NEURAL NETWORKS

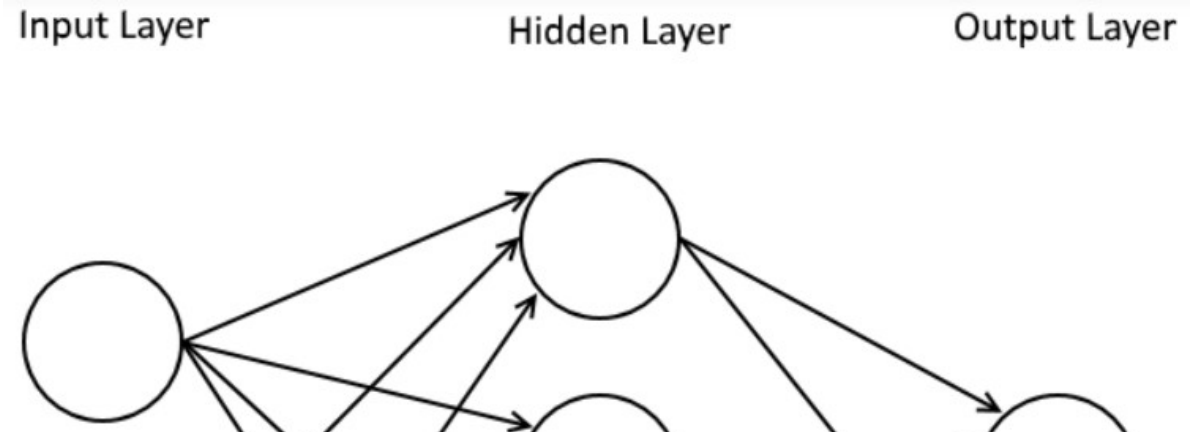
Overview

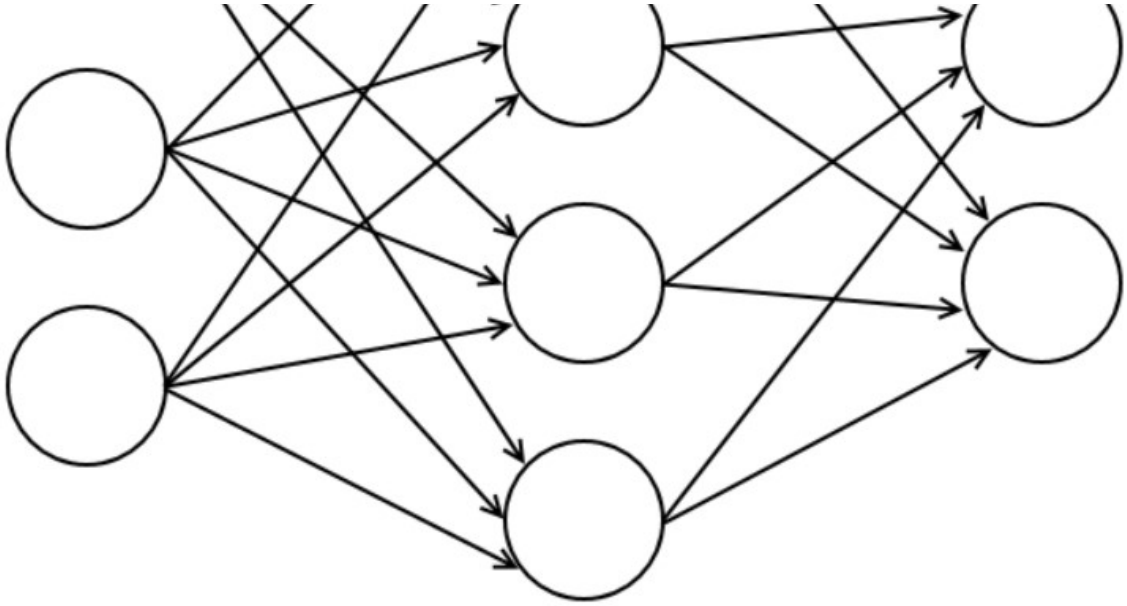
Although the Perceptron may seem like a good way to make classifications, it is a linear classifier (which, roughly, means it can only draw straight lines to divide spaces) and therefore it can be stumped by more complex problems. To solve this issue we can extend Perceptron by employing multiple layers of its functionality. The construct we are left with is called a Neural Network, or a Multi-Layer Perceptron, and it is a non-linear classifier. It achieves that by combining the results of linear functions on each layer of the network.

