Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies 2 in CpE
2nd Semester	AY 2023-2024
Hands-on Activity 6.2 Training Neural Networks	
Name	Albo, Russel Zen D.
Section	CPE32S9
Date Performed:	April 09, 2024
Date Submitted:	April 09, 2024
Instructor:	Engr. Roman M. Richard

Objective(s):

This activity aims to demonstrate how to train neural networks using keras

Intended Learning Outcomes (ILOs):

- Demonstrate how to build and train neural networks
- Demonstrate how to evaluate and plot the model using training and validation loss

Resources:

- Jupyter Notebook

CI Pima Diabetes Dataset

- pima-indians-diabetes.csv

Procedures

Load the necessary libraries

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, precision_recall_curve, roc_auc_score, roc_curve, accuracy_score
from sklearn.ensemble import RandomForestClassifier

import seaborn as sns

%matplotlib inline
```

```
## Import Keras objects for Deep Learning

from keras.models  import Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.optimizers import Adam, SGD, RMSprop
```

Load the dataset

```
filepath = "pima-indians-diabetes.csv"
names = ["times_pregnant", "glucose_tolerance_test", "blood_pressure", "skin_thickness", "insulin",
         "bmi", "pedigree_function", "age", "has_diabetes"]
diabetes_df = pd.read_csv(filepath, names=names)
```

Check the top 5 samples of the data

```
print(diabetes_df.shape)
diabetes_df.sample(5)
```

(768, 9)

	times_pregnant	glucose_tolerance_test	blood_pressure	skin_thickness	insulin	bmi	pedigree_function	age	has_
748	3	187	70	22	200	36.4	0.408	36	
634	10	92	62	0	0	25.9	0.167	31	
292	2	128	78	37	182	43.3	1.224	31	
465	0	124	56	13	105	21.8	0.452	21	
760	2	88	58	26	16	28.4	0.766	22	

```
diabetes_df.dtypes
```

```
times_pregnant              int64
glucose_tolerance_test      int64
blood_pressure              int64
skin_thickness              int64
insulin                     int64
bmi                       float64
pedigree_function         float64
age                         int64
has_diabetes                int64
dtype: object
```

```
X = diabetes_df.iloc[:, :-1].values
y = diabetes_df["has_diabetes"].values
```

Split the data to Train, and Test (75%, 25%)

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=11111)
```

```
np.mean(y), np.mean(1-y)
```

```
(0.3489583333333333, 0.6510416666666666)
```

Build a single hidden layer neural network using 12 nodes. Use the sequential model with single layer network and input shape to 8.

Normalize the data

```
normalizer = StandardScaler()
X_train_norm = normalizer.fit_transform(X_train)
X_test_norm = normalizer.transform(X_test)
```

Define the model:

- Input size is 8-dimensional
- 1 hidden layer, 12 hidden nodes, sigmoid activation
- Final layer with one node and sigmoid activation (standard for binary classification)

```
model  = Sequential([
    Dense(12, input_shape=(8,), activation="relu"),
    Dense(1, activation="sigmoid")
])
```

View the model summary

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 12)                108

 dense_1 (Dense)             (None, 1)                 13

=================================================================
Total params: 121 (484.00 Byte)
Trainable params: 121 (484.00 Byte)
Non-trainable params: 0 (0.00 Byte)
```

Train the model

- Compile the model with optimizer, loss function and metrics
- Use the fit function to return the run history.

```
model.compile(SGD(lr = .003), "binary_crossentropy", metrics=["accuracy"])
run_hist_1 = model.fit(X_train_norm, y_train, validation_data=(X_test_norm, y_test), epochs=200)
```

```
===============] - 0s 3ms/step - loss: 0.6480 - accuracy: 0.6042 - val_loss: 0.6593 - val_accuracy: 0.5885

===============] - 0s 3ms/step - loss: 0.6340 - accuracy: 0.6354 - val_loss: 0.6471 - val_accuracy: 0.6094

===============] - 0s 4ms/step - loss: 0.6214 - accuracy: 0.6597 - val_loss: 0.6361 - val_accuracy: 0.6302

===============] - 0s 3ms/step - loss: 0.6100 - accuracy: 0.6684 - val_loss: 0.6263 - val_accuracy: 0.6406

===============] - 0s 3ms/step - loss: 0.5996 - accuracy: 0.6771 - val_loss: 0.6174 - val_accuracy: 0.6406

