Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2
2nd Semester	AY 2024 - 2025

ACTIVITY NO. 6.2	**Training Neural Networks**
Name	Calvadores, Kelly Joseph
Section	CPE32S3
Date Performed:	March 30, 2024
Date Submitted:	April 2, 2024
Instructor:	Engr. Roman M. Richard

```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
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
from keras.models  import Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.optimizers import Adam, SGD, RMSprop
%matplotlib inline
```

Load Dataset

```
names = ["times_pregnant", "glucose_tolerance_test", "blood_pressure", "skin_thickness", "insulin",
         "bmi", "pedigree_function", "age", "has_diabetes"]
diabetes_df = pd.read_csv("pima-indians-diabetes.csv", 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
349	5	0	80	32	0
187	1	128	98	41	58
517	7	125	86	0	0
665	1	112	80	45	132
230	4	142	86	0	0

```
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)
```

Normalize the data

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

Define the model:

```
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

```
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)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimiz
Epoch 1/200
18/18 [==============================] - 1s 14ms/step - loss: 0.7711 - accuracy: 0.4167 - val_loss: 0.7685 - val_accuracy: 0.4427
Epoch 2/200
18/18 [==============================] - 0s 4ms/step - loss: 0.7405 - accuracy: 0.4497 - val_loss: 0.7423 - val_accuracy: 0.4583
Epoch 3/200
18/18 [==============================] - 0s 4ms/step - loss: 0.7159 - accuracy: 0.4948 - val_loss: 0.7210 - val_accuracy: 0.4896
Epoch 4/200
18/18 [==============================] - 0s 4ms/step - loss: 0.6956 - accuracy: 0.5469 - val_loss: 0.7030 - val_accuracy: 0.5938
Epoch 5/200
18/18 [==============================] - 0s 4ms/step - loss: 0.6785 - accuracy: 0.5990 - val_loss: 0.6876 - val_accuracy: 0.6198
Epoch 6/200
18/18 [==============================] - 0s 3ms/step - loss: 0.6638 - accuracy: 0.6424 - val_loss: 0.6742 - val_accuracy: 0.6615
Epoch 7/200
18/18 [==============================] - 0s 4ms/step - loss: 0.6511 - accuracy: 0.6858 - val_loss: 0.6623 - val_accuracy: 0.6875
Epoch 8/200
18/18 [==============================] - 0s 4ms/step - loss: 0.6403 - accuracy: 0.7118 - val_loss: 0.6519 - val_accuracy: 0.6875
Epoch 9/200
18/18 [==============================] - 0s 4ms/step - loss: 0.6305 - accuracy: 0.7222 - val_loss: 0.6423 - val_accuracy: 0.6823
Epoch 10/200
18/18 [==============================] - 0s 4ms/step - loss: 0.6216 - accuracy: 0.7257 - val_loss: 0.6336 - val_accuracy: 0.7031
Epoch 11/200
18/18 [==============================] - 0s 4ms/step - loss: 0.6136 - accuracy: 0.7344 - val_loss: 0.6256 - val_accuracy: 0.7188
Epoch 12/200
18/18 [==============================] - 0s 4ms/step - loss: 0.6062 - accuracy: 0.7448 - val_loss: 0.6182 - val_accuracy: 0.7292
Epoch 13/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5995 - accuracy: 0.7535 - val_loss: 0.6114 - val_accuracy: 0.7396
Epoch 14/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5932 - accuracy: 0.7483 - val_loss: 0.6051 - val_accuracy: 0.7292
Epoch 15/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5873 - accuracy: 0.7535 - val_loss: 0.5993 - val_accuracy: 0.7240
```

```
Epoch 16/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5818 - accuracy: 0.7517 - val_loss: 0.5938 - val_accuracy: 0.7292
Epoch 17/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5766 - accuracy: 0.7517 - val_loss: 0.5887 - val_accuracy: 0.7292
Epoch 18/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5717 - accuracy: 0.7535 - val_loss: 0.5839 - val_accuracy: 0.7292
Epoch 19/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5671 - accuracy: 0.7535 - val_loss: 0.5793 - val_accuracy: 0.7292
Epoch 20/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5627 - accuracy: 0.7535 - val_loss: 0.5751 - val_accuracy: 0.7292
Epoch 21/200
18/18 [==============================] - 0s 3ms/step - loss: 0.5586 - accuracy: 0.7517 - val_loss: 0.5711 - val_accuracy: 0.7292
Epoch 22/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5547 - accuracy: 0.7535 - val_loss: 0.5674 - val_accuracy: 0.7292
Epoch 23/200
18/18 [==============================] - 0s 3ms/step - loss: 0.5511 - accuracy: 0.7535 - val_loss: 0.5638 - val_accuracy: 0.7344
Epoch 24/200
18/18 [==============================] - 0s 3ms/step - loss: 0.5476 - accuracy: 0.7535 - val_loss: 0.5604 - val_accuracy: 0.7344
Epoch 25/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5442 - accuracy: 0.7535 - val_loss: 0.5572 - val_accuracy: 0.7344
Epoch 26/200
18/18 [==============================] - 0s 3ms/step - loss: 0.5412 - accuracy: 0.7552 - val_loss: 0.5541 - val_accuracy: 0.7396
Epoch 27/200
18/18 [==============================] - 0s 4ms/step - loss: 0.5381 - accuracy: 0.7535 - val_loss: 0.5512 - val_accuracy: 0.7396
Epoch 28/200
```

