Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2
2nd Semester	AY 2023-2024
Hands-on Activity 6.2	**Training Neural Networks**
Name	Guevarra, Hans Angelo C.
Section	CPE32S3
Date Performed:	4-1-24
Date Submitted:	4-1-24
Instructor:	Engr. Roman M. Richard

Activity 1.2 : Training Neural Networks

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

In [1]:

```
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
```

In [2]:

```
## 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

In [3]:

```
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

In [4]:

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

```
(768, 9)
```

Out[4]:

	times_pregnant	glucose_tolerance_test	blood_pressure	skin_thickness	insulin	bmi	pedigree_function	age	has_diabet
589	0	73	0	0	0	21.1	0.342	25	
362	5	103	108	37	0	39.2	0.305	65	
488	4	99	72	17	0	25.6	0.294	28	
475	0	137	84	27	0	27.3	0.231	59	
465	0	124	56	13	105	21.8	0.452	21	

In [5]:

```
diabetes_df.dtypes
```

Out[5]:

```
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
```

In [6]:

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

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

In [7]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1
1111)
```

In [8]:

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

Out[8]:

```
(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

In [9]:

```
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)**

In [10]:

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

View the model summary

In [11]:

```
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.**

In [ ]:

```
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), epo
chs=200)
```

In [21]:

```python
## 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_prob_nn_1 = model.predict(X_test_norm)
# Convert probabilities to classes based on a threshold (e.g., 0.5)
y_pred_class_nn_1 = (y_pred_prob_nn_1 > 0.5).astype(int)
```

```
6/6 [==============================] - 0s 4ms/step
```

In [22]:

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

Out[22]:

```
array([[1],
       [1],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [1],
       [0]])
```

In [23]:

```python
y_pred_prob_nn_1[:10]
```

Out[23]:

```
array([[0.5580696 ],
       [0.6834491 ],
       [0.23982008],
       [0.23846155],
       [0.09017289],
       [0.52163357],
       [0.02463016],
       [0.23529987],
       [0.9169272 ],
       [0.22186932]], dtype=float32)
```

