4a) (5 points) **Data Generation:**

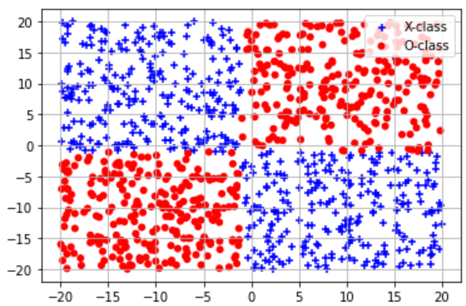
```
In [1]: #importing libraries
        import numpy as np
        import matplotlib.pyplot as plt
        from keras.models import Sequential
        from keras.layers import Dense
```

```
In [2]: #4a data generation
        N = 250 ;
        Uh=20;
        Ul= -1;

        x1 =np.concatenate([np.random.uniform(Ul, Uh, N), np.random.uniform(-Uh, Ul, N)])
        y1 =np.concatenate([np.random.uniform(-Uh, Ul, N), np.random.uniform(Ul, Uh, N)])

        x2 =np.concatenate([np.random.uniform(Ul, Uh, N), np.random.uniform(-Uh, Ul, N)])
        y2 =np.concatenate([np.random.uniform(Ul, Uh, N), np.random.uniform(-Uh, Ul, N)])

        plt.scatter(x1, y1,marker='+', c='blue', label='X-class')
        plt.scatter(x2, y2,marker='o', c='red',edgecolors ='none', label='o-class')
        #plt.show()
        plt.legend(["X-class", "O-class"])
        plt.grid(True)
```



b) (5 points) **Training Dataset Establishment:**

```
In [3]: #4b trainning data establishment (x_train)
        x = np.concatenate((x1, x2), axis=None)
        y = np.concatenate((y1, y2), axis=None)
        x = np.array(x)
        y = np.array(y)
        xy =np.vstack((x,y))
        x_train= xy.transpose()
        print(x_train)
```

```
[[  7.05010528 -16.12021039]
 [  6.01424623  -4.56865049]
 [ 14.0247238  -17.21127765]
 ...
 [ -7.84692454 -19.3408771 ]
 [-14.69889708  -5.27669173]
 [-17.13990842  -2.51916732]]
```

```
In [4]: #4b trainning data establishment (y_train)
        y_x = [1, 0]
        y_o =[0, 1]
        y_train= np.array([y_x,y_o])
        y_train=np.repeat(y_train, [2*N, 2*N], axis=0)
        print(y_train)
```

```
[[1 0]
 [1 0]
 [1 0]
 ...
 [0 1]
 [0 1]
 [0 1]]
```

(c, d) (5 points) **Model Setup:** (5 points) **Model Training:**

In [5]:

```
#4c Model Setup
model = Sequential()
model.add(Dense(8, input_dim=2, activation='relu'))
model.add(Dense(2, activation='sigmoid'))
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 8)                 24

 dense_1 (Dense)             (None, 2)                 18

=================================================================
Total params: 42
Trainable params: 42
Non-trainable params: 0
_________________________________________________________________
None
```

In [6]:

```
#4d Model Training
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=['accuracy'],
)
# fit the keras model on the dataset
history = model.fit(x_train, y_train, epochs=200, batch_size=10, verbose= 1)
```

```
Epoch 1/200
100/100 [==============================] - 0s 691us/step - loss: 2.3113 - accuracy: 0.5190
Epoch 2/200
100/100 [==============================] - 0s 642us/step - loss: 1.0787 - accuracy: 0.5720
Epoch 3/200
100/100 [==============================] - 0s 628us/step - loss: 0.6321 - accuracy: 0.6700
Epoch 4/200
100/100 [==============================] - 0s 597us/step - loss: 0.4833 - accuracy: 0.7170
Epoch 5/200
100/100 [==============================] - 0s 671us/step - loss: 0.4165 - accuracy: 0.7650
Epoch 6/200
100/100 [==============================] - 0s 671us/step - loss: 0.3773 - accuracy: 0.8060
Epoch 7/200
100/100 [==============================] - 0s 603us/step - loss: 0.3488 - accuracy: 0.8320
Epoch 8/200
100/100 [==============================] - 0s 624us/step - loss: 0.3263 - accuracy: 0.8630
Epoch 9/200
100/100 [==============================] - 0s 606us/step - loss: 0.3080 - accuracy: 0.8910
Epoch 10/200
```

