**NAME: RADHIKA GUPTA**

**REG. NO: 22MCA1119**

# ITA-6016 Machine Learning

## Digital Assignment –Lab-4

**SUBMITTED TO: Dr Dominic Savio M**

# PERCEPTRON:

# CODE OF THE PROGRAM AND OUTPUT:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
import tensorflow as tf
import keras
import math
from keras.models import Sequential
from keras.layers import Dense
%matplotlib inline
```

In [2]:
```
df = pd.read_csv('D:\\vit notes\\MCA Second Semester\\MachineLearning\\train27303.csv')
df.head()
```

Out[2]:

| | timestamp | hourly_traffic_count |
|---|---|---|
| 0 | 2015-10-04 00:00:00 | 3 |
| 1 | 2015-10-04 00:05:00 | 16 |
| 2 | 2015-10-04 00:10:00 | 9 |
| 3 | 2015-10-04 00:15:00 | 12 |
| 4 | 2015-10-04 00:20:00 | 19 |

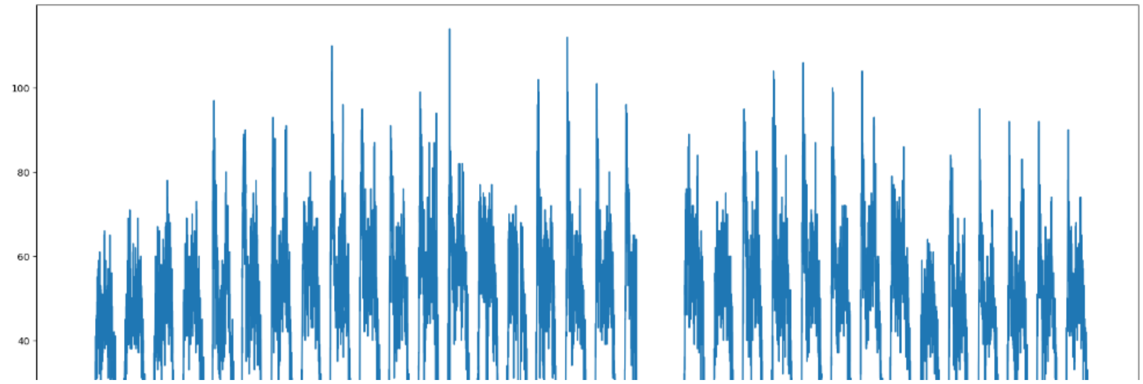
In [3]:
```
df1 = df.reset_index()['hourly_traffic_count']
df1.head()
```

Out[3]:
```
0     3
1    16
2     9
3    12
4    19
Name: hourly_traffic_count, dtype: int64
```

In [4]:
```
df1 = df1.iloc[:9792,]
df1.tail()
```

Out[4]:
```
9787    23
9788    25
9789    16
9790    18
9791    25
Name: hourly_traffic_count, dtype: int64
```

In [5]:
```
plt.figure(figsize=(20,10))
plt.plot(df1)
plt.show()
```



```
In [7]: def create_dataset(dataset, window=1):
            dataX, dataY= [], []
            for i in range(len(dataset)-window-1):
                a = dataset[i:(i+window),0]
                dataX.append(a)
                dataY.append(dataset[i+window,0])
            return np.array(dataX), np.array(dataY)
```

```
In [8]: scaler = MinMaxScaler(feature_range=(0,1))
        df1 = scaler.fit_transform(np.array(df1).reshape(-1,1))
```

```
In [9]: training_size = int(len(df1)*0.80)
        test_size = len(df1)-training_size
        train_data, test_data = df1[0:training_size,:], df1[training_size:len(df1),:1]
```

```
In [10]: window = 288
         X_train,y_train = create_dataset(train_data,window)
         X_test, y_test = create_dataset(test_data,window)
```

```
In [11]: model = Sequential()
         model.add(Dense(40, input_dim=window, activation='relu'))
         model.add(Dense(50, activation='relu'))
         model.add(Dense(40, activation='relu'))
         model.add(Dense(1))
```

```
In [12]: opt  = keras.optimizers.Adagrad(learning_rate = 0.05)
```

```
In [13]: model.compile(optimizer=opt ,loss='mean_squared_error')
```

In [14]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 40)                11560

 dense_1 (Dense)             (None, 50)                2050

 dense_2 (Dense)             (None, 40)                2040

 dense_3 (Dense)             (None, 1)                 41

=================================================================
Total params: 15,691
Trainable params: 15,691
Non-trainable params: 0
_________________________________________________________________
```

