Artificial Neural Network Algorithm:

ANN is a Black-Box type method in which we have no idea what exactly information is being calculated or use so due to which its not much explanable due to its complexity.How human brian works , ANN try to mimic it hence name is Artificial.

Advantage : Ability to learn by its own depending upon the function or on past Data. Ability to generalize the output if it has been taught to deal with it. High Adaptivity as it easily retrained to the changing environmental conditions.

Neuron : Its carrier for the input information.

ANN is made up of 3 layers.

INPUT LAYER : It is From Data.

HIDDEN LAYER : It can be one or more depending upon the level of information.

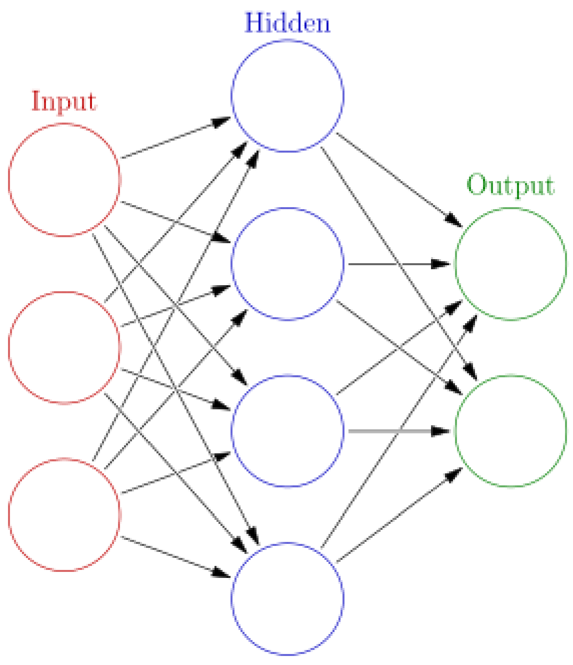
OUTPUT LAYER : Gives the Ouput.

Synaptic weights : its an numerical value which is randomly assigned by the ANN algorithm.

How Algorithm works:

From Data input layer takes the input and pass it to the Hidden layer and then Hidden layer does the calculation and get the output.

Example :



Step 1 : There are three input layers x1 = 5,x2 = 8 and x3 = 17. Step 2 : There are 4 Hidden layers N1,N2,N3 and N4. Step 3 : There are two output layers Y1 and Y2.

What algorithm do at the beginning itself it randomly selects the synaptic weights for every input to connect to the Hidden layers for example for x1 will carry weight (w1),x2 = w2 ,x3 = w3. Now all the input layers will multiply with all the hidden layers which will be like x1w1 *N1, x1w2* N2, x1w3 *N3, x1w4* N4. x2w5 *N1, x2w6* N1, x2w7 *N1, x2w8* N1 and so on for all the input weights. Then Hidden layer will calculate all the information and then pass it to the outputs and then combing both the outputs (as in the below image there are 2 outputs) one final output will come and that will be the Predicted value.

For example : if our original output was 80 and our ANN algorithm gives us the value 62 then ANN again run with the changed synaptic weights and then again calculate the output and if in the 2nd iteration the output will be around 64 then again ANN will randomly selects the weights which will be increase from last iteration. This iteration will be stop until and unless ANN algorithm will not reach one optimal value in this case till above 75 with no error, as it canot be 80 which will be overfitted situation.

As objective of ANN algorithm is to bring down or close to the output or gives predicted value close to the expected value.

Loss Function = y - ybar^ where y is output and ybar^ is predicted output, sometimes values could be in negative hence we will be squaring it for avoiding negative output so we will be taking modulus of /y-ybar ^/.

How to Bring down the Loss Function:

we dont know how many weights needs to increase or decrease so that error value/loss function will bring down to the expected output, so we will be using the DIFFERENTIATION method in which it gives the rate of which function value is MOVING and DIRECTION where its moving. So depending upon the DIRECTION if it is positive value we know that it is increasing and if it is negative value it is decreasing.