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

```
y_pred_class_nn_1[:10]
```

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

```
y_pred_prob_nn_1[:10]
```

```
array([[0.57572615],
       [0.66893107],
       [0.2787839 ],
       [0.15336382],
       [0.21870871],
       [0.52485234],
       [0.02949159],
       [0.19661158],
       [0.9426653 ],
       [0.11621345]], dtype=float32)
```

Create the plot_roc function

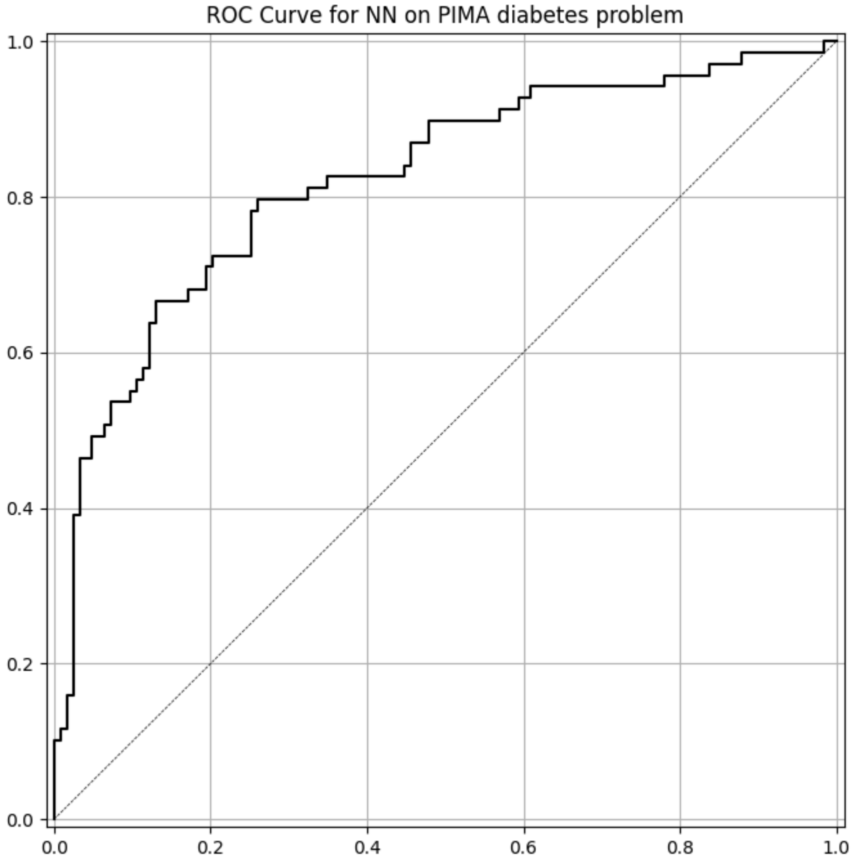
```
def plot_roc(y_test, y_pred, model_name):
    fpr, tpr, thr = roc_curve(y_test, y_pred)
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.plot(fpr, tpr, 'k-')
    ax.plot([0, 1], [0, 1], 'k--', linewidth=.5)  # roc curve for random model
    ax.grid(True)
    ax.set(title='ROC Curve for {} on PIMA 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 0.822
```