In [15]:
```
model.fit(X_train, y_train, epochs=100, batch_size=10, verbose=1)
```

```
Epoch 1/100
755/755 [==============================] - 4s 3ms/step - loss: 0.0096
Epoch 2/100
755/755 [==============================] - 3s 3ms/step - loss: 0.0064
Epoch 3/100
755/755 [==============================] - 3s 4ms/step - loss: 0.0058
Epoch 4/100
755/755 [==============================] - 3s 4ms/step - loss: 0.0055
Epoch 5/100
755/755 [==============================] - 3s 4ms/step - loss: 0.0054
```

```
In [16]: train_predict = model.predict(X_train)
         test_predict = model.predict(X_test)
```

```
236/236 [==============================] - 1s 3ms/step
53/53 [==============================] - 0s 3ms/step
```

```
In [17]: train_predict = scaler.inverse_transform(train_predict)
         test_predict = scaler.inverse_transform(test_predict)
         y_train = scaler.inverse_transform(y_train.reshape(-1, 1))
         y_test = scaler.inverse_transform(y_test.reshape(-1, 1))
```

```
In [18]: train_predict = train_predict.astype(int)
         test_predict = test_predict.astype(int)
         y_train = y_train.astype(int)
         y_test = y_test.astype(int)
```

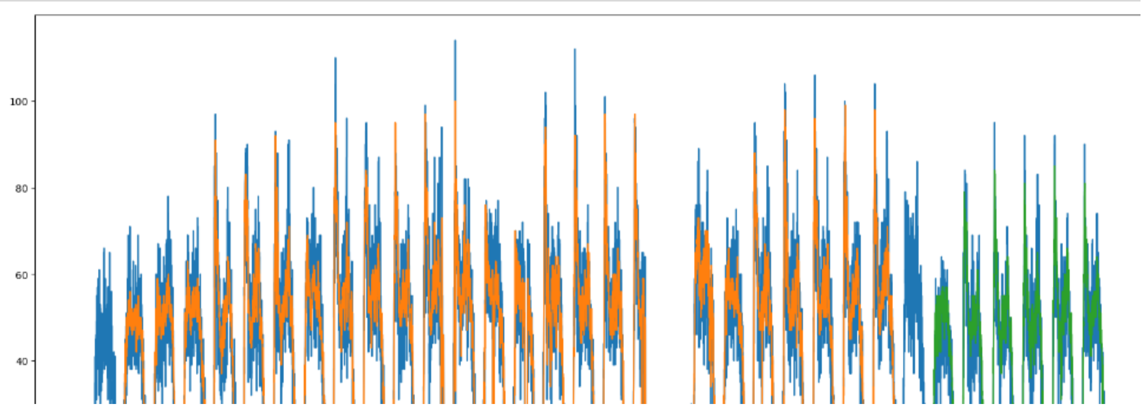
```
In [19]: from sklearn.metrics import mean_squared_error, mean_absolute_error,r2_score
         print('RMSE-train:',math.sqrt(mean_squared_error(y_train,train_predict)))
         print('MAE-train:',mean_absolute_error(y_train,train_predict))
         print('R_2-train:',r2_score(y_train,train_predict))
```

```
RMSE-train: 6.46107522840471
MAE-train: 4.761399787910922
R_2-train: 0.9264831054092415
```

```
In [20]: print('RMSE-test:',math.sqrt(mean_squared_error(y_test,test_predict)))
         print('MAE-test:',mean_absolute_error(y_test,test_predict))
         print('R_2-train:',r2_score(y_test,test_predict))
```

```
RMSE-test: 7.463042676283176
MAE-test: 5.743712574850299
R_2-train: 0.8614505151272462
```

```
In [21]: # shift train predictions for plotting
         trainPredictPlot = np.empty_like(df1)
         trainPredictPlot[:, :] = np.nan
         trainPredictPlot[window:len(train_predict)+window, :] = train_predict
         # shift test predictions for plotting
         testPredictPlot = np.empty_like(df1)
         testPredictPlot[:, :] = np.nan
         testPredictPlot[len(train_predict)+(window*2)+1:len(df1)-1, :] = test_predict
         # plot baseline and predictions
         plt.figure(figsize=(20,10))
         plt.plot(scaler.inverse_transform(df1))
         plt.plot(trainPredictPlot)
         plt.plot(testPredictPlot)
         plt.show()
```