For above example we know our expected output is 80 and we have got the predicted output is 62 so here we have to increase the weights so that our Predicted value will bring down close to the 80.

Requirement of Differentiation is Values should be differentiative from each other,so new loss function will become.

New Loss FUNCTION.

(y-ybar)^2 we will be squaring the residual means (y-ybar)^2. Objective of Squaring is if the error made by model is large then punishment will also be large and if error made by model is low then punishment is also low , example : if (y-ybar^) = 5 then squaring it given 25 and if (y-ybar^) = 3 then squaring it gives 9 so here error is less.

Optimazation Technique (How this Loss function will work): Stochastic Gradient Discent.

Bringing maximum error point to minimum error point is called as SGD. So once we know the Differentiation the direction in which the wights are moving then by using SGD it automatically adjust the weights itself, like if predicted value is far from the expected value then its in POSITIVE direction so in this case SGD will decrease the weights to bring minimize the error and if predicted value is less than expected value then its in NEGATIVE direction and then SGD will increase the weights automatically to minimize the error.

Positive value then Decrease the weights.(expected 80 and predicted is 116 then weights will decrease).

Negative value then Increase the weights.(expected 80 and predicted is 62 then weights will increase).

Back Propogation : It updates the weights from Output to input.

As our optimal goal is to bring down the error loss for that model will run multiple times which means it will do a complex mathematical calculation which take much more time. We have to ensure that model will atleast reach to near by expected value by properly selecting or playing with the features in different algorithms but here in ANN algorithm we dont have that benefits so hence we have to give some STOPPING techniques which is ### THRESHOLD.

THRESHOLD: It is for Differential value which comes from the loss function.If Threshold value is LOW then Model will run MORE number of times and if Threshold value is MORE then Model will run LESS number of times.

example: T = 0.1 , 0.01, 0.001 which means 0.1 = Model will run with the High speed and 0.001 Model will run with less speed but gives the More accuracy. If value is less then MORE ACCURACY but execution of model will take more time. If value is more then Accuracy will not be same but model will take less time for execution.

Biased : Sometimes if input is 0 then ANN algorithm assign the Hidden layer with value of 1 so that output should get the Positive output other than 0.

Activation Function:

If my output is Categorical then we will be activating the activation function,we set a threshold of 0 and 1 , all Continous values will lie into 0 and all categorical value will lie into 1. Types of Activation Functions are : UNIT , SIGMOID , RELU (Rectifier linear unit) and TANH etc, the objective of all the functions are to convert the continous output into the Categorical output or the Probability to classify 0's and 1's.

Summary :

Layers : Input , Hidden and Output. Check the Error loss. Differentiate the Error loss to know the direction of moving weights. Direction is positive then decrease weights , Direction is negative then increase weights (Stochastic Gradient Discent). Threshold : minimum value gives more accuracy but maximum time to execute and maximum value gives minimum accuracy but less time. Model Evaluation : Confusion matrix and Classification Report. Visualisation: ROC AUC curve.

In [ ]:

```

```

In [ ]:

```

```

Lets try practically with a banking dataset where the objective is to predict whether the customers will take a loan or not via marketing campaign.

In [91]:

```
# importing the libraries.

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.model_selection import train_test_split
```

In [2]:

```
bank = pd.read_csv (r'E:\Great Learning\Data Mining-Unsupervised\From Videos\Artificial
Neural Networks\Bank Dataset.csv')
bank.head()
```

Out[2]:

	ID	Age	Experience	Income	ZIP_Code	Family_members	CCAvg	Education	Mortgage
0	1	25	1	49	91107	4	1.6	1	0
1	2	45	19	34	90089	3	1.5	1	0
2	3	39	15	11	94720	1	1.0	1	0
3	4	35	9	100	94112	1	2.7	2	0
4	5	35	8	45	91330	4	1.0	2	0

