Name: Tania Rajabally

Roll No: 43

UID: 2017130047

Batch: C

Course Code: CEL71 (AI and Soft Computing Lab)

Date:02/09/2020

**Experiment No 3B**

**To implement Basic Neural Network learning rules**

**Aim:** To implement Basic Neural Network learning rules using Perceptron Learning Algorithm to distinguish between apple and orange.

**Theory:**

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so the network generates the best possible result without needing to redesign the output criteria.

A neural network works similarly to the human brain’s neural network. A “neuron” in a neural network is a mathematical function that collects and classifies information according to a specific architecture. The network bears a strong resemblance to statistical methods such as curve fitting and regression analysis.

A neural network contains layers of interconnected nodes. Each node is a perceptron and is similar to a multiple linear regression. The perceptron feeds the signal produced by a multiple linear regression into an activation function that may be nonlinear.

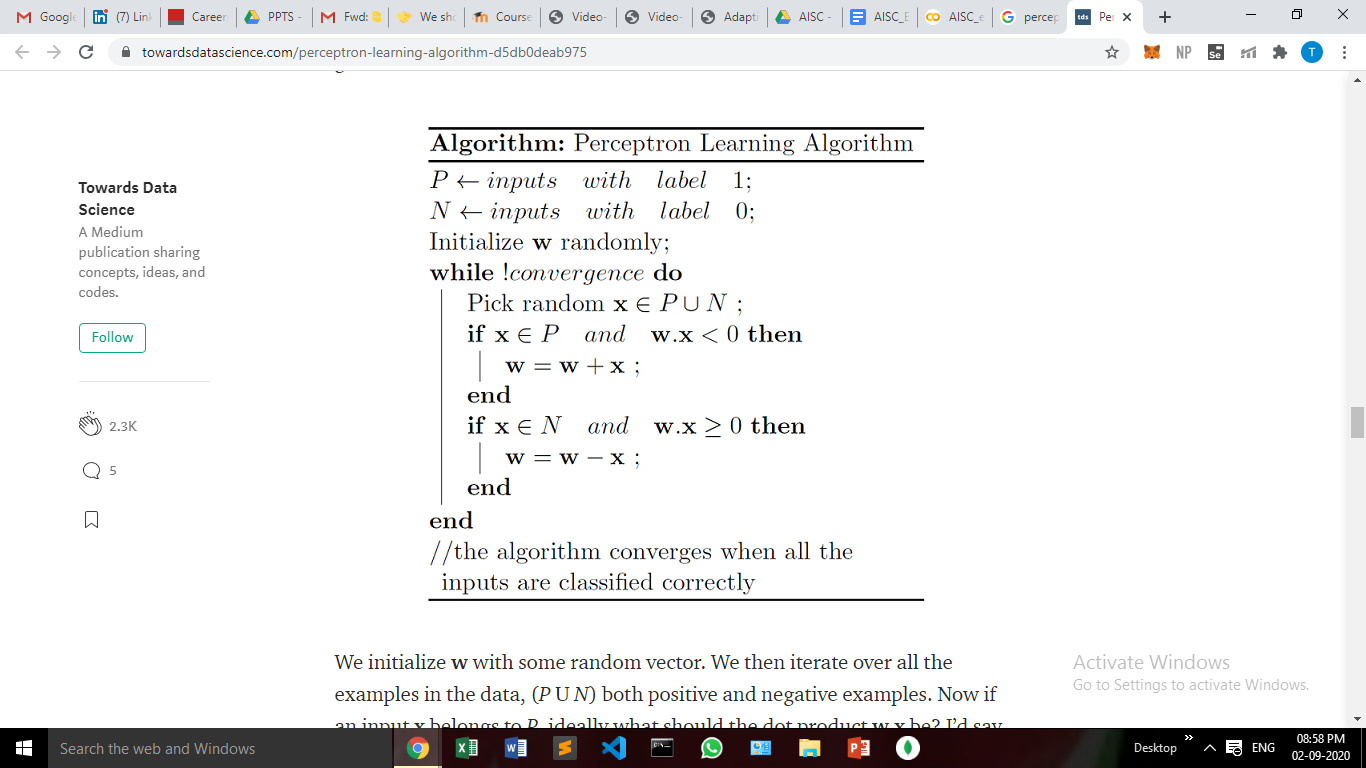
In a multi-layered perceptron (MLP), perceptrons are arranged in interconnected layers. The input layer collects input patterns. The output layer has classifications or output signals to which input patterns may map. For instance, the patterns may comprise a list of quantities for technical indicators about a security; potential outputs could be “buy,” “hold” or “sell.”

