Final Year B. Tech (EE)

Trimester: I **Subject: Artificial Intelligence and Machine Learning**

Name: Shreerang Mhatre **Class: TY**

Roll No: 52 **Batch: A3**

Experiment No: 01

Name of the Experiment: Calculate the output of a simple neuron

Marks	Teacher's Signature with date

Performed on: 14/09/2023

Submitted on: 29/09/2023

Aim: To create a simple neural network and calculate its output.

Prerequisite: Knowledge of logic gates, perceptron, various activation functions.

Objective:

To create a simple single layer neural network and calculate its output using Python Programming.

Components and Equipment required:

Python software

Theory:

Based on nature, neural networks are the usual representation we make of the brain: neurons interconnected to other neurons which forms a network. The operation of a complete neural network is straightforward: one enters variables as inputs, and after some calculations, an output is returned. Artificial neural network is usually put on columns, so that a neuron of the column *n* can only be connected to neurons from columns *n-1* and *n+1*. There are few types of networks that use a different architecture.

A simple artificial neural network is represented as below:

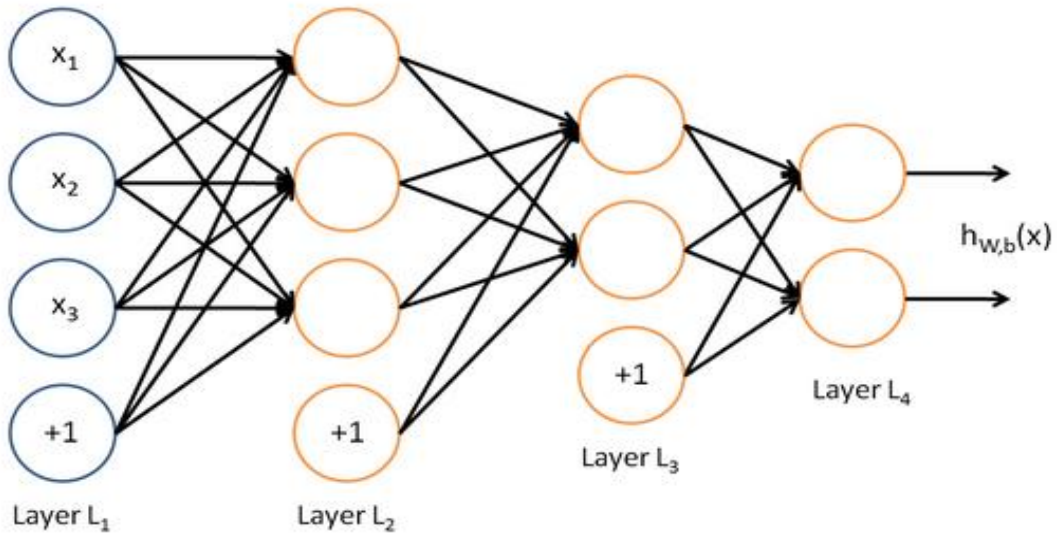


Figure 1 — Representation of a neural network

Neural networks can usually be read from left to right. Here, the first layer is the layer in which inputs are entered. There are 2 internals layers (called hidden layers) that do some math, and one last layer that contains all the possible outputs. “+1” s at the bottom of every column, it is something called “bias”**.**

Operation of a neuron:

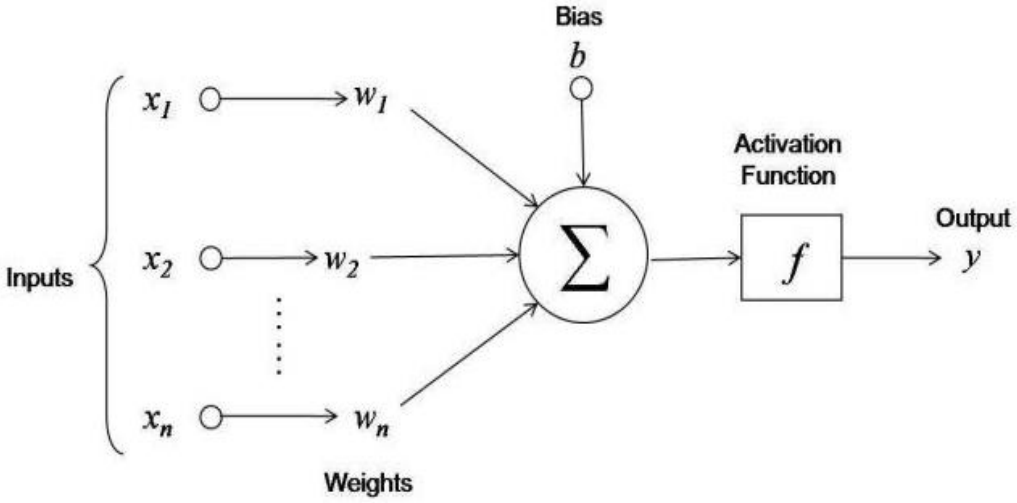


Figure 2 — Operations done by a neuron

First, it adds up the value of every neuron from the previous column it is connected to. On the Figure 2, there are 3 inputs (x1, x2, x3) coming to the neuron, so 3 neurons of the previous column are connected to our neuron. This value is multiplied, before being added, by another variable called “weight” (w1, w2, w3) which determines the connection between the two neurons. Each connection of neurons has its own weight, and those are the only values that will be modified during the learning process.

Moreover, a bias value may be added to the total value calculated. It is not a value coming from a specific neuron and is chosen before the learning phase, but can be useful for the network.

After all those summations, the neuron finally applies a function called “activation function” to the obtained value. Thus a neuron takes all values from connected neurons multiplied by their respective weight, add them, and apply an activation function. Then, the neuron is ready to send its new value to other neurons.

After every neurons of a column did it, the neural network passes to the next column. In the end, the last values obtained should be one usable to determine the desired output.

Network under consideration:

It consists of 2 neurons in the inputs column and 1 neuron in the output column. This configuration allows to create a simple classifier to distinguish 2 groups.

- if A is true and B is true, then A or B is true.
- if A is true and B is false, then A or B is true.
- if A is false and B is true, then A or B is true.
- if A is false and B is false, then A or B is false.

To create a simple Neural Network using Python:

Step-1

```
import numpy, random, os
lr = 1 #learning rate
bias = 1 #value of bias
weights = [random.random(),random.random(),random.random()] #weights generated in a list (3 weights in total for 2 neurons and the bias)
```

In the beginning of the program we define libraries and the values of the parameters, and creates a list which contains the values of the weights that will be modified.

Step-2

```
def Perceptron(input1, input2, output) :
outputP = input1*weights[0]+input2*weights[1]+bias*weights[2]
if outputP > 0 : #activation function (here Heaviside)
outputP = 1
else :
outputP = 0
error = output - outputP
weights[0] += error * input1 * lr
weights[1] += error * input2 * lr
weights[2] += error * bias * lr
```

Here we create a function which defines the work of the output neuron. It takes 3 parameters (the 2 values of the neurons and the expected output). "outputP" is the variable corresponding to the output given by the Perceptron. Then we calculate the error, used to modify the weights of every connection to the output neuron right after.

Step-3

```
for i in range(50) :
Perceptron(1,1,1) #True or true
Perceptron(1,0,1) #True or false
Perceptron(0,1,1) #False or true
Perceptron(0,0,0) #False or false
```