In [4]:

```
# we will be dropping the inconvinient coloumns such as ID and Zip code.

bank = bank.drop (['ID' , 'ZIP_Code'],axis=1)
```

In [6]:

```
bank.head() # to confirm the removing of 2 coloumns.
```

Out[6]:

	Age	Experience	Income	Family_members	CCAvg	Education	Mortgage	Personal_Loan
0	25	1	49	4	1.6	1	0	0
1	45	19	34	3	1.5	1	0	0
2	39	15	11	1	1.0	1	0	0
3	35	9	100	1	2.7	2	0	0
4	35	8	45	4	1.0	2	0	0

In [8]:

```
x = bank.drop (['Personal_Loan'] , axis=1) # dropping the dependent variable foe Traini
ng data.
y = bank.pop ('Personal_Loan') # adding only dependent variable for testing data.
```

In [9]:

```
x_train,x_test,y_train,y_test = train_test_split (x,y, test_size = .30 , random_state =
8)
```

In [10]:

```
### For Artificial Neural Network Algorithm , Scaling is Mandatory.

from sklearn.preprocessing import StandardScaler
```

In [11]:

```
sc = StandardScaler()
```

In [12]:

```
x_train = sc.fit_transform (x_train)
```

In [13]:

```
x_train # all are in scaled now.
```

Out[13]:

```
array([[-0.20916275, -0.10485419,  0.01982845, ..., -0.26138301,
        -1.21513888,  1.52554583],
       [ 0.22642642,  0.33062291, -1.39389185, ..., -0.26138301,
        -1.21513888,  1.52554583],
       [ 0.31354426,  0.33062291, -0.65440739, ..., -0.26138301,
         0.8229512 , -0.65550309],
       ...,
       [ 1.01048694,  1.02738628, -1.04589916, ..., -0.26138301,
         0.8229512 ,  1.52554583],
       [-0.55763409, -0.62742672,  1.43354874, ..., -0.26138301,
         0.8229512 , -0.65550309],
       [ 1.09760477,  1.02738628, -0.98065053, ..., -0.26138301,
        -1.21513888, -0.65550309]])
```

In [14]:

```
x_test = sc.transform (x_test) # we will not be doing fit here just to ensure that data
is uniform.
```

In [15]:

```
x_test
```

Out[15]:

```
array([[ 0.05219075, -0.01775877, -0.19766698, ..., -0.26138301,
        -1.21513888, -0.65550309],
       [ 1.35895828,  1.46286338,  1.84679006, ...,  3.82580335,
         0.8229512 ,  1.52554583],
       [ 0.9233691 ,  1.02738628, -0.48041104, ..., -0.26138301,
        -1.21513888, -0.65550309],
       ...,
       [ 1.35895828,  1.46286338, -1.11114779, ..., -0.26138301,
         0.8229512 , -0.65550309],
       [-1.16745894, -1.06290382,  1.19430377, ...,  3.82580335,
         0.8229512 ,  1.52554583],
       [-1.42881244, -1.41128551, -1.11114779, ..., -0.26138301,
        -1.21513888, -0.65550309]])
```

In [29]:

```
# now we will be building the MLP classifier function.

# hidden_layer_sizes = how many hidden layer we want for our input,each input will pass
the information with some numeric value (which
# will be randomly selected by alogorithm and then) and its multiply it with the Hidden
layer.
# we can keep single hidden layer 10 or we can keep multiple hidden layers in the form
 of tuples i.e (10,10,10).
# it should be 10 times of variables in Dataset.

# max_iter = how many times we want our algorithm run to bring down the error value dow
n.can be 500 always.
# solver = sgd = which means based on differentitaion (either negative or positive), it
will take decision to bring down the weights.
# (positive value weights will be decrease and for negative value weights will get incr
ease).
# verbose = True = to print the iterations.
# tol = threshold = minimum the value max accuracy but less speed and maximum the value
then minimum accuracy but high speed.
# random_state = 8 = to keep the alogorithm fix.

clf = MLPClassifier (hidden_layer_sizes= 100 , max_iter= 5000 ,
                     solver='sgd' , verbose=True , random_state=8, tol=0.01)
```