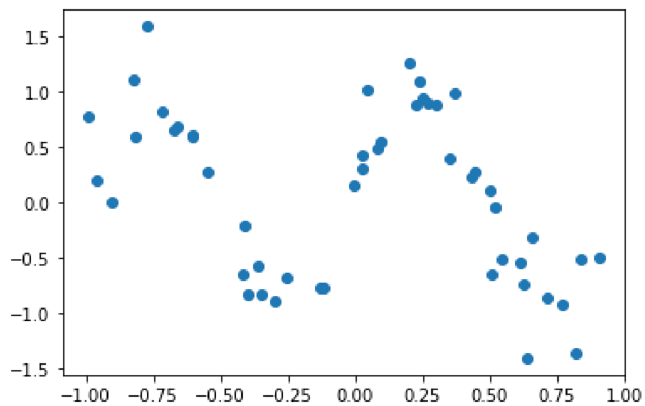
# BACKPROPOGATION:

# CODE OF THE PROGRAM AND OUTPUT:

## Backpropogation Algorithm

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
#synthetic data with noise
np.random.seed(10)

X = 2*np.random.rand(1, 50) - 1
T = np.sin(2*np.pi*X) + 0.3*np.random.randn(1, 50)
N = np.size(T,1)
plt.scatter(X, T)
plt.show()
```




In [3]:
```
# NN implementaion with feed-forward propagation and backprojection
def Neural_Network_Simple(LR,beta,max_ite,input_nodes,hidden_nodes,output_nodes):
```

In [3]:

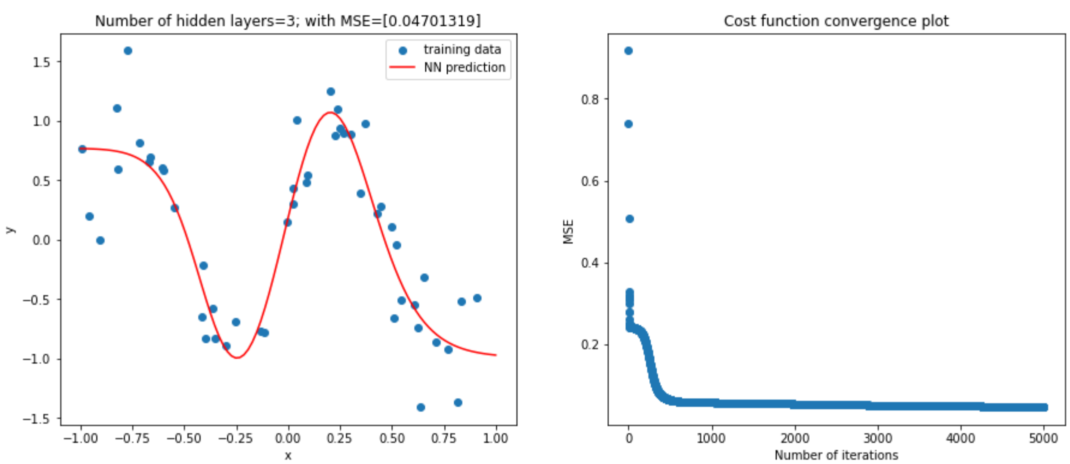
```python
# NN implementaion with feed-forward propagation and backprojection
def Neural_Network_Simple(LR,beta,max_ite,input_nodes,hidden_nodes,output_nodes):

    # weight initialization function
    W_1 = np.random.randn(hidden_nodes, input_nodes)
    W_2 = np.random.randn(output_nodes, hidden_nodes)
    B_1 = np.zeros((hidden_nodes, 1))
    B_2 = np.zeros((output_nodes, 1))

    # gradient descent with momentum
    Vdw_1 = np.random.randn(hidden_nodes, input_nodes)
    Vdw_2 = np.random.randn(output_nodes, hidden_nodes)
    Vdb_1 = np.zeros((hidden_nodes, 1))
    Vdb_2 = np.zeros((output_nodes, 1))

    # cost initialization
    Cost = np.zeros((max_ite,1))

    for i in range(max_ite):
        #feed-forward propagation
        A_1 = W_1.dot(X) + np.tile(B_1, (1, N))
        Z_1 = (np.exp(A_1) - np.exp(-A_1)) / (np.exp(A_1) + np.exp(-A_1))