e) (5 points) **Model Evaluation:**

In [7]:
```
#4e Model Evaluation
Nt = 75 ; # 150/2 each quadrant
Uh=20;
Ul= -1;

x1t =np.concatenate([np.random.uniform(Ul, Uh, Nt), np.random.uniform(-Uh, Ul, Nt)])
y1t =np.concatenate([np.random.uniform(-Uh, Ul, Nt), np.random.uniform(Ul, Uh, Nt)])

x2t =np.concatenate([np.random.uniform(Ul, Uh, Nt), np.random.uniform(-Uh, Ul, Nt)])
y2t =np.concatenate([np.random.uniform(Ul, Uh, Nt), np.random.uniform(-Uh, Ul, Nt)])

xt = np.concatenate((x1t, x2t), axis=None)
yt = np.concatenate((y1t, y2t), axis=None)
print(xt)
print(yt)

xt = np.array(xt)
yt = np.array(yt)
print(x)
print(y)
xyt =np.vstack((xt,yt))
x_test= xyt.transpose()
print(x_test)
```

```
 [ -4.18459049 -12.54199749]
 [-11.40074416 -16.88259877]
 [ -8.42709174  -2.1811314 ]
 [-16.69689896  -8.11438842]
 [ -4.51979674 -10.96430038]
 [-16.28798259 -10.65795338]
 [ -3.26881996 -11.27391464]
 [ -4.5107384   -8.85798949]
 [-17.63128935  -6.0794541 ]
 [-15.70684645  -7.5585757 ]
 [-15.6124232   -9.3331966 ]
 [-13.5027872  -18.39645875]
 [-11.763087   -16.70784693]
 [ -3.21766421  -5.34029121]
 [-10.42844306  -7.22122486]
 [-17.13851124 -10.82746535]
 [ -8.03514811  -4.76183157]
 [ -6.59514714 -13.50513296]
 [ -4.93266053  -5.31135168]]
```

In [8]:
```
#4e Model Evaluation
y_xt = [1, 0]
y_ot =[0, 1]
y_test= np.array([y_xt,y_ot])
y_test=np.repeat(y_test, [2*Nt, 2*Nt], axis=0)
print(y_test)
```

```
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
 [1 0]
```

In [9]:
```
#4e Model Evaluation
_, score = model.evaluate(x_test, y_test, verbose= 0)

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

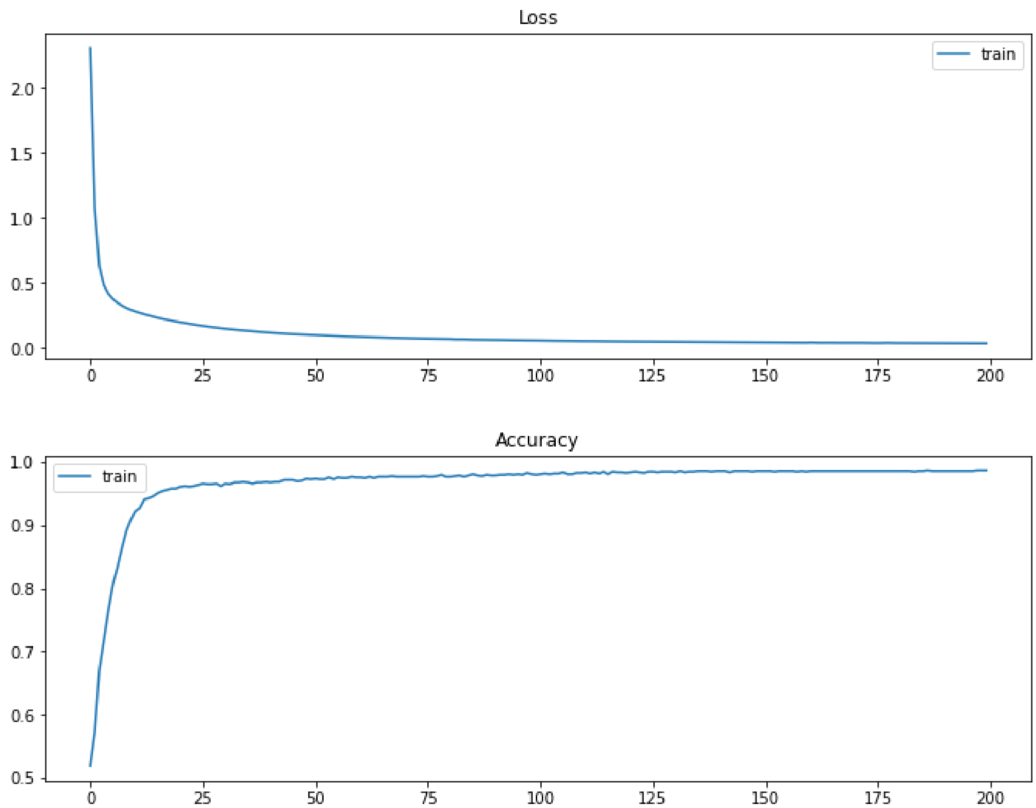
```
Accuracy : 0.9800000190734863
```