Similar to the Perceptron, this network also has an input and output layer; however, it can also have a number of hidden layers. These hidden layers are responsible for the non-linearity of the network. The layers are comprised of nodes. Each node in a layer (excluding the input one), holds some values, called *weights*, and takes as input the output values of the previous layer. The node then calculates the dot product of its inputs and its weights and then activates it with an *activation function* (e.g. sigmoid activation function). Its output is then fed to the nodes of the next layer. Note that sometimes the output layer does not use an activation function, or uses a different one from the rest of the network. The process of passing the outputs down the layer is called *feed-forward*.

After the input values are fed-forward into the network, the resulting output can be used for classification. The problem at hand now is how to train the network (i.e. adjust the weights in the nodes). To accomplish that we utilize the *Backpropagation* algorithm. In short, it does the opposite of what we were doing up to this point. Instead of feeding the input forward, it will track the error backwards. So, after we make a classification, we check whether it is correct or not, and how far off we were. We then take this error and propagate it backwards in the network, adjusting the weights of the nodes accordingly. We will run the algorithm on the given input/dataset for a fixed amount of time, or until we are satisfied with the results. The number of times we will iterate over the dataset is called *epochs*. In a later section we take a detailed look at how this algorithm works.

NOTE: Sometimes we add another node to the input of each layer, called *bias*. This is a constant value that will be fed to the next layer, usually set to 1. The bias generally helps us "shift" the computed function to the left or right.





In [31]:
```
# Import required libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import sklearn
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import train_test_split
```

Let us now read the dataset as a pandas dataframe. We will also print the shape of the dataset and a transposed summary of the variables present in the dataset

In [32]:
```
df = pd.read_csv('diabetes.csv')
print(df.shape)
```

```
(768, 9)
```

Out[32]:

	count	mean	std	min	25%	50%	75%
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000

Let us now split the feature attributes and the class attribute as we have been doing in all the

supervised learning schemes implemented so far. Once the features have been isolated they are also normalized by simply dividing them by the maximum values of each of the attributes.

In [33]:
```
target_column = ['Outcome']
predictors = list(set(list(df.columns))-set(target_column))
df[predictors] = df[predictors]/df[predictors].max()
df.describe().transpose()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	0.352941	0.743719	0.590164	0.353535	0.000000	0.500745	0.259091	0.617284	1
1	0.058824	0.427136	0.540984	0.292929	0.000000	0.396423	0.145041	0.382716	0
2	0.470588	0.919598	0.524590	0.000000	0.000000	0.347243	0.277686	0.395062	1
3	0.058824	0.447236	0.540984	0.232323	0.111111	0.418778	0.069008	0.259259	0
4	0.000000	0.688442	0.327869	0.353535	0.198582	0.642325	0.945455	0.407407	1
..	...	...	...	...	...	...	...	...	...
763	0.588235	0.507538	0.622951	0.484848	0.212766	0.490313	0.070661	0.777778	0
764	0.117647	0.613065	0.573770	0.272727	0.000000	0.548435	0.140496	0.333333	0
765	0.294118	0.608040	0.590164	0.232323	0.132388	0.390462	0.101240	0.370370	0
766	0.058824	0.633166	0.491803	0.000000	0.000000	0.448584	0.144215	0.580247	1
767	0.058824	0.467337	0.573770	0.313131	0.000000	0.453055	0.130165	0.283951	0

[768 rows x 9 columns]

Next we create a training and testing split.

In [34]:
```
X = df[predictors].values
y = df[target_column].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, rand
print(X_train.shape)
```