In [30]:

```
# now we will fit the model on our training data.

clf.fit(x_train , y_train)
```

```
Iteration 1, loss = 0.64466049
Iteration 2, loss = 0.55365749
Iteration 3, loss = 0.47761554
Iteration 4, loss = 0.42589460
Iteration 5, loss = 0.38983001
Iteration 6, loss = 0.36304098
Iteration 7, loss = 0.34243128
Iteration 8, loss = 0.32552582
Iteration 9, loss = 0.31154620
Iteration 10, loss = 0.29950887
Iteration 11, loss = 0.28896967
Iteration 12, loss = 0.27959266
Iteration 13, loss = 0.27106860
Iteration 14, loss = 0.26351006
Iteration 15, loss = 0.25649199
Iteration 16, loss = 0.25001643
Iteration 17, loss = 0.24407080
Iteration 18, loss = 0.23853060
Iteration 19, loss = 0.23340787
Iteration 20, loss = 0.22865501
Iteration 21, loss = 0.22418256
Iteration 22, loss = 0.22000341
Training loss did not improve more than tol=0.010000 for 10 consecutive ep
ochs. Stopping.
```

Out[30]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=
0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=100, learning_rate='constant',
              learning_rate_init=0.001, max_iter=5000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=8, shuffle=True, solver='sgd', tol=0.01,
              validation_fraction=0.1, verbose=True, warm_start=False)
```

In [31]:

```
# As we can see that Model has converged in 22 iterations only when tolerance level is
 very smal which is 0.01 (speed is fast but accuracy is low)
```

In [ ]:

```

```

In [24]:

```
y_pred = clf.predict (x_test)
```

In [32]:

```
# we can check the GRIDSEARCH cv to check how model choose the best feature and its val
ue.
```

In [33]:

```
from sklearn.model_selection import GridSearchCV
```

In [56]:

```
param_grid = {'hidden_layer_sizes': [10,100],
             'max_iter': [1000,3000],
             'solver': ['sgd'],
             'verbose': ['True'],
             'tol': [0.001,0.000001],
             }
```

In [57]:

```
clf=MLPClassifier()
```

In [58]:

```
grid = GridSearchCV (param_grid = param_grid , cv= 3,estimator=clf)
```

In [60]:

```
grid.best_params_
```

Out[60]:

```
{'hidden_layer_sizes': 10,
 'max_iter': 3000,
 'solver': 'sgd',
 'tol': 1e-06,
 'verbose': 'True'}
```

In [61]:

```
grid.best_estimator_
```

Out[61]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=
0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=10, learning_rate='constant',
              learning_rate_init=0.001, max_iter=3000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='sgd', tol=1e-06,
              validation_fraction=0.1, verbose='True', warm_start=False)
```

In [62]:

```
# So out of this number od parameters we have found the best and also out of number of
 models we have found the best estimator.
```

In [63]:

```
best_model = grid.best_estimator_
```

In [69]:

```
ytest_pred = best_model.predict (x_test)
ytrain_pred = best_model.predict (x_train)
```

In [70]:

```
print(classification_report (y_test,ytest_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1348
           1       0.91      0.86      0.89       147

    accuracy                           0.98      1495
   macro avg       0.95      0.93      0.94      1495
weighted avg       0.98      0.98      0.98      1495
```

In [71]:

```
print(classification_report(y_train,ytrain_pred))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99      3156
           1       0.95      0.87      0.91       331

    accuracy                           0.98      3487
   macro avg       0.97      0.93      0.95      3487
weighted avg       0.98      0.98      0.98      3487
```