===============] - 0s 4ms/step - loss: 0.5903 - accuracy: 0.6858 - val_loss: 0.6093 - val_accuracy: 0.6510

===============] - 0s 3ms/step - loss: 0.5815 - accuracy: 0.6927 - val_loss: 0.6020 - val_accuracy: 0.6510

===============] - 0s 4ms/step - loss: 0.5737 - accuracy: 0.6962 - val_loss: 0.5952 - val_accuracy: 0.6562

===============] - 0s 3ms/step - loss: 0.5665 - accuracy: 0.6944 - val_loss: 0.5890 - val_accuracy: 0.6719

===============] - 0s 4ms/step - loss: 0.5598 - accuracy: 0.7066 - val_loss: 0.5834 - val_accuracy: 0.6719

===============] - 0s 4ms/step - loss: 0.5536 - accuracy: 0.7135 - val_loss: 0.5782 - val_accuracy: 0.6719

===============] - 0s 4ms/step - loss: 0.5480 - accuracy: 0.7153 - val_loss: 0.5734 - val_accuracy: 0.6823

===============] - 0s 6ms/step - loss: 0.5428 - accuracy: 0.7170 - val_loss: 0.5690 - val_accuracy: 0.6927

===============] - 0s 6ms/step - loss: 0.5378 - accuracy: 0.7205 - val_loss: 0.5650 - val_accuracy: 0.6979

===============] - 0s 5ms/step - loss: 0.5333 - accuracy: 0.7222 - val_loss: 0.5613 - val_accuracy: 0.6979

===============] - 0s 5ms/step - loss: 0.5290 - accuracy: 0.7240 - val_loss: 0.5580 - val_accuracy: 0.7031

===============] - 0s 5ms/step - loss: 0.5252 - accuracy: 0.7309 - val_loss: 0.5549 - val_accuracy: 0.7083

===============] - 0s 5ms/step - loss: 0.5216 - accuracy: 0.7326 - val_loss: 0.5520 - val_accuracy: 0.7188

===============] - 0s 5ms/step - loss: 0.5183 - accuracy: 0.7344 - val_loss: 0.5493 - val_accuracy: 0.7240

===============] - 0s 4ms/step - loss: 0.5150 - accuracy: 0.7361 - val_loss: 0.5467 - val_accuracy: 0.7292

===============] - 0s 4ms/step - loss: 0.5120 - accuracy: 0.7344 - val_loss: 0.5443 - val_accuracy: 0.7344

===============] - 0s 5ms/step - loss: 0.5093 - accuracy: 0.7309 - val_loss: 0.5421 - val_accuracy: 0.7344

===============] - 0s 4ms/step - loss: 0.5067 - accuracy: 0.7361 - val_loss: 0.5400 - val_accuracy: 0.7344

===============] - 0s 4ms/step - loss: 0.5043 - accuracy: 0.7396 - val_loss: 0.5381 - val_accuracy: 0.7344

===============] - 0s 4ms/step - loss: 0.5020 - accuracy: 0.7413 - val_loss: 0.5363 - val_accuracy: 0.7344

===============] - 0s 4ms/step - loss: 0.4998 - accuracy: 0.7396 - val_loss: 0.5346 - val_accuracy: 0.7448

===============] - 0s 4ms/step - loss: 0.4977 - accuracy: 0.7413 - val_loss: 0.5330 - val_accuracy: 0.7448

===============] - 0s 5ms/step - loss: 0.4957 - accuracy: 0.7396 - val_loss: 0.5316 - val_accuracy: 0.7448

===============] - 0s 5ms/step - loss: 0.4941 - accuracy: 0.7413 - val_loss: 0.5303 - val_accuracy: 0.7396
```

```
## Like we did for the Random Forest, we generate two kinds of predictions
#  One is a hard decision, the other is a probabilitistic score.

y_pred_class_nn_1 = np.argmax(model.predict(X_test_norm), axis=-1)
y_pred_prob_nn_1 = model.predict(X_test_norm)
```

```
6/6 [==============================] - 0s 5ms/step
6/6 [==============================] - 0s 3ms/step
```

```
# Let's check out the outputs to get a feel for how keras apis work.
y_pred_class_nn_1[:10]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
y_pred_prob_nn_1[:10]
```