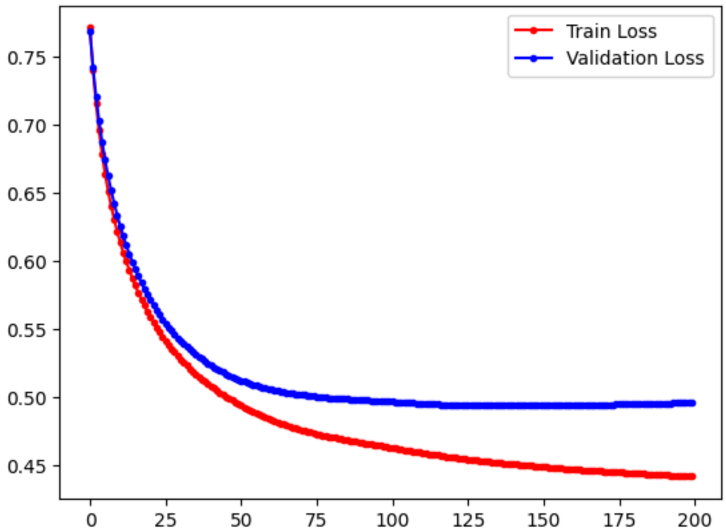
Plot the training loss and the validation loss over the different epochs and see how it looks

```
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 0x7c3fc0c16410>
```



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

- The result of the train and validation loss are having big gaps by the end of the graph, this means that the data is overfitting due to noises and outliers that may impact the dataset.

Supplementary Activity

```
# Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
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
from keras.models  import Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.optimizers import Adam, SGD, RMSprop
%matplotlib inline
```

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

```
diabetes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   times_pregnant          768 non-null    int64
 1   glucose_tolerance_test  768 non-null    int64
 2   blood_pressure          768 non-null    int64
 3   skin_thickness          768 non-null    int64
 4   insulin                 768 non-null    int64
 5   bmi                     768 non-null    float64
 6   pedigree_function       768 non-null    float64
 7   age                     768 non-null    int64
 8   has_diabetes            768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

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

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=11111)
```

```
normalize = StandardScaler()
XTrainNorm = normalize.fit_transform(X_train)
XTestNorm = normalize.transform(X_test)
```

```
#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(1, activation="sigmoid")
])
```

```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_5 (Dense)             (None, 6)                 54

 dense_6 (Dense)             (None, 1)                 7

=================================================================
Total params: 61 (244.00 Byte)
Trainable params: 61 (244.00 Byte)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-8006da3ef532> in <cell line: 2>()
      1 #Use a learning rate of .003 and train for 1500 epochs
----> 2 model.compile(SGD(lr = .003), "binary_crossentropy", metrics=["accuracy"])
      3 ModelSupp = model.fit(XTrainNorm, y_train, validation_data=(XTestNorm,
y_test), epochs=1500)

NameError: name 'model' is not defined
```