```
(537, 8)
(231, 8)
```

Next, lets train a neural network for the training dataset. We will be learning a Multi-Layer Perceptron Classifier for this dataset.

In [37]:

```
from sklearn.neural_network import MLPClassifier

mlp = MLPClassifier(hidden_layer_sizes=(8,8,8), activation='logistic', solver=
mlp.fit(X_train,y_train.ravel())

predict_train = mlp.predict(X_train)
```

The second line instantiates the model with the 'hidden_layer_sizes' argument set to three layers, which has the same number of neurons as the count of features in the dataset. We will also select 'logistic' (Sigmoid function) as the activation function and 'adam' as the solver for weight optimization. To learn more about 'relu' and 'adam', please refer to the SKlearn manuals, one possible link is provided [https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html (https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html)](https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html).

The third line of code fits the model to the training data. The line also transforms the target column into a ID array for the model to work with. The fourth and fifth lines use the trained model to generate predictions on the training and test dataset, respectively.

In [38]:

```
from sklearn.metrics import classification_report,confusion_matrix
print(confusion_matrix(y_train,predict_train))
```

```
[[358   0]
 [179   0]]
              precision    recall  f1-score   support

           0       0.67      1.00      0.80       358
           1       0.00      0.00      0.00       179

    accuracy                           0.67       537
   macro avg       0.33      0.50      0.40       537
weighted avg       0.44      0.67      0.53       537


C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

The above output shows the performance of the model on training data. The accuracy and the F1 score is around 0.78 and 0.77, respectively. Ideally, the perfect model will have the value of

1 for both these metrics, but that is next to impossible in real-world scenarios.

The next step is to evaluate the performance of the model on the test data that is done with the