```
array([[0.52686846],
       [0.5850985 ],
       [0.27110088],
       [0.30811527],
       [0.14348966],
       [0.5298821 ],
       [0.02847539],
       [0.24657233],
       [0.9109988 ],
       [0.19329573]], dtype=float32)
```

Create the plot_roc function

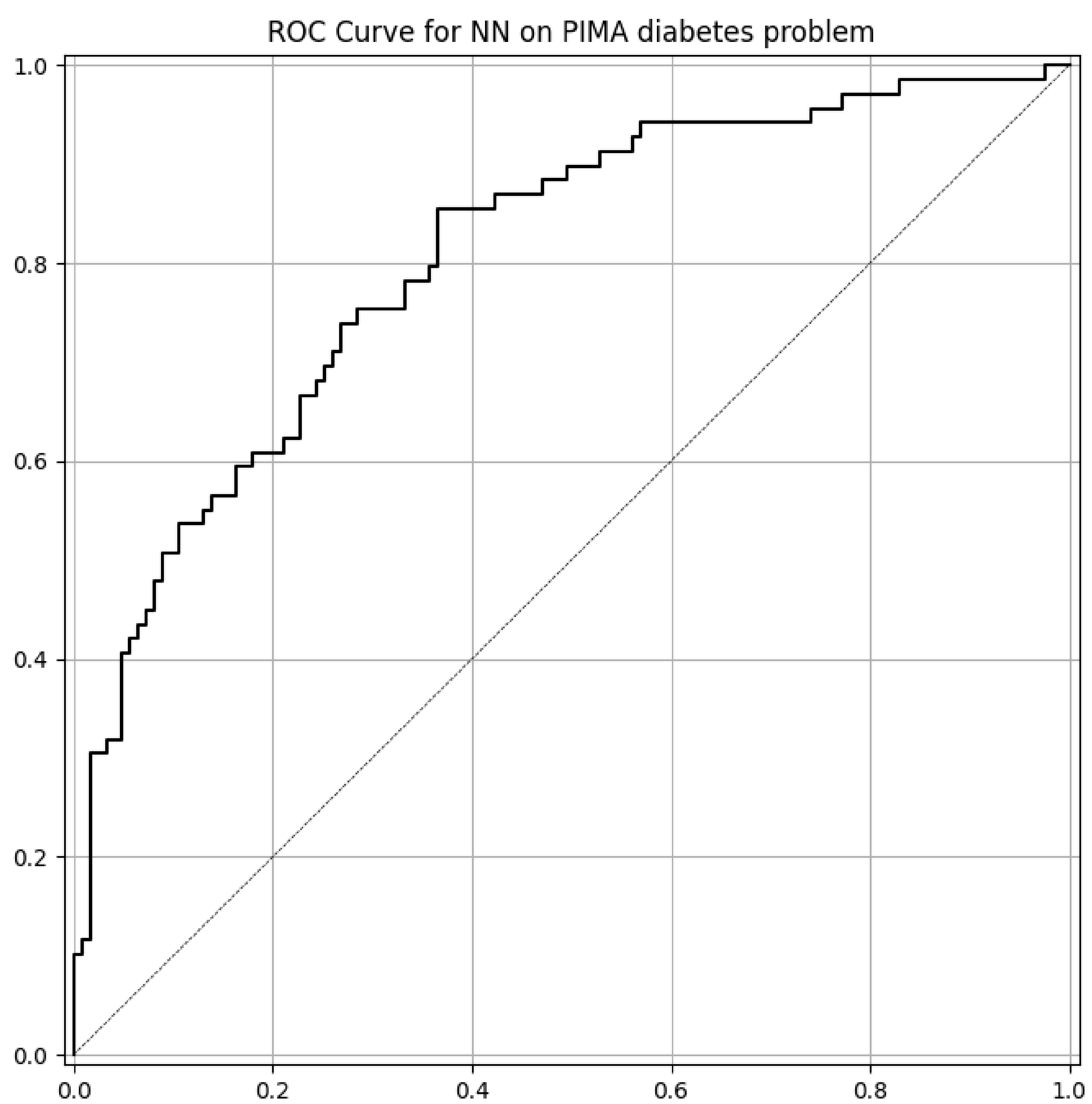
```
def plot_roc(y_test, y_pred, model_name):
    fpr, tpr, thr = roc_curve(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.plot(fpr, tpr, 'k-')
    ax.plot([0, 1], [0, 1], 'k--', linewidth=.5)  # roc curve for random model
    ax.grid(True)
    ax.set(title='ROC Curve for {} on PIMA diabetes problem'.format(model_name),
           xlim=[-0.01, 1.01], ylim=[-0.01, 1.01])
```

Evaluate the model performance and plot the ROC CURVE

