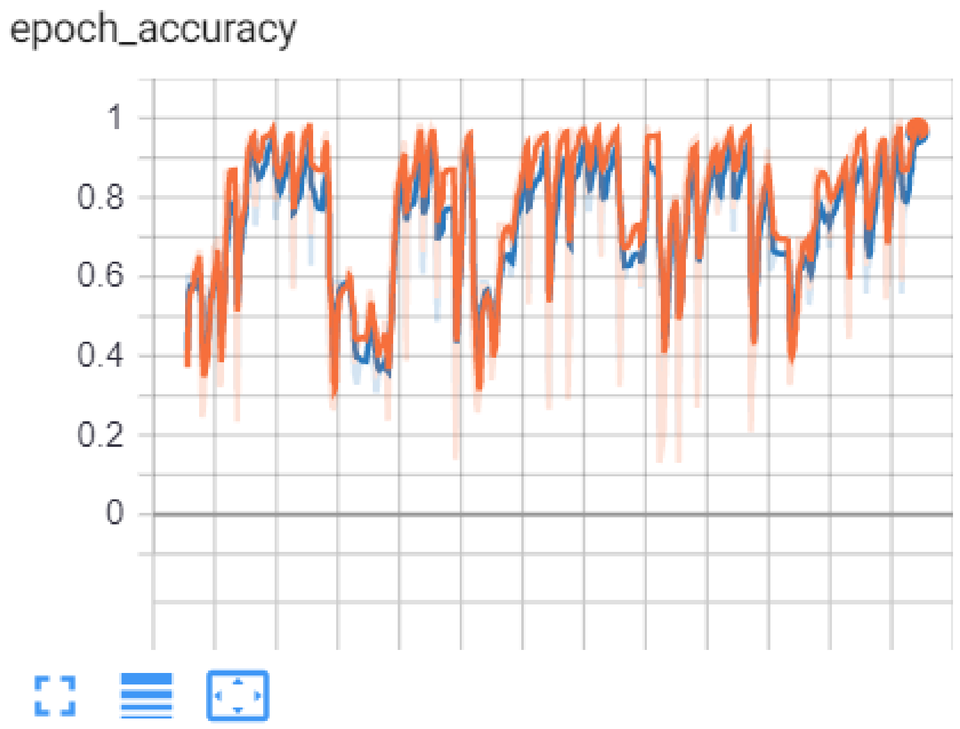
The obtained best "number of nodes" was 6 as it was repeated three times in five iterations as shown below:



```
[22]  best_nodes = [i+1 for i in argmaxes]
```

```
[23]  print(best_nodes)
      print(mode(best_nodes))

      [5, 6, 6, 6, 7]
      6
```

The related log of this experiment (accuracies of all models in order) with 40 generated models (8 models for each iteration) are as follows:



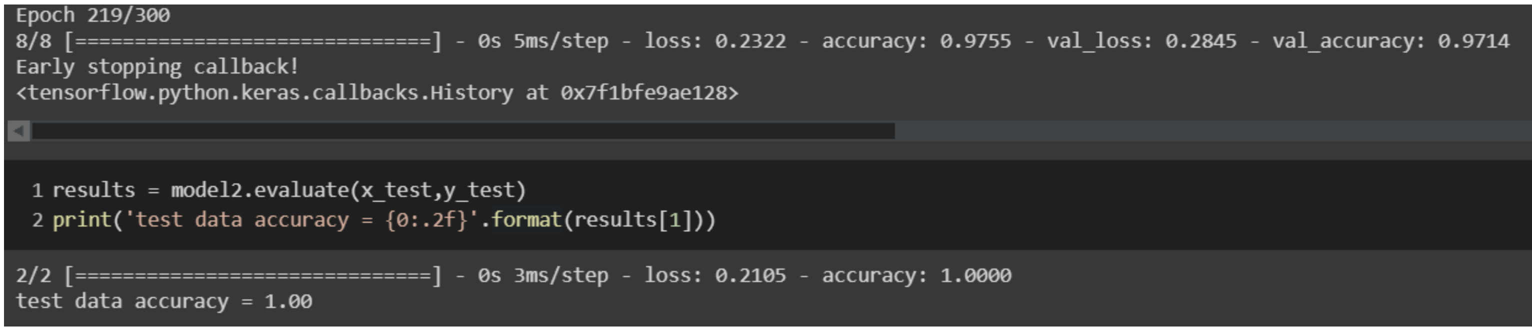
As it is illustrated in the plot above, accuracies reach a peak as the number of nodes increases in each iteration.