In [39]:
```python
print(confusion_matrix(y_test,predict_test))
```

```
[[142   0]
 [ 89   0]]
              precision    recall  f1-score   support

           0       0.61      1.00      0.76       142
           1       0.00      0.00      0.00        89

    accuracy                           0.61       231
   macro avg       0.31      0.50      0.38       231
weighted avg       0.38      0.61      0.47       231


C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\user\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:
1318: UndefinedMetricWarning: Precision and F-score are ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division` parameter
to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Neural Network for Prediction

The great thing about Neural Networks are that they are not just a good classification model but they are equally good when it comes to prediction problems. Let us explore a Neural Networks capability in a prediction scenario against the SalaryAge dataset we have already explored in the regression notebook

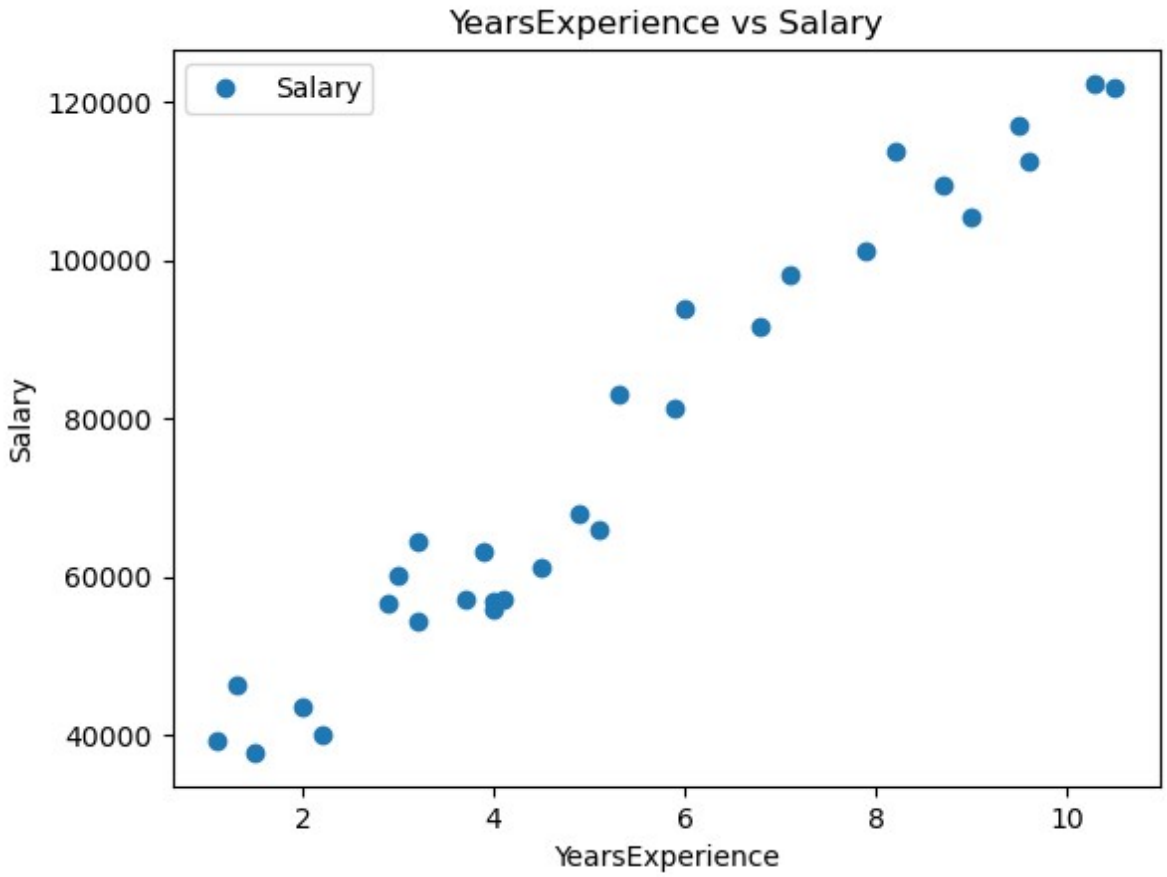
In [40]:
```python
import numpy as np
import matplotlib.pyplot as plot
import pandas
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPRegressor
from sklearn import metrics
```

After importing the required libraries, we go ahead and import the dataset. Make sure that you have the dataset in the same folder as your notebook. Otherwise provide the complete location of your destination data file.

In [41]:
```
# Import the dataset
dataset = pandas.read_excel('SalaryAge.xlsx')
x = dataset['YearsExperience'].values.reshape(-1,1)
y = dataset['Salary'].values.reshape(-1,1)
```

let's plot our data points on a 2-D graph to look at our dataset and see if we can manually find any relationship between the data using the below script:

In [42]:
```
dataset.plot(x = 'YearsExperience', y ='Salary', style= 'o')
plot.title('YearsExperience vs Salary')
plot.xlabel('YearsExperience')
plot.ylabel('Salary')
```



Training and Testing Split

In [53]:

In [54]:
```
mlp_R = MLPRegressor(hidden_layer_sizes=(18,18,18), activation='relu', solver=
mlp_R.fit(xTrain,yTrain.ravel())

predict_train = mlp_R.predict(xTrain)
```

```
C:\Users\user\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_
perceptron.py:692: ConvergenceWarning: Stochastic Optimizer: Maximum iteratio
ns (5000) reached and the optimization hasn't converged yet.
  warnings.warn(
```

In [55]:

```
[74425.23282941 92137.12614442 60649.31580662 81313.1913408
 66553.28024496 89185.14392525]
```

In [56]:
```
df = pandas.DataFrame({'Actual': yTest.flatten(), 'Predicted': predict_test.fl
```

Out[56]:

	Actual	Predicted
0	83088	74425.232829
1	98273	92137.126144
2	63218	60649.315807
3	93940	81313.191341
4	61111	66553.280245
5	91738	89185.143925

In [57]:
```
print('Mean Absolute Error:', metrics.mean_absolute_error(yTest, predict_test)
print('Mean Squared Error:', metrics.mean_squared_error(yTest, predict_test))
```

```
Mean Absolute Error: 6331.5450330746135
Mean Squared Error: 52477067.804620855
Root Mean Squared Error: 7244.105728426446
```