```
print('accuracy is {:.3f}'.format(accuracy_score(y_test,y_pred_class_nn_1)))
print('roc-auc is {:.3f}'.format(roc_auc_score(y_test,y_pred_prob_nn_1)))

plot_roc(y_test, y_pred_prob_nn_1, 'NN')
```

```
accuracy is 0.641
roc-auc is 0.806
```



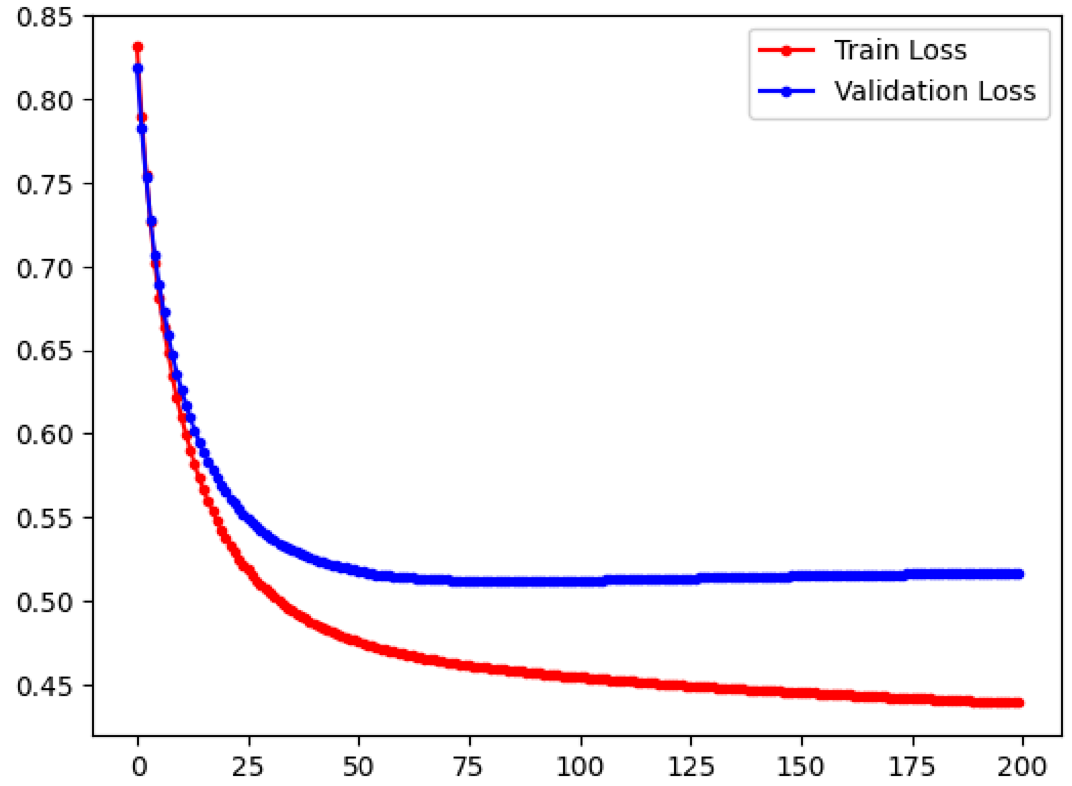
Plot the training loss and the validation loss over the different epochs and see how it looks

```
run_hist_1.history.keys()
```

```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

```
fig, ax = plt.subplots()
ax.plot(run_hist_1.history["loss"],'r', marker='.', label="Train Loss")
ax.plot(run_hist_1.history["val_loss"],'b', marker='.', label="Validation Loss")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7b6fd21dc790>
```



What is your interpretation about the result of the train and validation loss?

- In plotting about the result of train and validation loss, I notice that it is decreasing since train and validation are not close to each other.