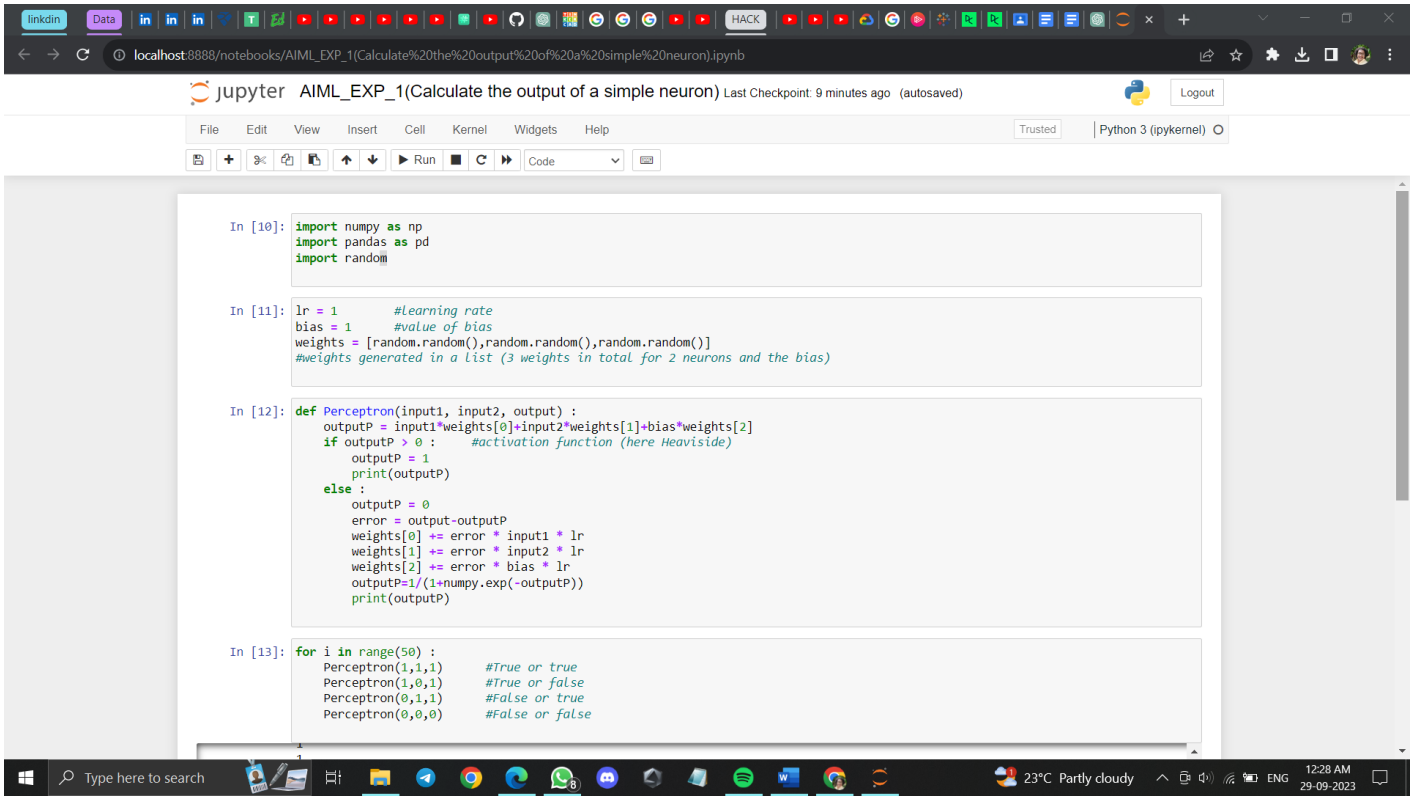
We create a loop that makes the neural network repeat every situation several times. This part is the learning phase. The number of iteration is chosen according to the precision we want.