        A_2 = W_2.dot(Z_1) + np.tile(B_2, (1, N))
        Z_2 = A_2

        #back propagation
        del_2 = Z_2 - T
        del_1 = W_2.T.dot(del_2) * (1 - Z_1 ** 2)

        #gradient
        dw_2 = del_2.dot(Z_1.T)
        dw_1 = del_1.dot(X.T)
```

```python
        dw_2 = del_2.dot(Z_1.T)
        dw_1 = del_1.dot(X.T)
        db_2 = np.sum(del_2, 1)
        db_1 = np.sum(del_1, 1).reshape(hidden_nodes,1)

        #GD with momentum
        Vdw_2 = beta * Vdw_2 + (1 - beta) * dw_2
        Vdw_1 = beta * Vdw_1 + (1 - beta) * dw_1
        Vdb_2 = beta * Vdb_2 + (1 - beta) * db_2
        Vdb_1 = beta * Vdb_1 + (1 - beta) * db_1

        #update weight and bias with batch GD
        W_2 = W_2 - LR * Vdw_2
        W_1 = W_1 - LR * Vdw_1
        B_2 = B_2 - LR * Vdb_2
        B_1 = B_1 - LR * Vdb_1

        Cost[i] = 0.5 * np.sum(del_2**2)/N
    return W_1,W_2,B_1,B_2,Cost
```

In [4]:
```python
# prediction with forward propagation
def forwardNN_reg(W_1,W_2,B_1,B_2,X):
    A_1 = W_1.dot(X) + np.tile(B_1, (1, 1))
    Z_1 = (np.exp(A_1) - np.exp(-A_1)) / (np.exp(A_1) + np.exp(-A_1))

    A_2 = W_2.dot(Z_1) + np.tile(B_2, (1, 1))
    pred = A_2
    return pred
```

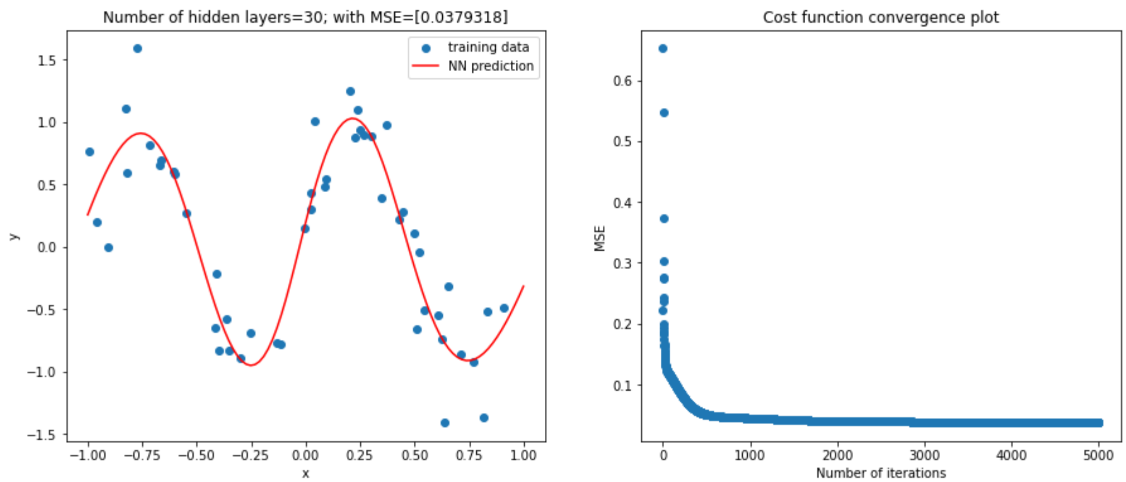
In [5]:
```python
W_1,W_2,B_1,B_2,Cost = Neural_Network_Simple(0.01,0.8,5000,1,3,1)

x_pre =np.linspace(-1,1,100).reshape(1,100)
y_pre =forwardNN_reg(W_1,W_2,B_1,B_2,x_pre)
```

```python
plt.xlabel('x');plt.ylabel('y')
plt.title('Number of hidden layers=' + str(3) + '; with MSE=' + str(Cost[-1]))
plt.legend()

plt.subplot(1,2,2)
plt.scatter(np.linspace(0,4999,5000),Cost)
plt.xlabel('Number of iterations');plt.ylabel('MSE')
plt.title('Cost function convergence plot')
plt.show()
```




In [6]:
```python
W_1,W_2,B_1,B_2,Cost = Neural_Network_Simple(0.01,0.8,5000,1,30,1)
```