Supplementary Activity

- Build a model with two hidden layers, each with 6 nodes
- Use the "relu" activation function for the hidden layers, and "sigmoid" for the final layer
- Use a learning rate of .003 and train for 1500 epochs
- Graph the trajectory of the loss functions, accuracy on both train and test set
- Plot the roc curve for the predictions
- Use different learning rates, numbers of epochs, and network structures.
- Plot the results of training and validation loss using different learning rates, number of epocgs and network structures
- Interpret your result

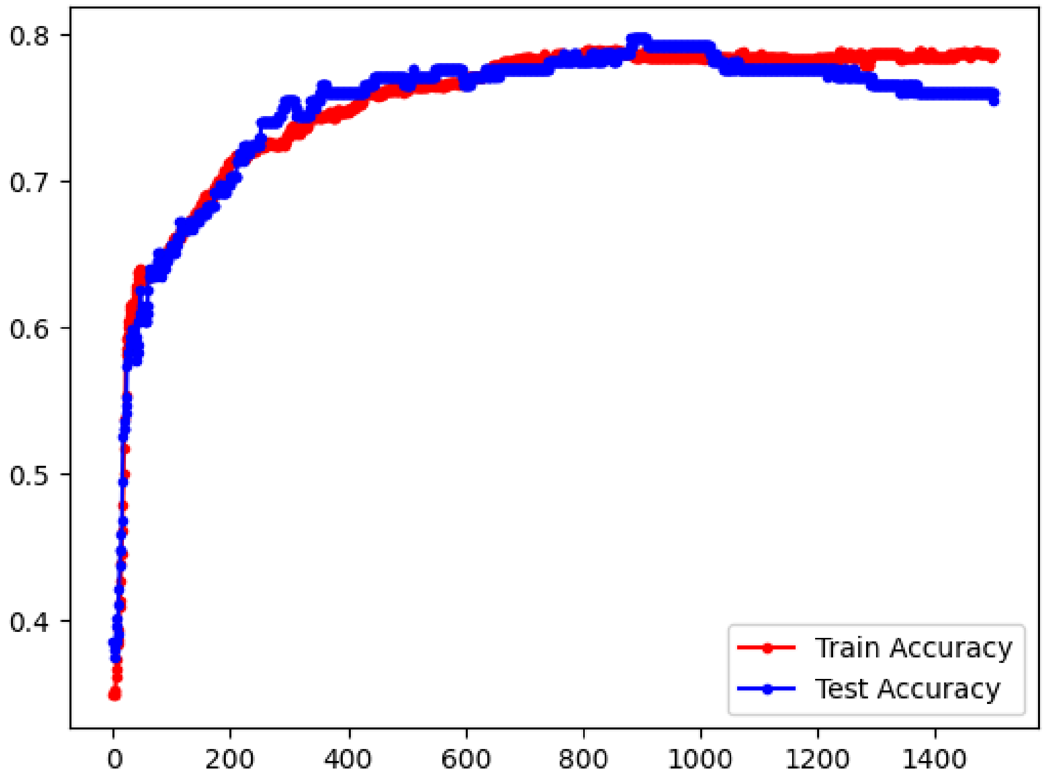
```
#Build a model with two hidden layers, each with 6 nodes
#Use the "relu" activation function for the hidden layers, and "sigmoid" for the final layer
model  = Sequential([
    Dense(6, input_shape=(8,), activation="relu"),
    Dense(6, activation="relu"),
    Dense(1, activation='sigmoid')
])
```

```
#Use a learning rate of .003 and train for 1500 epochs
model.compile(SGD(lr = .003), "binary_crossentropy", metrics=["accuracy"])
run_hist_2 = model.fit(X_train_norm, y_train, batch_size=250, validation_data=(X_test_norm, y_test), epochs=1500)
```

```
Epoch 1483/1500
3/3 [==============================] - 0s 30ms/step - loss: 0.4345 - accuracy: 0.7865 - val_loss: 0.5001 - val_accurac
Epoch 1484/1500
3/3 [==============================] - 0s 22ms/step - loss: 0.4345 - accuracy: 0.7865 - val_loss: 0.5002 - val_accurac
Epoch 1485/1500
3/3 [==============================] - 0s 22ms/step - loss: 0.4344 - accuracy: 0.7865 - val_loss: 0.5003 - val_accurac
Epoch 1486/1500
3/3 [==============================] - 0s 32ms/step - loss: 0.4345 - accuracy: 0.7865 - val_loss: 0.5004 - val_accurac
Epoch 1487/1500
3/3 [==============================] - 0s 29ms/step - loss: 0.4345 - accuracy: 0.7865 - val_loss: 0.5004 - val_accurac
Epoch 1488/1500
3/3 [==============================] - 0s 32ms/step - loss: 0.4344 - accuracy: 0.7865 - val_loss: 0.5004 - val_accurac