In [72]:

```
confusion_matrix (y_train,ytrain_pred)
```

Out[72]:

```
array([[3141,   15],
       [  42,  289]], dtype=int64)
```

In [73]:

```
confusion_matrix (y_test,ytest_pred)
```

Out[73]:

```
array([[1336,   12],
       [  20,  127]], dtype=int64)
```

In [78]:

```
from sklearn.metrics import roc_auc_score
```

In [86]:

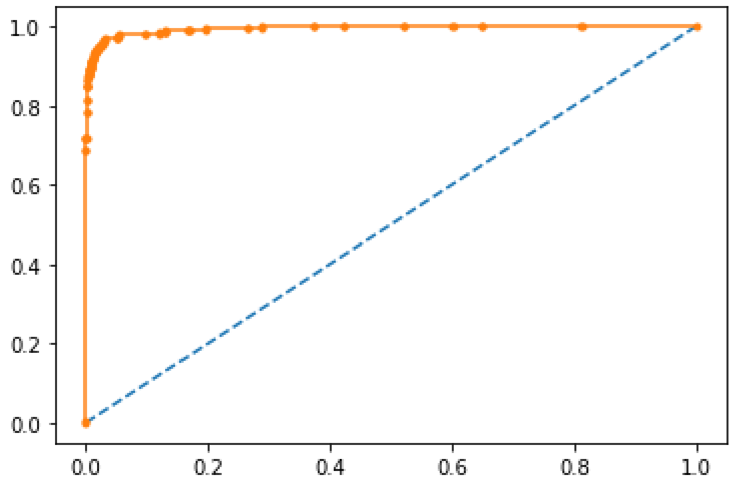
```
# predicting probabilities for Train Data.
probs = best_model.predict_proba (x_train)
probs = probs [:,1] # we will need only 1s.
```

In [88]:

```
auc = roc_auc_score (y_train , probs) # For calculating AUC score
print ('AUC %.3f' % auc)

from sklearn.metrics import roc_curve
fpr , tpr, thresholds = roc_curve (y_train , probs)
plt.plot ([0,1],[0,1], linestyle = '--')
plt.plot (fpr , tpr, marker= '.')
plt.show()
```

```
AUC 0.994
```



In [89]:

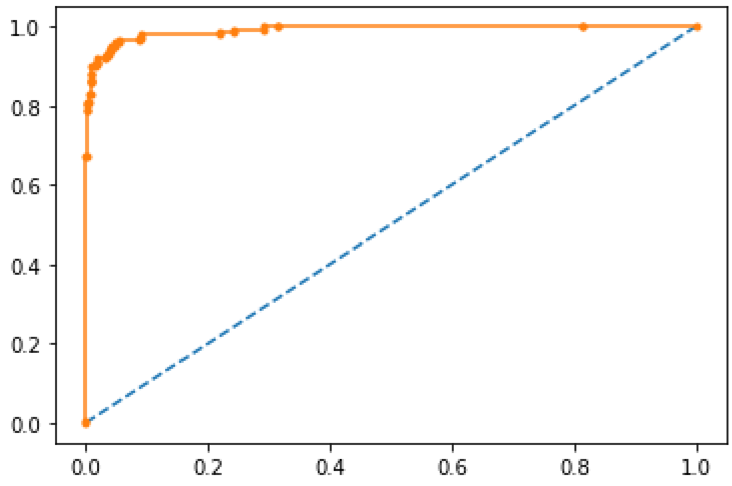
```
# predicting probabilities for Train Data.
probs = best_model.predict_proba (x_test)
probs = probs [:,1] # we will need only 1s.
```

In [90]:

```
auc = roc_auc_score (y_test , probs) # For calculating AUC score
print ('AUC %.3f' % auc)

from sklearn.metrics import roc_curve
fpr , tpr, thresholds = roc_curve (y_test , probs)
plt.plot ([0,1],[0,1], linestyle = '--')
plt.plot (fpr , tpr, marker= '.')
plt.show()
```

```
AUC 0.990
```



So model has given the Same score of 99 % for Training and Testing Data, but in general its an Overfitting need to add more Data.

In [ ]:

```

```