Second Experiment (number of hidden layer nodes):

Another experiment was conducted in order to select the best optimizer. After evaluating all other available keras optimizers, it seemed that the model either stuck in local minimums or the learning process becomes slower with other optimizers compared with 'Adam' optimizer. Hence, 'Adam' optimizer was selected as the best choice.

Final best model evaluation:

Obtained best model reached to 100% accuracy on test data after 219 epochs with early stop call back as shown below:



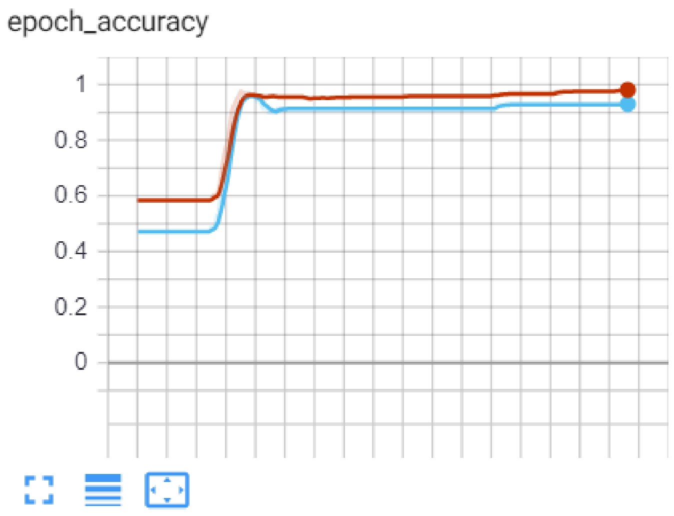
```
Epoch 219/300
8/8 [==============================] - 0s 5ms/step - loss: 0.2322 - accuracy: 0.9755 - val_loss: 0.2845 - val_accuracy: 0.9714
Early stopping callback!
<tensorflow.python.keras.callbacks.History at 0x7f1bfe9ae128>
```

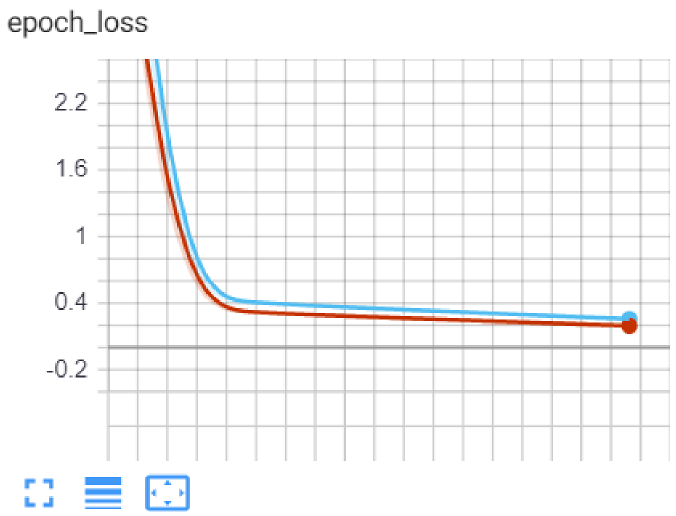
```
results = model2.evaluate(x_test,y_test)
print('test data accuracy = {0:.2f}'.format(results[1]))
```

```
2/2 [==============================] - 0s 3ms/step - loss: 0.2105 - accuracy: 1.0000
test data accuracy = 1.00
```

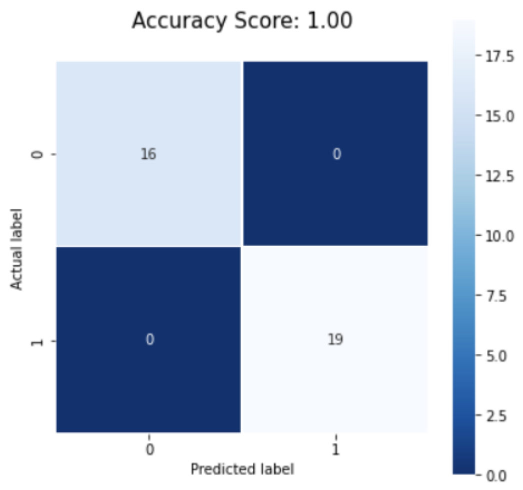
The logs of the best model are as follows:

As we can see in the plots below, the model has been learnt in few epochs fast and has reached to 100% accuracy.