Epoch 1489/1500
3/3 [==============================] - 0s 23ms/step - loss: 0.4344 - accuracy: 0.7865 - val_loss: 0.5004 - val_accurac
Epoch 1490/1500
3/3 [==============================] - 0s 33ms/step - loss: 0.4344 - accuracy: 0.7865 - val_loss: 0.5003 - val_accurac
Epoch 1491/1500
3/3 [==============================] - 0s 23ms/step - loss: 0.4344 - accuracy: 0.7865 - val_loss: 0.5005 - val_accurac
Epoch 1492/1500
3/3 [==============================] - 0s 25ms/step - loss: 0.4343 - accuracy: 0.7865 - val_loss: 0.5004 - val_accurac
Epoch 1493/1500
3/3 [==============================] - 0s 35ms/step - loss: 0.4343 - accuracy: 0.7865 - val_loss: 0.5004 - val_accurac
Epoch 1494/1500
3/3 [==============================] - 0s 26ms/step - loss: 0.4343 - accuracy: 0.7865 - val_loss: 0.5005 - val_accurac
Epoch 1495/1500
3/3 [==============================] - 0s 32ms/step - loss: 0.4343 - accuracy: 0.7865 - val_loss: 0.5004 - val_accurac
Epoch 1496/1500
3/3 [==============================] - 0s 36ms/step - loss: 0.4343 - accuracy: 0.7847 - val_loss: 0.5005 - val_accurac
Epoch 1497/1500
3/3 [==============================] - 0s 23ms/step - loss: 0.4343 - accuracy: 0.7865 - val_loss: 0.5006 - val_accurac
Epoch 1498/1500
3/3 [==============================] - 0s 33ms/step - loss: 0.4342 - accuracy: 0.7865 - val_loss: 0.5005 - val_accurac
Epoch 1499/1500
3/3 [==============================] - 0s 34ms/step - loss: 0.4342 - accuracy: 0.7865 - val_loss: 0.5006 - val_accurac
Epoch 1500/1500
3/3 [==============================] - 0s 33ms/step - loss: 0.4342 - accuracy: 0.7865 - val_loss: 0.5007 - val_accurac
```

```
#Graph the trajectory of the loss functions, accuracy on both train and test set
fig, ax = plt.subplots()
ax.plot(run_hist_2.history["accuracy"],'r', marker='.', label="Train Accuracy")
ax.plot(run_hist_2.history["val_accuracy"],'b', marker='.', label="Test Accuracy")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7b6fc1cbcbb0>
```