Create the plot_roc function

In [24]:

```python
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

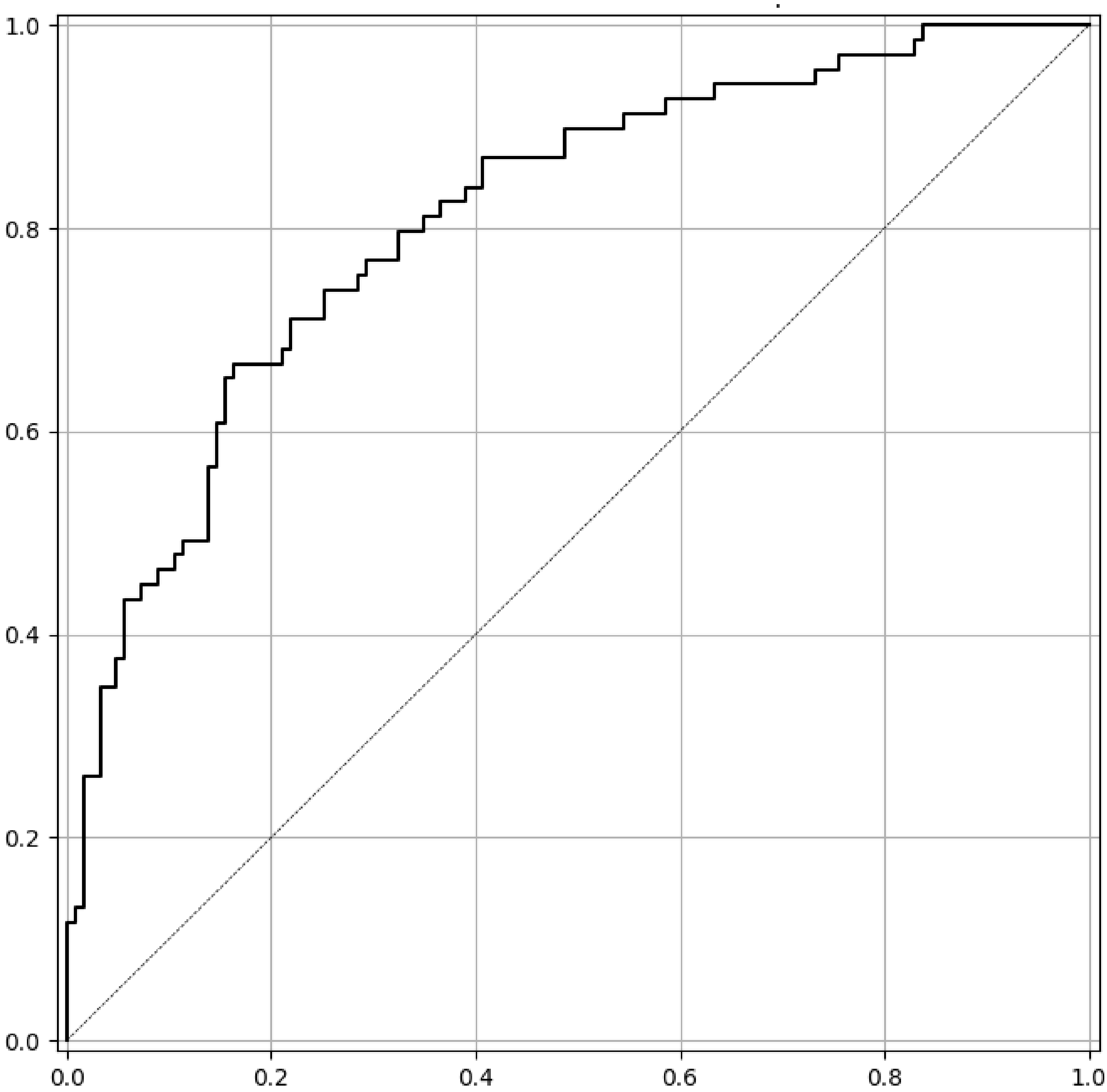
In [25]:

```python
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 0.760
roc-auc is 0.811
```

ROC Curve for NN on PIMA diabetes problem



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

In [26]:

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

Out[26]:

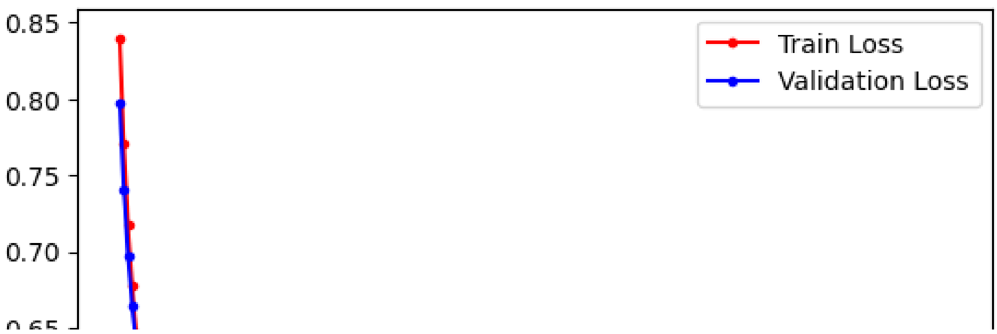
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

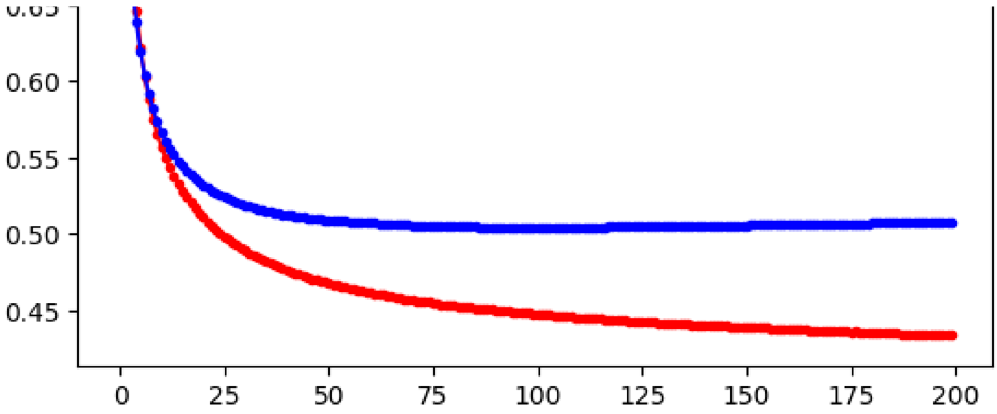
In [27]:

```
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()
```

Out[27]:

```
<matplotlib.legend.Legend at 0x7b0b1fec14b0>
```





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

I think the result is balanced because the train loss and validation loss in 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**

In [41]:

```
# 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 l
ayer

model = Sequential([
    Dense(6, input_shape=(8,), activation="relu"),
    Dense(6, activation="relu"),
    Dense(1, activation="sigmoid")
])
```

In [ ]:

```
# Use a learning rate of .003 and train for 1500 epochs

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), epo
chs=1500)
```

In [45]:

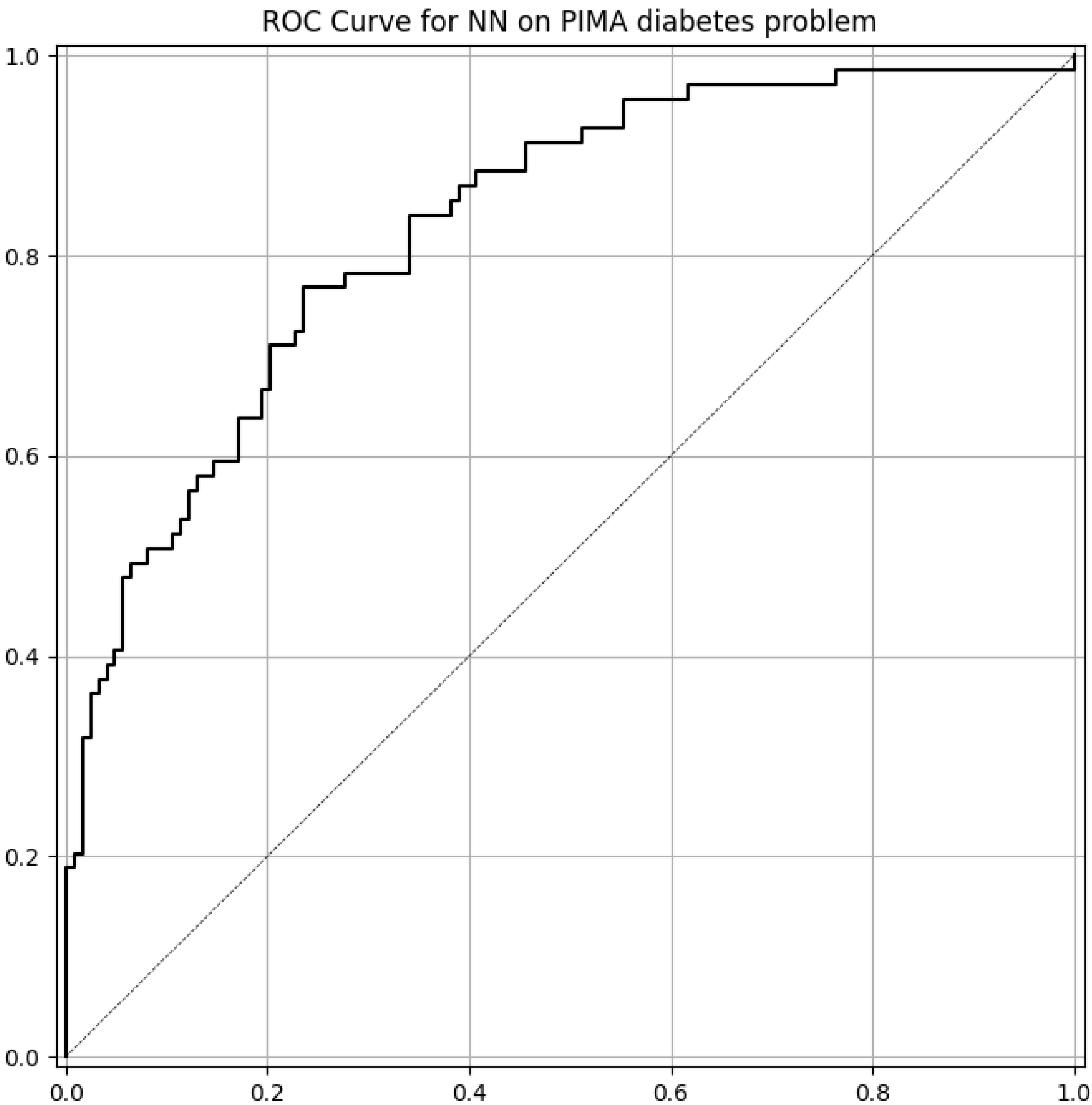
```
def plot_roc(y_test, y_pred, model_name):
    fpr, tpr, thr = roc_curve(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.plot(fpr, tpr, 'k-')
    ax.plot([0, 1], [0, 1], 'k--', linewidth=.5)  # roc curve for random model
    ax.grid(True)
    ax.set(title='ROC Curve for {} on PIMA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