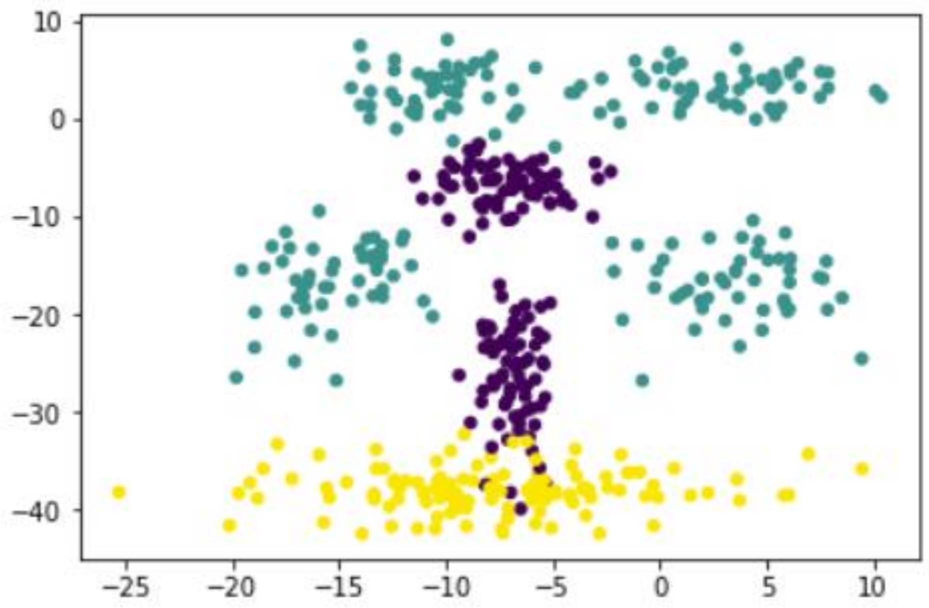




The confusion matrix of the best model is as follows:



3.