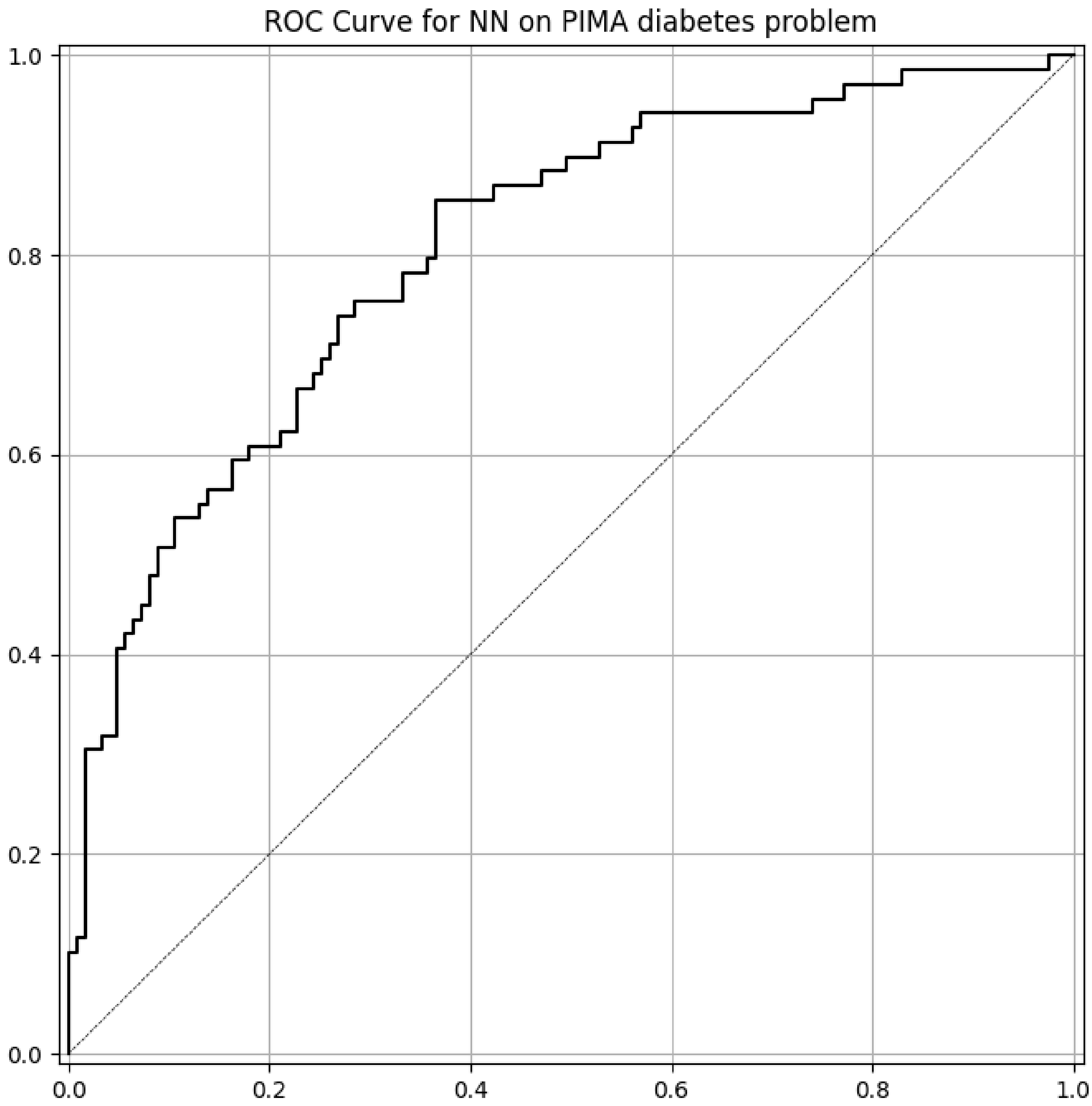



```
#Plot the roc curve for the predictions
def plot_roc(y_test, y_pred, model_name):
    fpr, tpr, thr = roc_curve(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.plot(fpr, tpr, 'k-')
    ax.plot([0, 1], [0, 1], 'k--', linewidth=.5)  # roc curve for random model
    ax.grid(True)
    ax.set(title='ROC Curve for {} on PIMA diabetes problem'.format(model_name),
           xlim=[-0.01, 1.01], ylim=[-0.01, 1.01])

print('accuracy is {:.3f}'.format(accuracy_score(y_test,y_pred_class_nn_1)))
print('roc-auc is {:.3f}'.format(roc_auc_score(y_test,y_pred_prob_nn_1)))

plot_roc(y_test, y_pred_prob_nn_1, 'NN')
```

```
accuracy is 0.641
roc-auc is 0.806
```