Hidden layers fine-tune the input weightings until the neural network’s margin of error is minimal. It is hypothesized that hidden layers extrapolate salient features in the input data that have predictive power regarding the outputs. This describes feature extraction, which accomplishes a utility similar to statistical techniques such as principal component analysis.

The perceptron model is a more general computational model than McCulloch-Pitts neuron. It takes an input, aggregates it (weighted sum) and returns 1 only if the aggregated sum is more than some threshold else returns 0.

A single perceptron can only be used to implement linearly separable functions. It takes both real and boolean inputs and associates a set of weights to them, along with a bias (the threshold thing I mentioned above). We learn the weights, we get the function.

**Procedure:**



We initialize w with some random vector. We then iterate over all the examples in the data, (P U N) both positive and negative examples. Now if an input x belongs to P, ideally what should the dot product w.x be? I’d say greater than or equal to 0 because that’s the only thing what our perceptron wants at the end of the day so let's give it that. And if x belongs to N, the dot product MUST be less than 0.

* Feed the features of the model that is required to be trained as input in the first layer.
* All weights and inputs will be multiplied – the multiplied result of each weight and input will be added up
* The Bias value will be added to shift the output function
* This value will be presented to the activation function (the type of activation function will depend on the need)
* The value received after the last step is the output value.