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

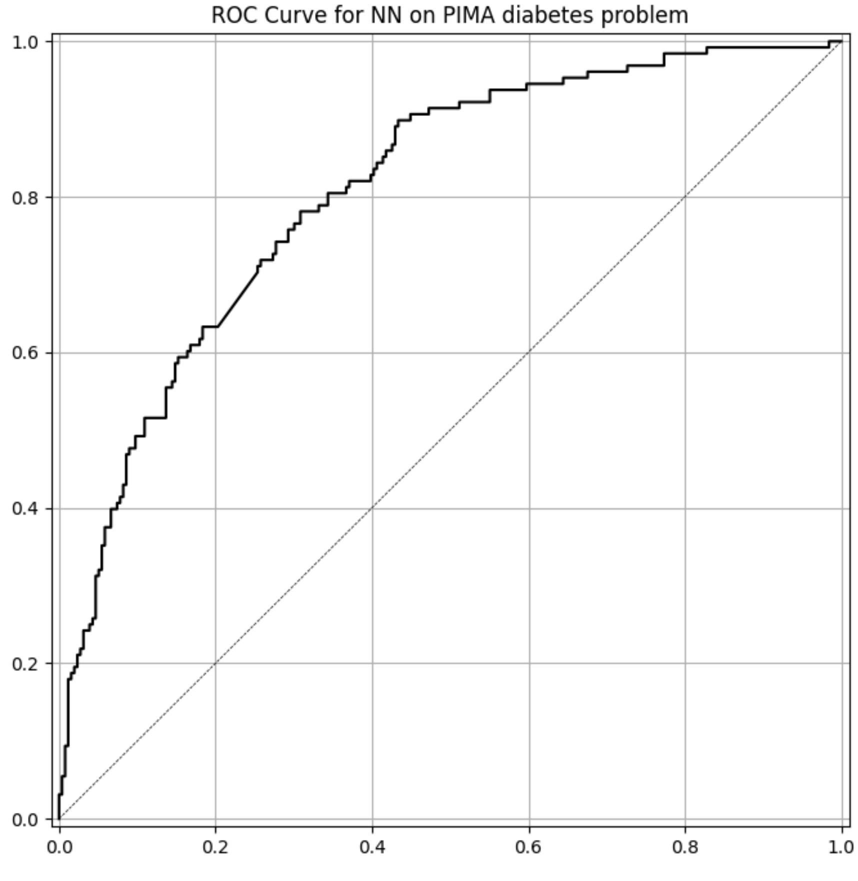
```
12/12 [==============================] - 0s 2ms/step
12/12 [==============================] - 0s 2ms/step
```

```
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])
```

```
#Plot the roc curve for the predictions
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 0.667
roc-auc is 0.809
```

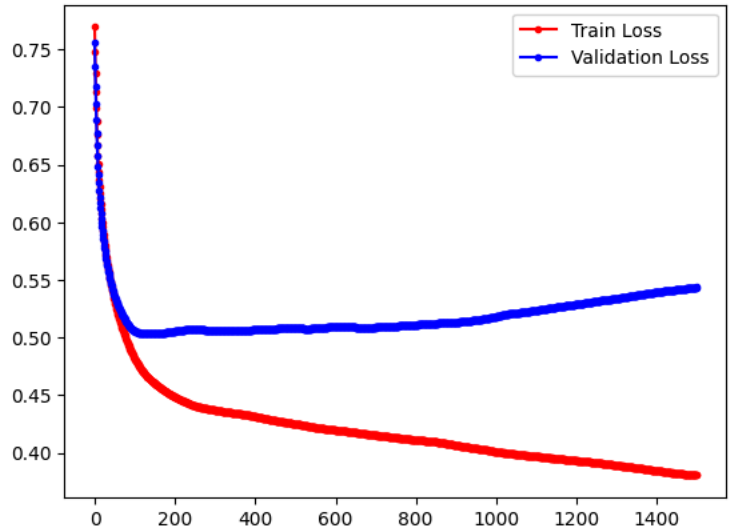


```
ModelSupp.history.keys()
```

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

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

```
<matplotlib.legend.Legend at 0x7c3fc0696cb0>
```



Remarks

In this graph, it has become overfitting but this time the dataset's train loss and validation loss create a huge gap to its graph by the end of it. The Accuracy of this data is no more than 0.677 and roc is much higher, which indicates that the data get the noise and and did not perform well.

Conclusion

In this activity, i work on demonstrating the training of neural networks using keras. I been able to train neural networks by building models using keras library, this difficult to train models you to some adjusting values such as weights learning rates and other codes, but are you able to adapt and understand training neural network in the given time but in some circumstances the models needs more time to able the data be more accurate.