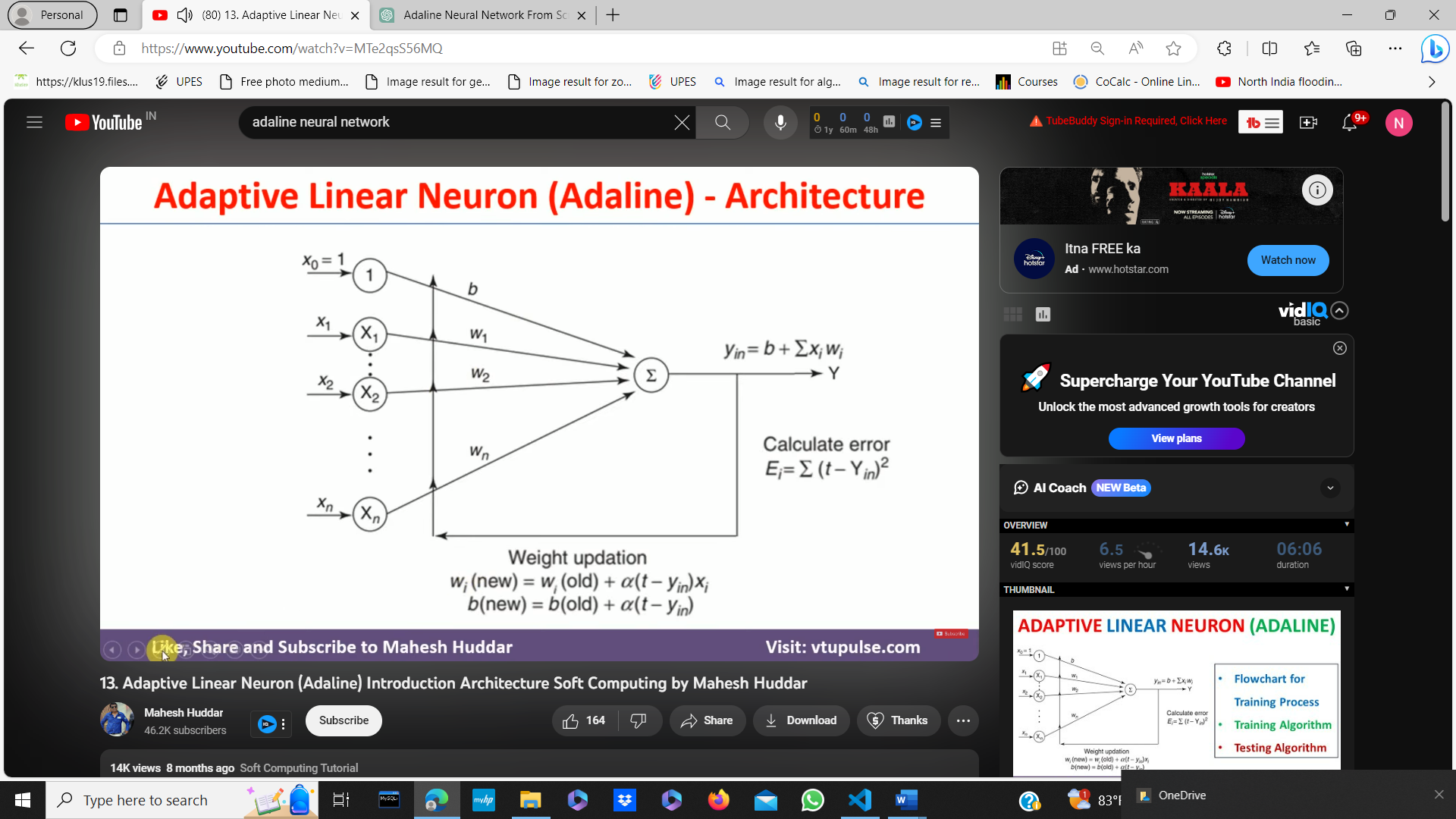
Experiment-5

Aim:- Implementation of Adaline(Adaptive Linear Neuron)

Adaline:- Adaline is a network with a single linear adaptive function or we can say single layer artificial neural network. In this adaptive linear neuron, input and output are in linear manner. In this we use Bipolar activation function with our input value. It make our prediction in 1 or -1.

**Code:-**

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  learning\_rate = 0.01 #step size during weight and bias updation  epochs = 100 #No of iteration training loop will perform over the data  # Generated sample data  X = np.array([[1, 2], [2, 3], [3, 4], [4, 5]])  y = np.array([1, 1, -1, -1])  # Initialize weights and bias to zeros  weights = np.zeros(X.shape[1])  bias = 0  costs = []  # Training loop  for \_ in range(epochs):  weighted\_sum = np.dot(X, weights) + bias # dot product of input x and new weight which is then added with the bias  # Compute the predicted output (activation function is the identity function)  predictions = weighted\_sum  # Compute the error  error = y - predictions  # mean squared error and append it to the list  cost = np.mean(error \*\* 2)  costs.append(cost)  # Update weights and bias  weights += learning\_rate \* np.dot(X.T, error)  bias += learning\_rate \* np.sum(error)  print("Updated weights:",weights)  print("Updated bias:",bias)  # Make predictions on new data  new\_data = np.array([[5, 6], [1, 1]])  weighted\_sum = np.dot(new\_data, weights) + bias  predictions = np.where(weighted\_sum >= 0, 1, -1)  print("Predictions:", predictions)  plt.plot(range(epochs), costs)  plt.xlabel('Epochs')  plt.ylabel('Cost')  plt.title('Adaline Training Cost')  plt.show() |

**Output:-** 