In [46]:

```
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 0.760
roc-auc is 0.831
```



In [43]:

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

Out[43]:

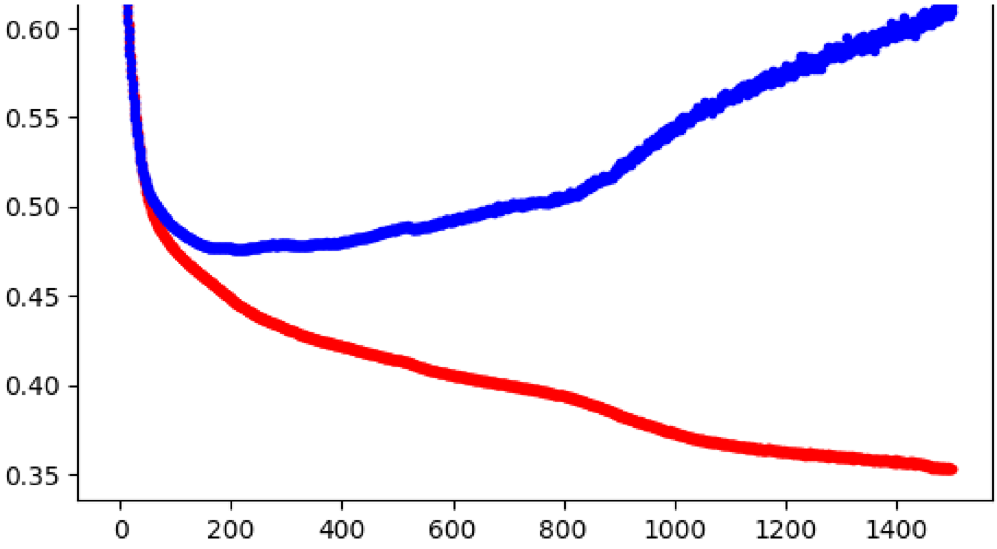
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

In [44]:

```
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()
```

Out[44]:

```
<matplotlib.legend.Legend at 0x7b0b1d7e5de0>
```



OBSERVATION: Based on the plot above, the validation loss is significantly higher than the train loss which means that it is overfitting due to the high number of epochs.

Conclusion

After doing this activity, I was able to perform training neural networks and evaluate them by showing the training and validation loss based on the number of epochs. I was also able to learn the relationship between the train loss, validation loss, and the number of epochs wherein a large number of epochs would result to overfitting which means there would be a significant difference between the train loss and validation loss.