```
In [10]: figure, axes = plt.subplots(nrows=2, ncols=1, figsize = (10, 8))

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

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

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



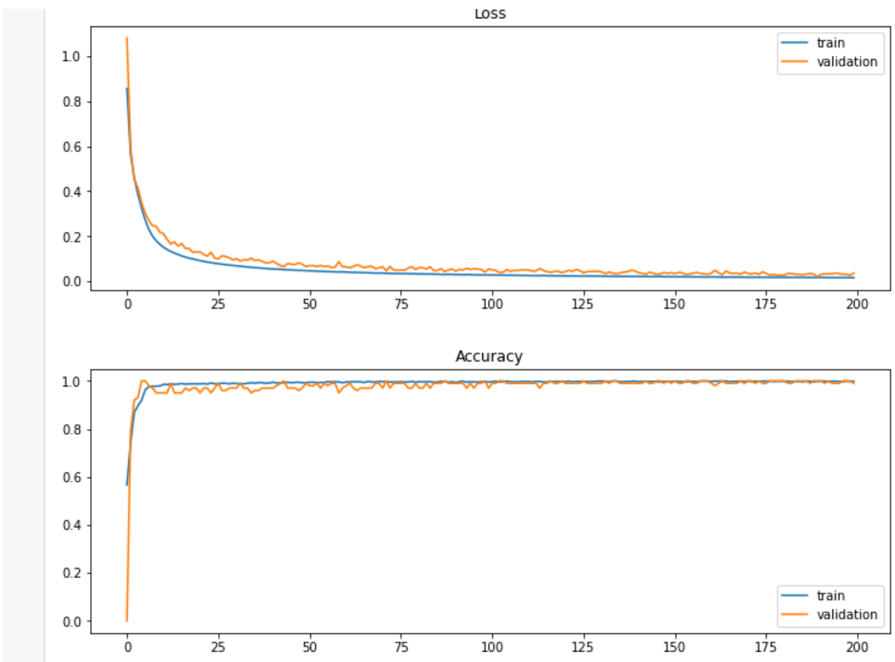
f) (8 points) **Code**: Submit your code as a python 3 file.

Source code submitted on the Webcampus .

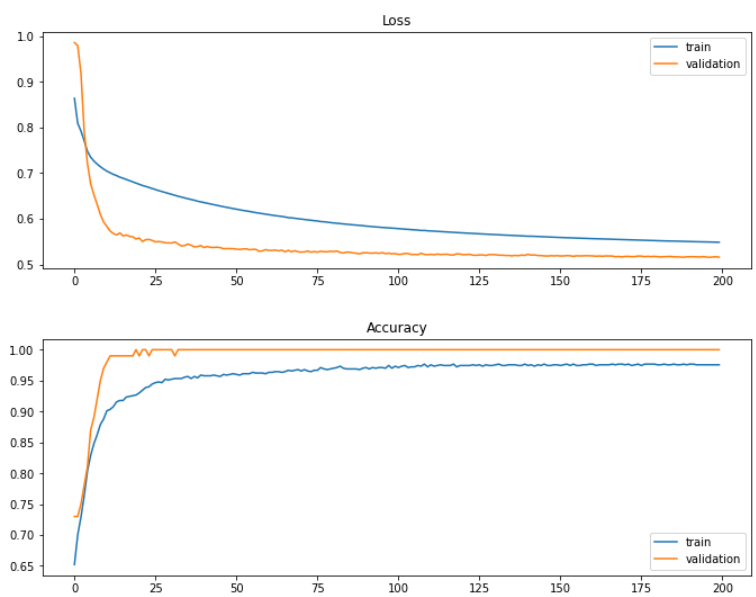
g) **Report:**

In this report we have used a deep network of 8 neuron in deep layer. In deep layer we used ‘relu’ function and in output layer we used ‘sigmoid’ function. Taking 10% of training set a validation set is created and loss has been calculated. The optimizer is ‘adam’ and we used ‘Binary cross entropy loss’ function.