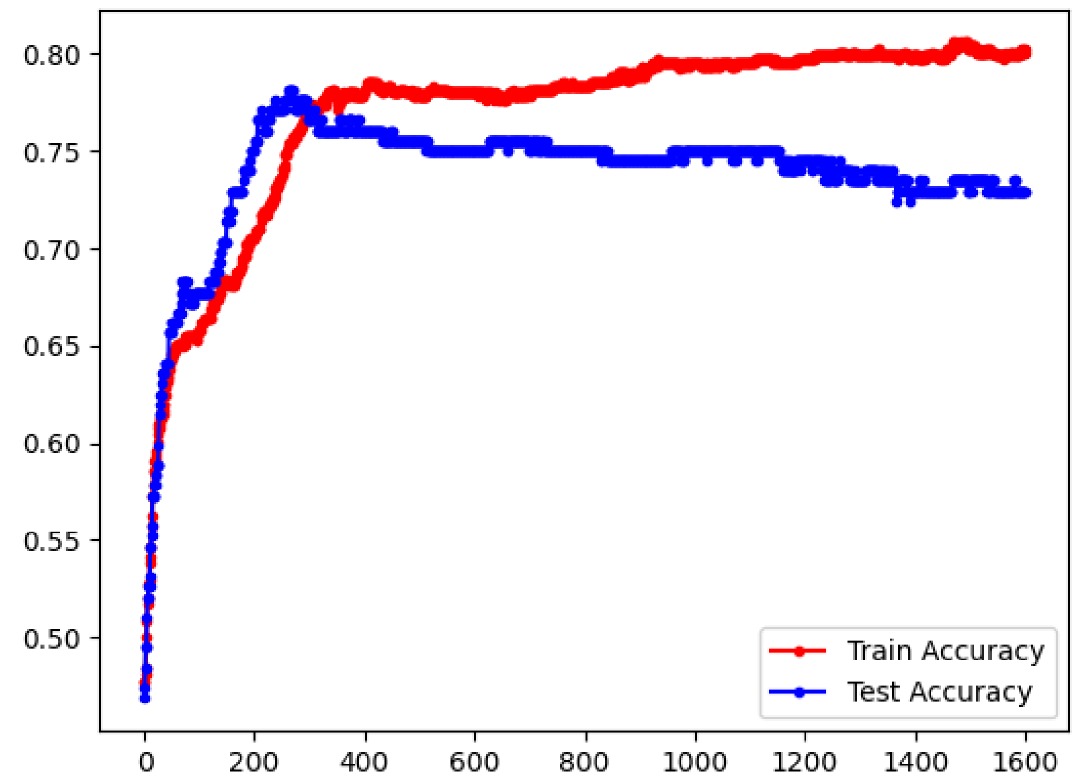

```
#Use different learning rates, numbers of epochs, and network structures.
model  = Sequential([
    Dense(6, input_shape=(8,), activation="relu"),
    Dense(6, activation="relu"),
    Dense(1, activation='sigmoid')
])

model.compile(SGD(lr = .005), "binary_crossentropy", metrics=["accuracy"])
run_hist_3 = model.fit(X_train_norm, y_train, batch_size=250, validation_data=(X_test_norm, y_test), epochs=1600)
```

```
3/3 [==============================] - 0s 24ms/step - loss: 0.4282 - accuracy: 0.8003 - val_loss: 0.5252 - val_accurac
Epoch 1592/1600
3/3 [==============================] - 0s 34ms/step - loss: 0.4282 - accuracy: 0.8003 - val_loss: 0.5253 - val_accurac
Epoch 1593/1600
3/3 [==============================] - 0s 24ms/step - loss: 0.4282 - accuracy: 0.8003 - val_loss: 0.5253 - val_accurac
Epoch 1594/1600
3/3 [==============================] - 0s 29ms/step - loss: 0.4282 - accuracy: 0.8021 - val_loss: 0.5253 - val_accurac
Epoch 1595/1600
3/3 [==============================] - 0s 30ms/step - loss: 0.4282 - accuracy: 0.8003 - val_loss: 0.5254 - val_accurac
Epoch 1596/1600
3/3 [==============================] - 0s 33ms/step - loss: 0.4282 - accuracy: 0.8003 - val_loss: 0.5256 - val_accurac
Epoch 1597/1600
3/3 [==============================] - 0s 32ms/step - loss: 0.4281 - accuracy: 0.8021 - val_loss: 0.5255 - val_accurac
Epoch 1598/1600
3/3 [==============================] - 0s 26ms/step - loss: 0.4281 - accuracy: 0.8021 - val_loss: 0.5254 - val_accurac
Epoch 1599/1600
3/3 [==============================] - 0s 31ms/step - loss: 0.4281 - accuracy: 0.8021 - val_loss: 0.5254 - val_accurac
Epoch 1600/1600
3/3 [==============================] - 0s 30ms/step - loss: 0.4281 - accuracy: 0.8003 - val_loss: 0.5256 - val_accurac
```

```
#Plot the results of training and validation loss using different learning rates, number of epocgs and network structures
fig, ax = plt.subplots()
ax.plot(run_hist_3.history["accuracy"],'r', marker='.', label="Train Accuracy")
ax.plot(run_hist_3.history["val_accuracy"],'b', marker='.', label="Test Accuracy")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x7b6fd1f28f40>
```



Interpret your result

- I noticed that the overall result I had using the number of epochs I put and in train and test accuracy has been increased since the two accuracy are closed to each other.

Conclusion

- In this activity, I learned and understand how to build train neural networks with evaluating and plotting it using training validation accuracy. I noticed the difference between the csv file and the supplementary that i do when the plot in csv file showing the result is underfit between each other and the second one in the supplementary activity is overfit. Also I notice that underfit is the result of test accuracy is higher than train accuracy, while overfit is the result of train is higher than test.