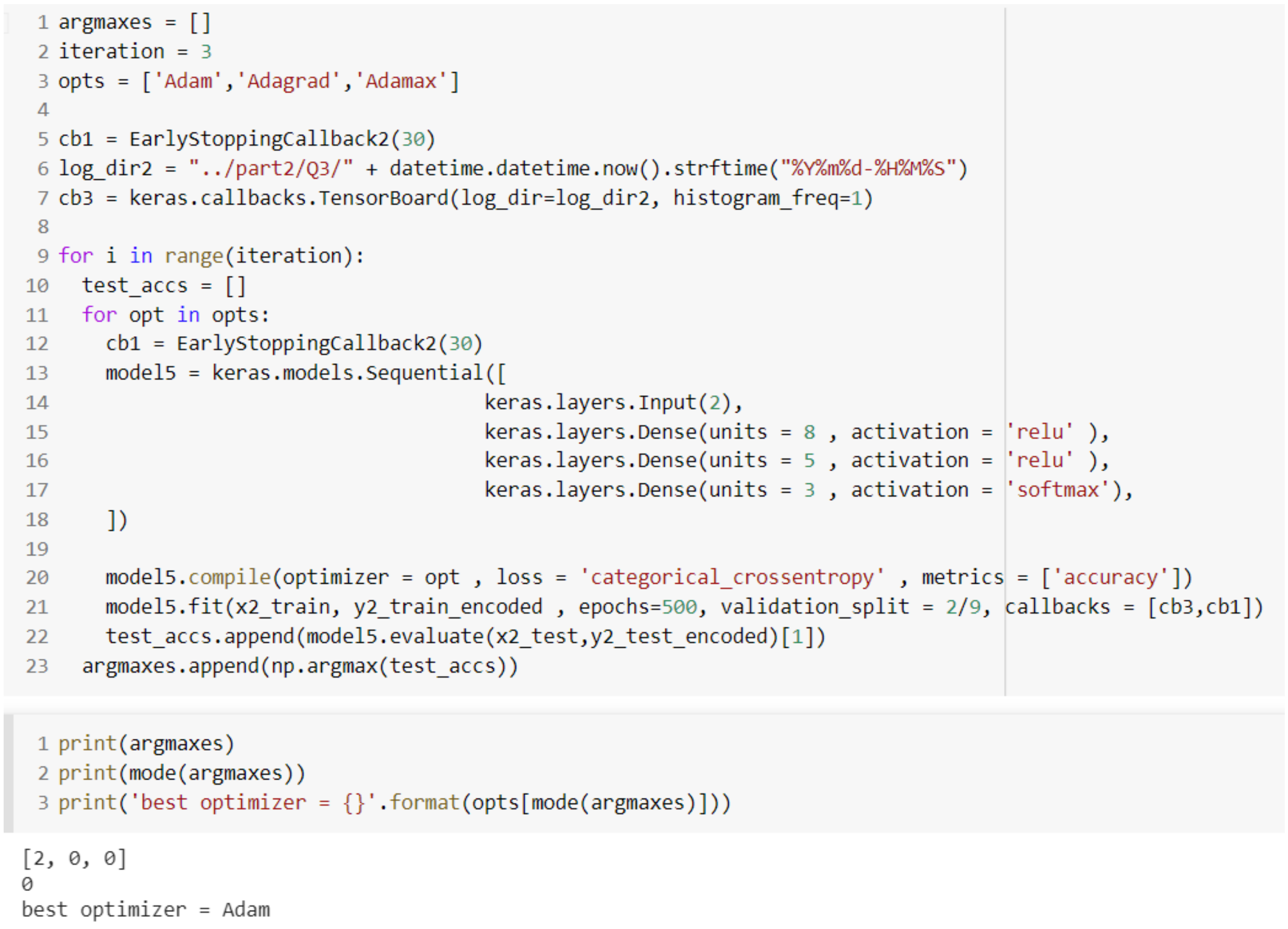
Firstly, considering the distribution of the data in the figure above, as the classes are increased to three classes compared to the previous problem and, more importantly, it seems that non-convex regions are needed to separate classes, a model with more than two hidden layers is needed; hence, for this problem a model with overall 4 layers was defined.

Secondly, for the output layer, three nodes with 'softmax' function was selected since the task is three-class classification; therefore, categorical-crossentropy loss function was selected since categorical-crossentropy loss function is the most compatible loss function with softmax activation function among other losses.

Thirdly, number of nodes in each hidden layer and the optimizer parameter were obtained by trial and error.

First experiment (optimizer selection):

In order to find the best optimizer, three optimizers were considered as candidates, namely, ‘Adam’, ‘Adagrad’, and ‘Adamax’ and the following script was implemented. That is, with three iterations, each time considering three mentioned optimizers, the optimizer with best final test accuracy among obtained test accuracies of three models was saved (for each iteration). After all, the most repeated optimizer was considered as the best optimizer. In this experiment ‘Adam’ optimizer determined as the best optimizer as shown below:

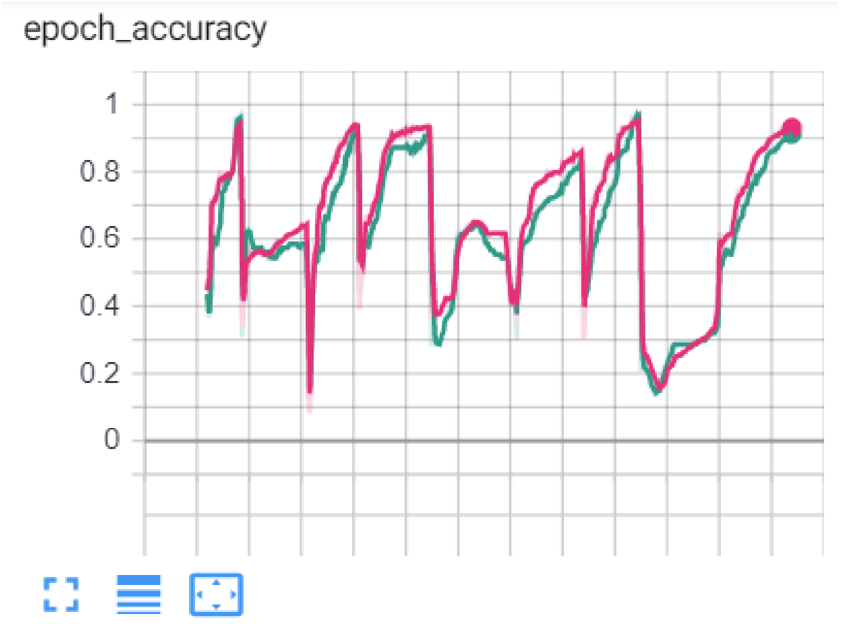


```python
argmaxes = []
iteration = 3
opts = ['Adam','Adagrad','Adamax']

cb1 = EarlyStoppingCallback2(30)
log_dir2 = "../part2/Q3/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
cb3 = keras.callbacks.TensorBoard(log_dir=log_dir2, histogram_freq=1)

for i in range(iteration):
  test_accs = []
  for opt in opts:
    cb1 = EarlyStoppingCallback2(30)
    model5 = keras.models.Sequential([
                                keras.layers.Input(2),
                                keras.layers.Dense(units = 8 , activation = 'relu' ),
                                keras.layers.Dense(units = 5 , activation = 'relu' ),
                                keras.layers.Dense(units = 3 , activation = 'softmax'),
    ])

    model5.compile(optimizer = opt , loss = 'categorical_crossentropy' , metrics = ['accuracy'])
    model5.fit(x2_train, y2_train_encoded , epochs=500, validation_split = 2/9, callbacks = [cb3,cb1])
    test_accs.append(model5.evaluate(x2_test,y2_test_encoded)[1])
  argmaxes.append(np.argmax(test_accs))
```

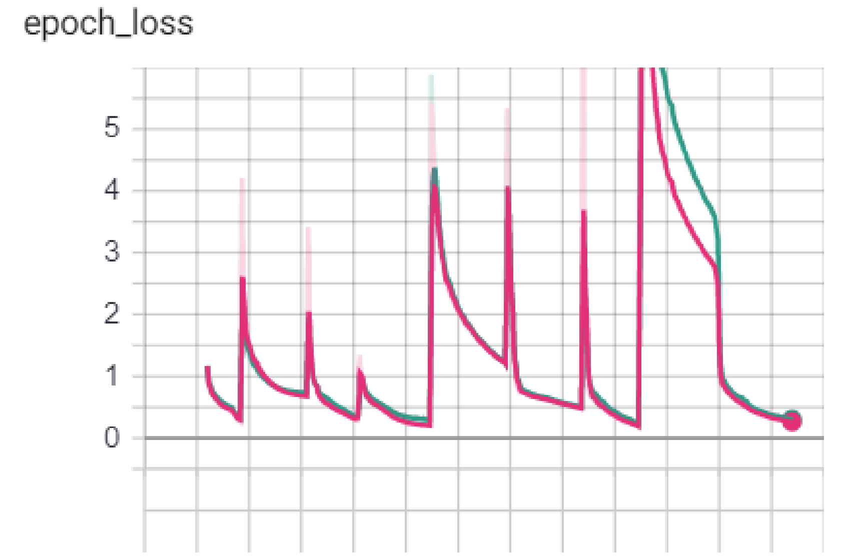
```python
print(argmaxes)
print(mode(argmaxes))
print('best optimizer = {}'.format(opts[mode(argmaxes)]))
```

```
[2, 0, 0]
0
best optimizer = Adam
```

The related log of this experiment (accuracies of all models in order) with 9 generated models (3 models for each iteration) are as follows:

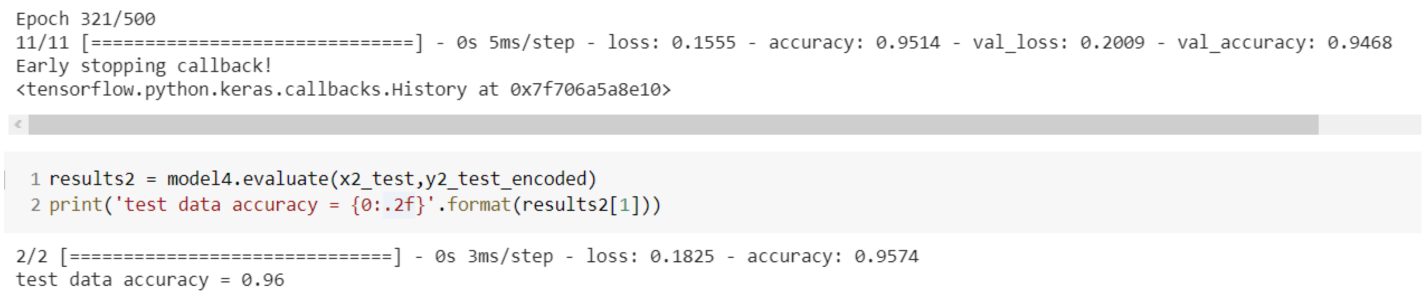


epoch_loss



Final best model evaluation:

Obtained best model reached to 96% accuracy on test data after 321 epochs with early stop call back as shown below:

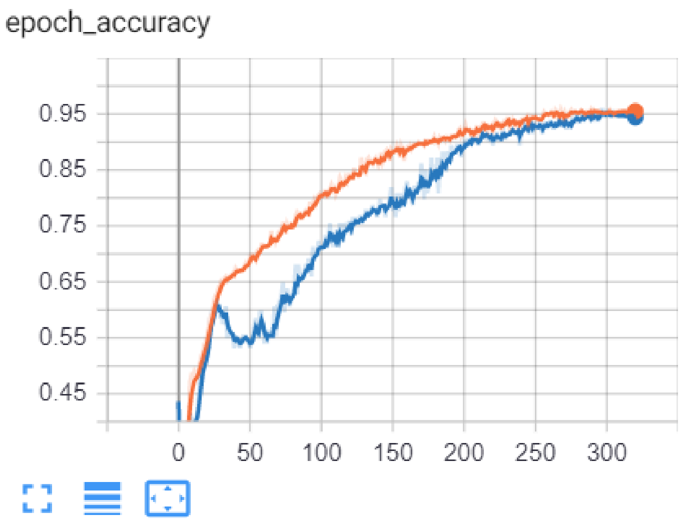


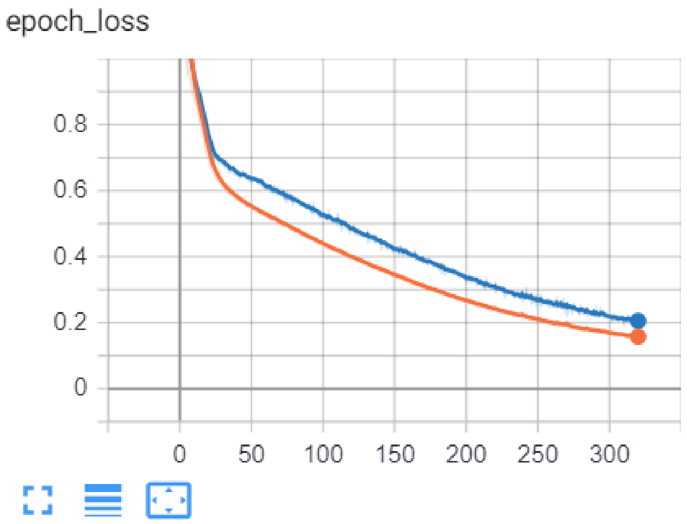
```
Epoch 321/500
11/11 [==============================] - 0s 5ms/step - loss: 0.1555 - accuracy: 0.9514 - val_loss: 0.2009 - val_accuracy: 0.9468
Early stopping callback!
<tensorflow.python.keras.callbacks.History at 0x7f706a5a8e10>
```

```
results2 = model4.evaluate(x2_test,y2_test_encoded)
print('test data accuracy = {0:.2f}'.format(results2[1]))
```