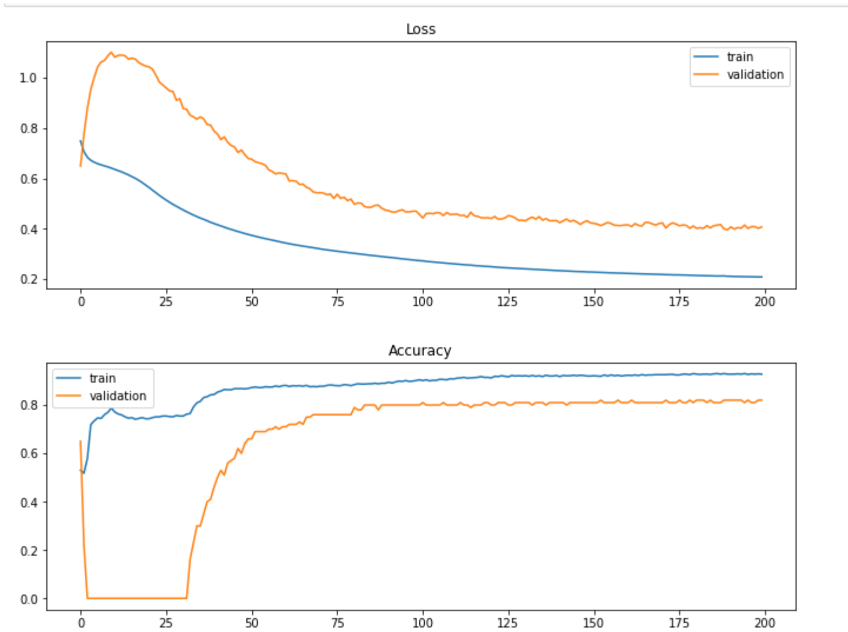
It seems for this case we get a very good accuracy : 0.99333



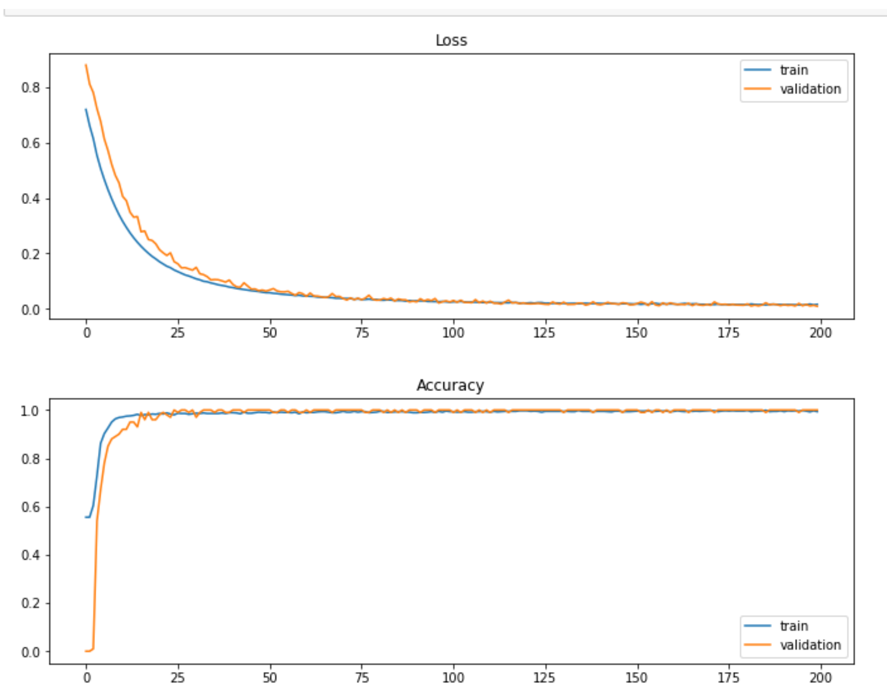
Changing the optimizer into ‘SGD’ and using ‘Hinge loss’ function, we observed that accuracy in the test set got lower.



Like our initial setup(adam optimizer and 'Binary cross entropy loss') , In the deeper layer we changed the activation function to sigmoid from relu and we observe that learning is slower than the previous case.



Adding another dense layer like the first setup , where the extra dense layer has 4 neuron and sigmoid activation function. We see that the accuracy increase slightly than the first case: 0.9966



(h) (10 points (bonus)) **Grad Only- plot decision boundary .**

Code:

```
In [11]: # 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(Dense(8, input_dim=2, activation='relu'))
         model_2.add(Dense(1, activation='sigmoid'))

         print(model_2.summary())
         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 = plt.get_cmap(cmap)

    # Define region of interest by data limits
    xmin, xmax = x1.min() - 1, y1.max() + 1
    ymin, ymax = x1.min() - 1, y1.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(figsize = (10, 8))
    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')
```

Output:

```
Epoch 198/200
100/100 [==============================] - 0s 587us/step - loss: 0.0220 - accuracy: 0.9950
Epoch 199/200
100/100 [==============================] - 0s 578us/step - loss: 0.0225 - accuracy: 0.9940
Epoch 200/200
100/100 [==============================] - 0s 586us/step - loss: 0.0218 - accuracy: 0.9940
Accuracy : 1.0
Out[11]: (<Figure size 720x576 with 1 Axes>, <AxesSubplot:>)
```