```
plt.xlabel('x');plt.ylabel('y')
plt.title('Number of hidden layers=' + str(30) + '; with MSE=' + str(Cost[-1]))
plt.legend()

plt.subplot(1,2,2)
plt.scatter(np.linspace(0,4999,5000),Cost)
plt.xlabel('Number of iterations');plt.ylabel('MSE')
plt.title('Cost function convergence plot')
plt.show()
```




In [7]:
```
from tensorflow import keras
from tensorflow.keras import layers
```

In [8]:
```
model_NN = keras.models.Sequential()
model_NN.add(layers.Dense(units=30,activation='tanh',input_dim=1))
model_NN.add(layers.Dense(units=1))
optimiz = keras.optimizers.SGD(lr=0.2, momentum=0.8, decay=0.0, nesterov=False)

model_NN.compile(loss="mean_squared_error",optimizer=optimiz,metrics=['mean_absolute_error', 'mean_squared_error'])
history = model_NN.fit(X.T,T.T,batch_size=50,epochs=1500)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\keras\optimizers\optimizer_v2\gradient_descent.py:108: UserWarning: The `lr` argum
ent is deprecated, use `learning_rate` instead.
  super(SGD, self).__init__(name, **kwargs)
```

```
Epoch 1/1500
1/1 [==============================] - 1s 831ms/step - loss: 0.5004 - mean_absolute_error: 0.6175 - mean_squared_error: 0.500
4
Epoch 2/1500
1/1 [==============================] - 0s 16ms/step - loss: 0.4878 - mean_absolute_error: 0.5975 - mean_squared_error: 0.4878
Epoch 3/1500
1/1 [==============================] - 0s 0s/step - loss: 0.4914 - mean_absolute_error: 0.5837 - mean_squared_error: 0.4914
Epoch 4/1500
1/1 [==============================] - 0s 0s/step - loss: 0.4881 - mean_absolute_error: 0.5809 - mean_squared_error: 0.4881
Epoch 5/1500
1/1 [==============================] - 0s 0s/step - loss: 0.4914 - mean_absolute_error: 0.5891 - mean_squared_error: 0.4914
Epoch 6/1500
1/1 [==============================] - 0s 0s/step - loss: 0.4876 - mean_absolute_error: 0.5894 - mean_squared_error: 0.4876
Epoch 7/1500
1/1 [==============================] - 0s 0s/step - loss: 0.4853 - mean_absolute_error: 0.5899 - mean_squared_error: 0.4853
Epoch 8/1500
```

In [9]:
```
import pandas as pd
hist = pd.DataFrame(history.history)
```

In [9]:
```python
import pandas as pd
hist = pd.DataFrame(history.history)
hist.head(10)
```

Out[9]:

| | loss | mean_absolute_error | mean_squared_error |
|---|---|---|---|
| 0 | 0.500394 | 0.617545 | 0.500394 |
| 1 | 0.487786 | 0.597534 | 0.487786 |
| 2 | 0.491409 | 0.583684 | 0.491409 |
| 3 | 0.488079 | 0.580928 | 0.488079 |
| 4 | 0.491431 | 0.589096 | 0.491431 |
| 5 | 0.487643 | 0.589445 | 0.487643 |
| 6 | 0.485289 | 0.589922 | 0.485289 |
| 7 | 0.487575 | 0.593041 | 0.487575 |
| 8 | 0.486082 | 0.597627 | 0.486082 |
| 9 | 0.486323 | 0.598421 | 0.486323 |

In [10]:
```python
plt.figure(figsize=(15,6))
plt.subplot(1,2,1)
plt.scatter(X, T,label = 'training data')
plt.plot(np.transpose(x_pre),model_NN.predict(np.transpose(x_pre)),'r',label = 'NN prediction from Keras')
plt.xlabel('x');plt.ylabel('y')
plt.title('Number of hidden layers=' + str(30) + '; with MSE=' + str(hist['mean_squared_error'].iloc[-1]))
plt.legend()

plt.subplot(1,2,2)
plt.scatter(np.linspace(0,1499,1500),hist['mean_squared_error'])
plt.xlabel('Number of iterations');plt.ylabel('MSE')
plt.title('Cost function convergence plot from Keras')
plt.show()
```