**Code:**

**ip = [[0,0],[0,1],[1,0],[1,1],[0.2,0.9],[0.9,0.2],[0.9,0.9],[0.3,0.3]]**

**op = [0,1,0,1,1,0,1,0]**

**theta = 0.5**

**lr = 0.3**

**w1 = 0**

**w2 = 0**

**b = 0**

**epochs = 1000**

**def calculate\_y(y,theta):**

**if y > theta:**

**return 1**

**else:**

**return 0**

**for i in range(epochs):**

**print("Epoch ",i)**

**for j in range(len(ip)):**

**yin = b + w1\*ip[j][0] + w2\*ip[j][1]**

**y = calculate\_y(yin,theta)**

**print("calculated", y, op[j])**

**if y != op[j]:**

**w1 = w1 + lr\*op[j]\*ip[j][0]**

**w2 = w2 + lr\*op[j]\*ip[j][1]**

**b = b + lr\*op[j]**

**print("w1", w1, "w2", w2, "b", b)**

**for i in ip:**

**print(i, end = " ")**

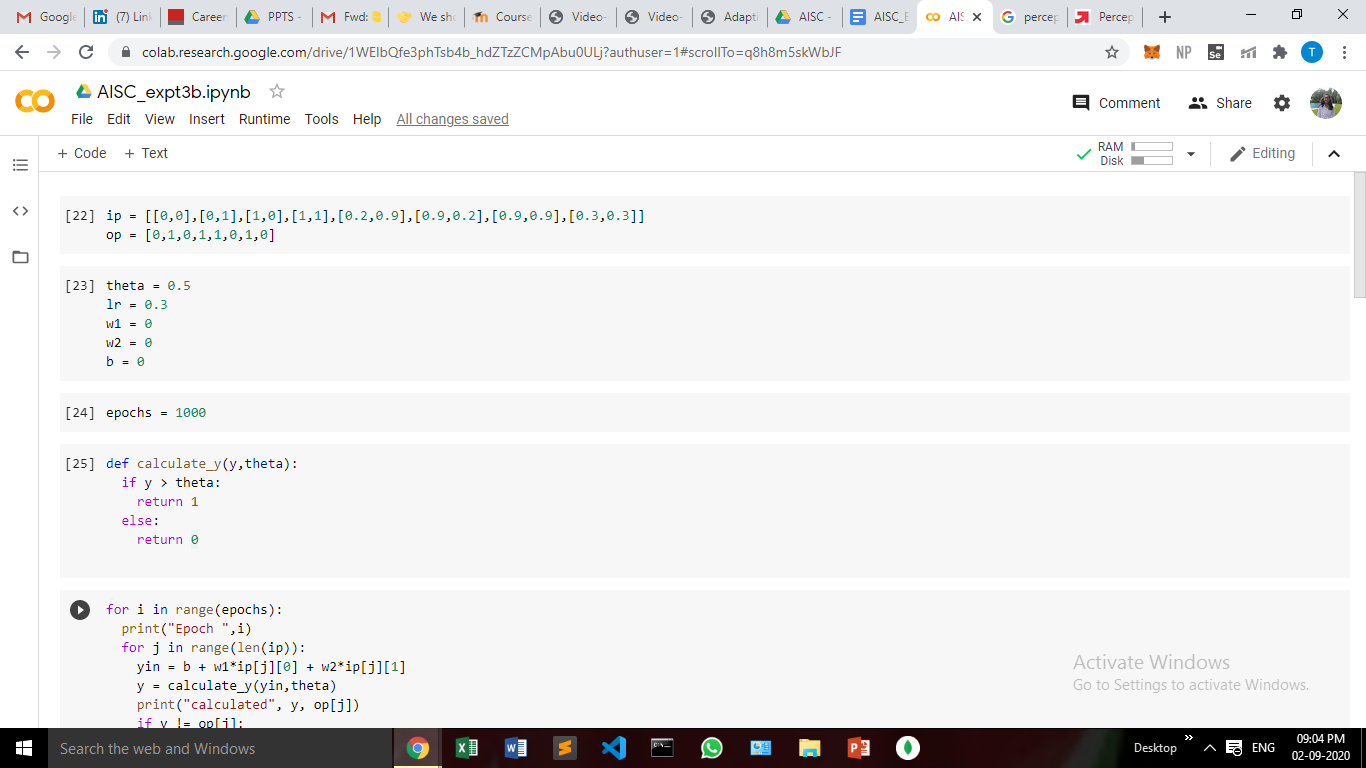
**y\_in = b + w1\*i[0] + w2\*i[1]**

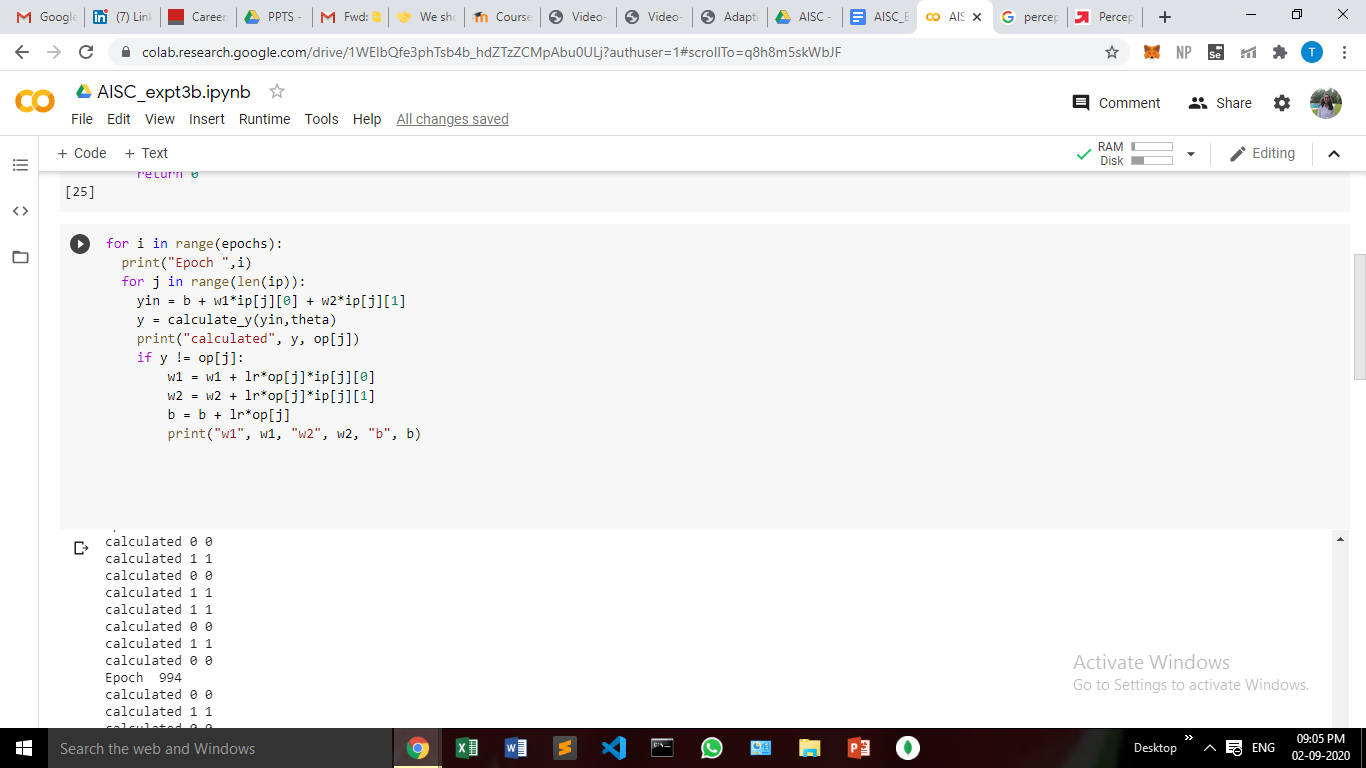
**if y\_in > theta:**

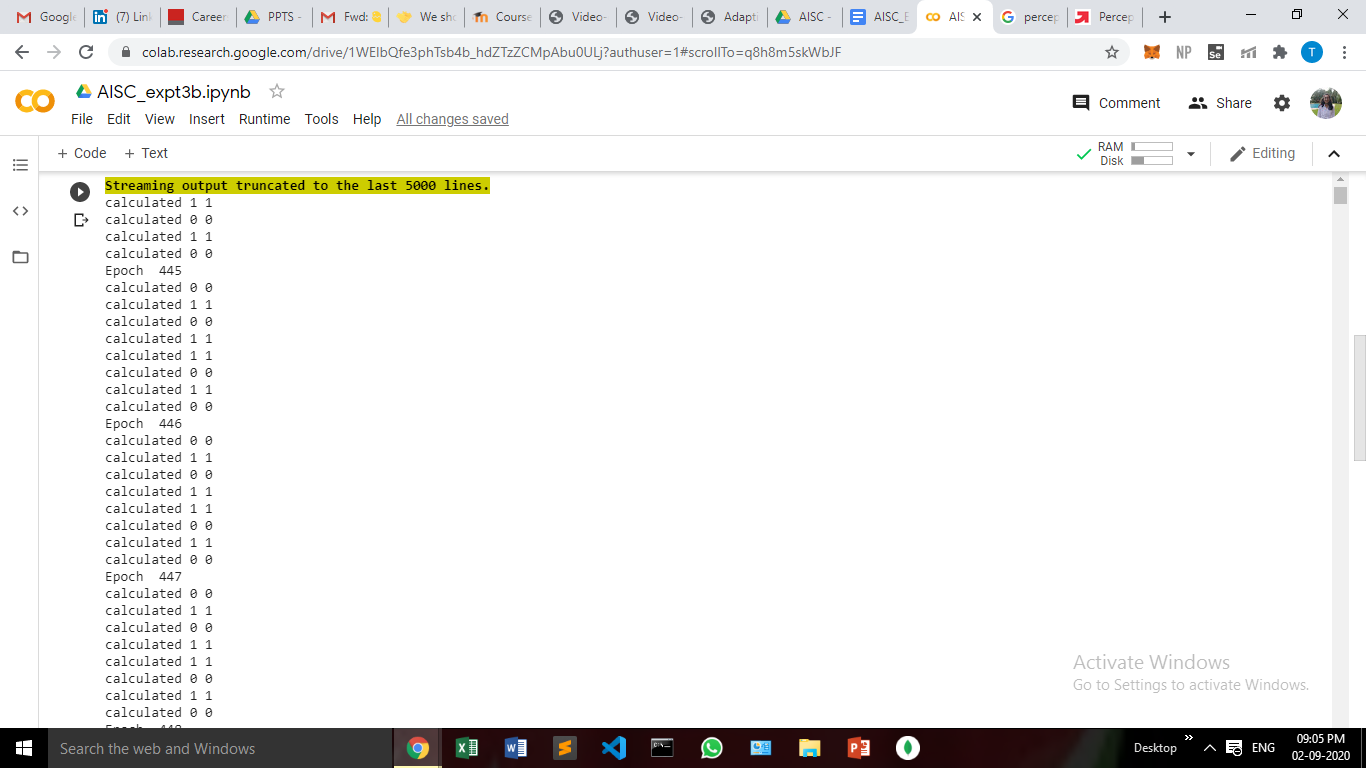
**print(1)**

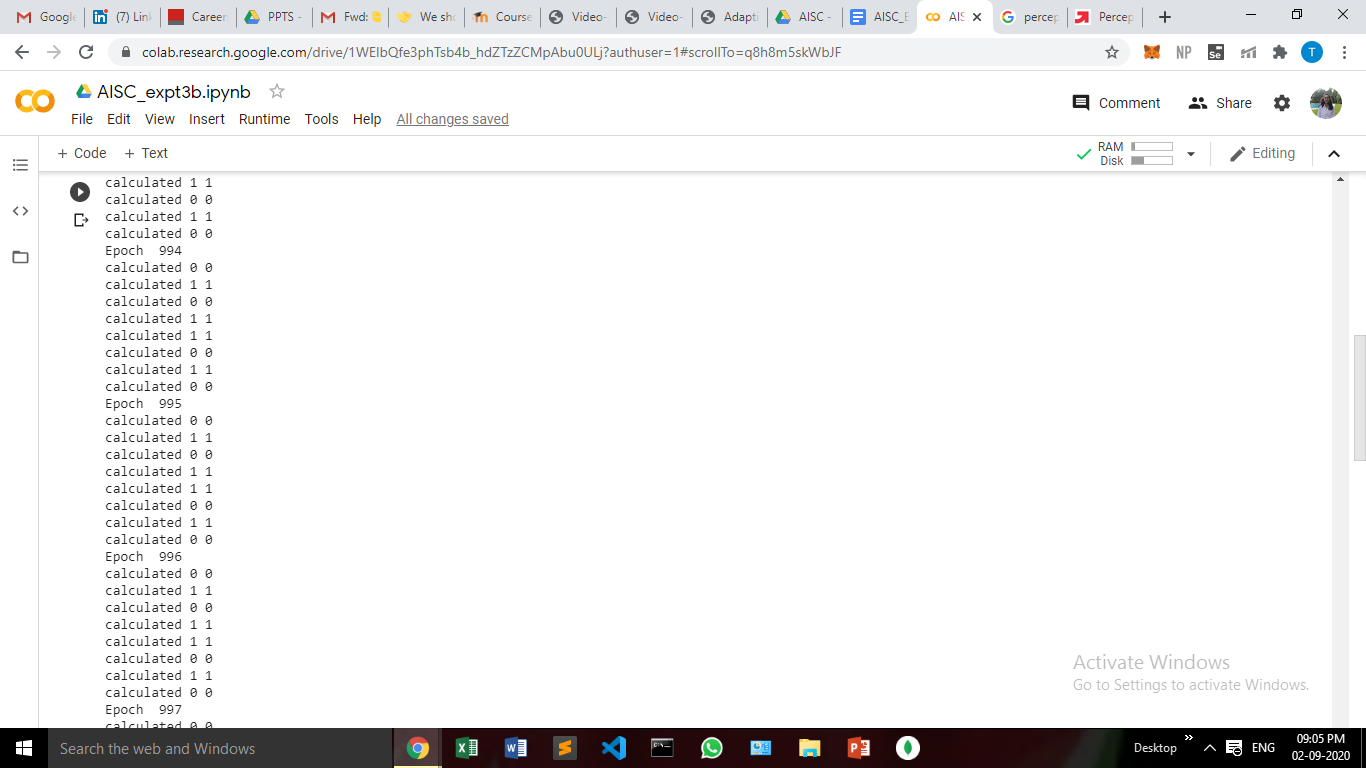
**else: print(0)**

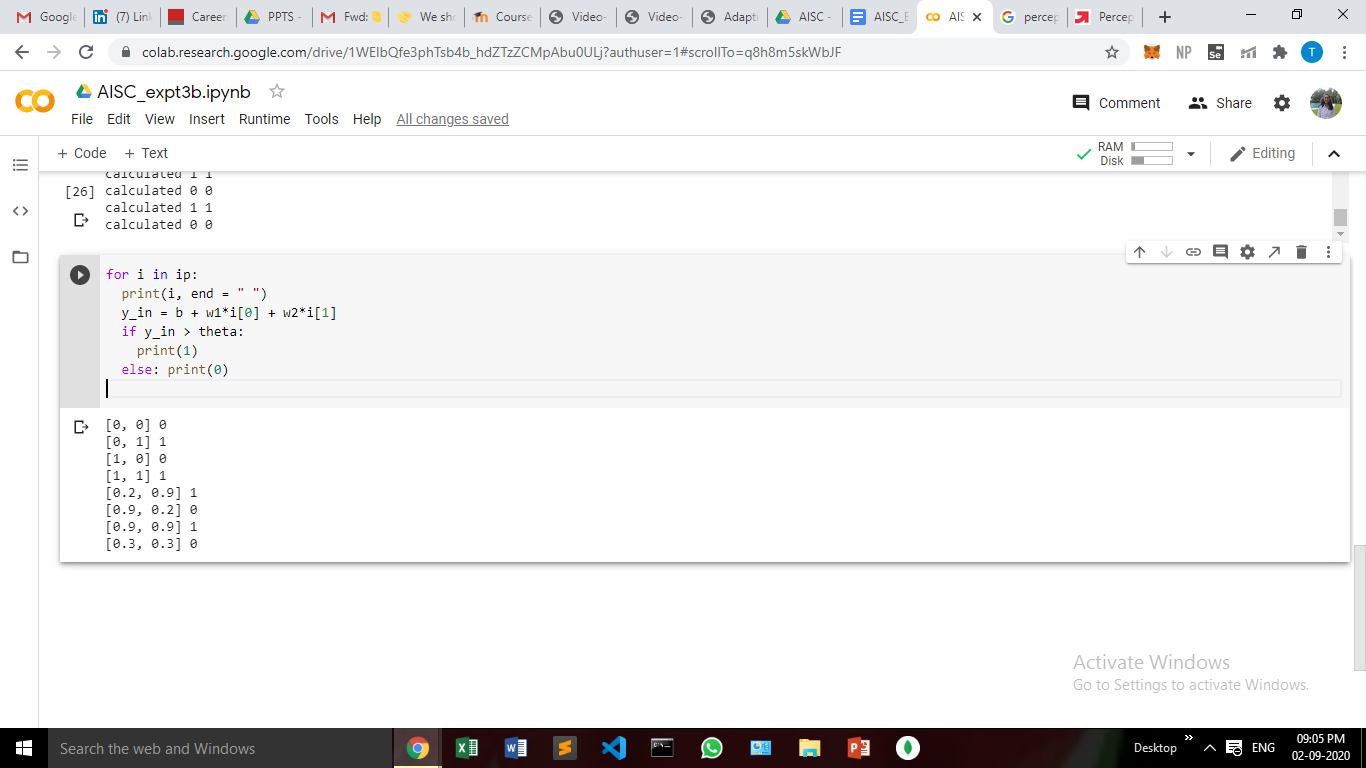
**Output:**

****

****

****

****

****

**Conclusion:**

In this experiment, we have distinguished between apples and oranges by implementing a basic neural network using perceptron learning algorithm. We have seen how the weights change and on that basis, the new output is calculated and the network keeps on learning.