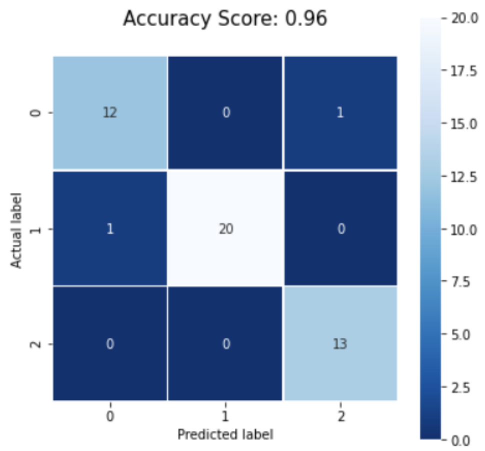
```
2/2 [==============================] - 0s 3ms/step - loss: 0.1825 - accuracy: 0.9574
test data accuracy = 0.96
```

The logs of the best model are as follows:

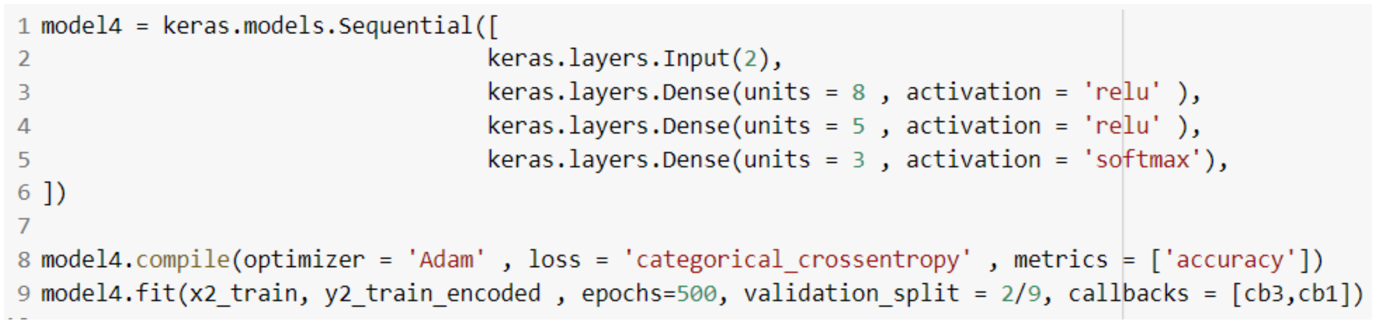




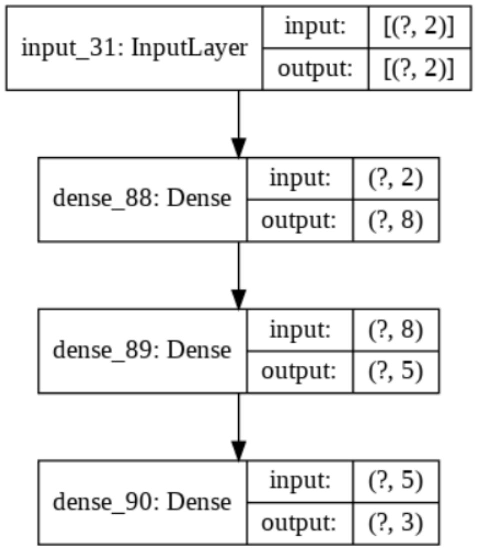
The confusion matrix of the best model is as follows:



The obtained best model configuration is as follows:



```
model4 = keras.models.Sequential([
                                    keras.layers.Input(2),
                                    keras.layers.Dense(units = 8 , activation = 'relu' ),
                                    keras.layers.Dense(units = 5 , activation = 'relu' ),
                                    keras.layers.Dense(units = 3 , activation = 'softmax'),
])

model4.compile(optimizer = 'Adam' , loss = 'categorical_crossentropy' , metrics = ['accuracy'])
model4.fit(x2_train, y2_train_encoded , epochs=500, validation_split = 2/9, callbacks = [cb3,cb1])
```



“Third Part”

1.

2.