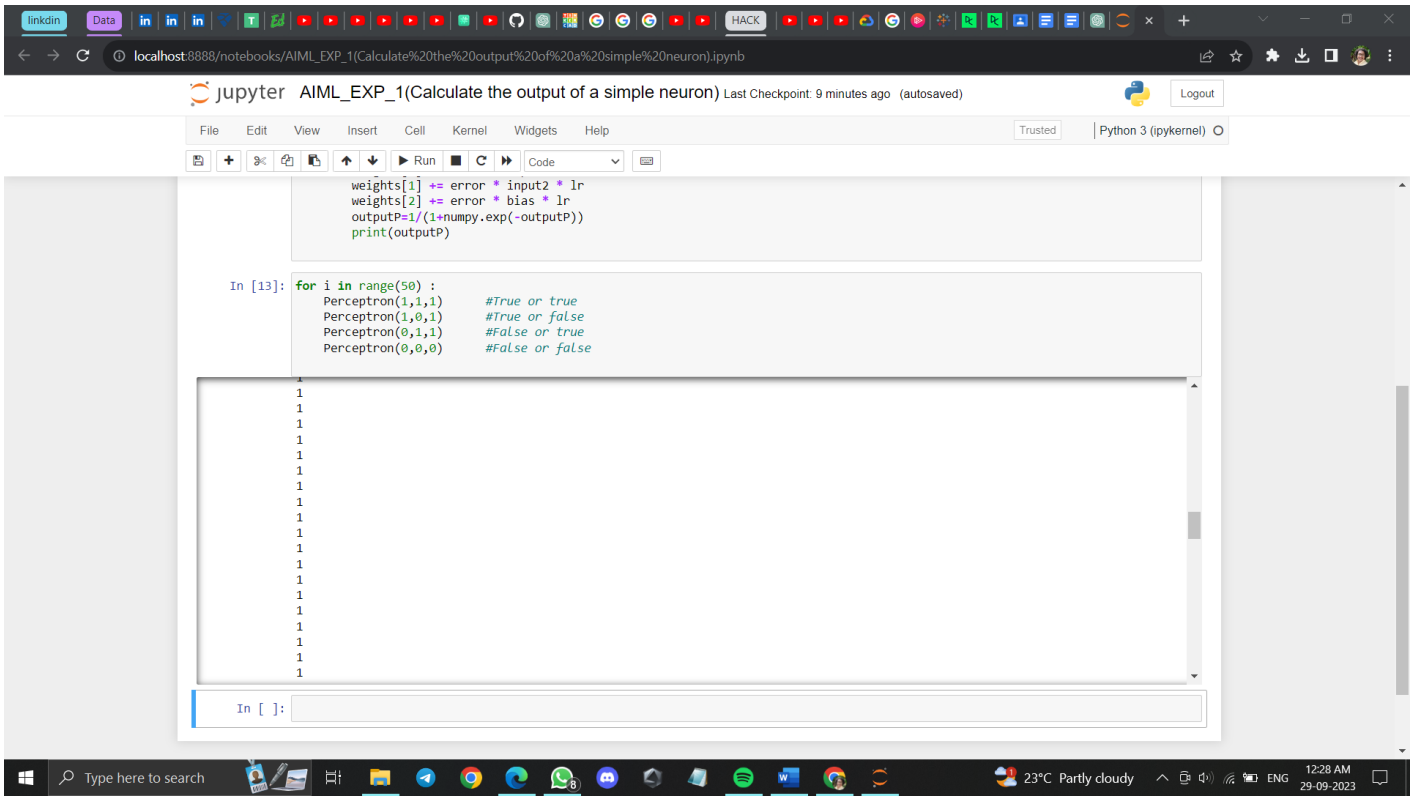
Conclusion:

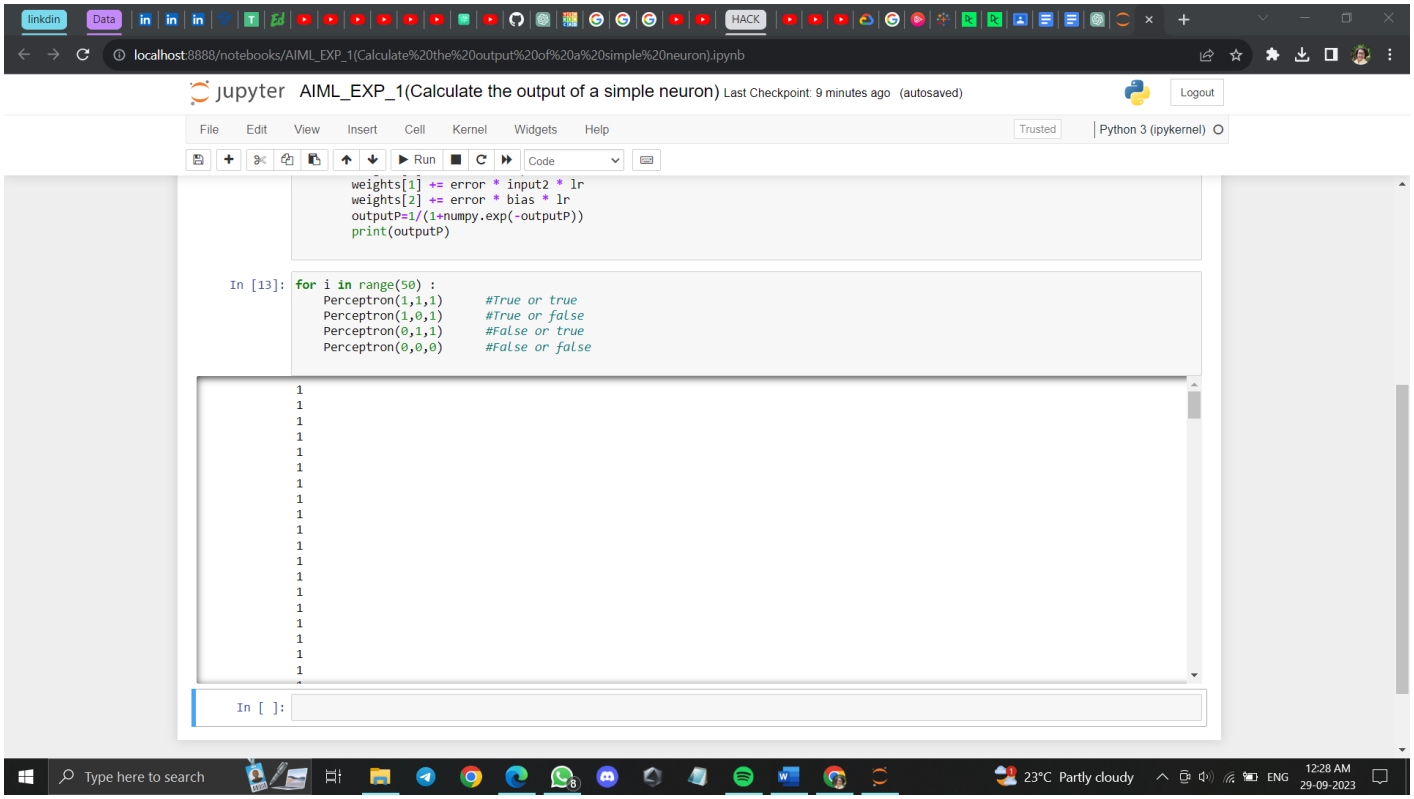
Post Lab Questions:

1. Comment on the similarities between biological neuron and artificial neuron.
2. What do you mean by activation function? State and explain its types.
3. Implement the above code considering sigmoid function.

(`outputP = 1/(1+numpy.exp(-outputP)) #sigmoid function)`


localhost:8888/notebooks/AIML_EXP_1(Calculate%20the%20output%20of%20a%20simple%20neuron).ipynb
jupyter AIML_EXP_1(Calculate the output of a simple neuron) Last Checkpoint: 9 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [10]:
import numpy as np
import pandas as pd
import random
In [11]:
lr = 1 #Learning rate
bias = 1 #value of bias
weights = [random.random(),random.random(),random.random()]
#weights generated in a list (3 weights in total for 2 neurons and the bias)
In [12]:
def Perceptron(input1, input2, output) :
outputP = input1*weights[0]+input2*weights[1]+bias*weights[2]
if outputP > 0 : #activation function (here Heaviside)
outputP = 1
print(outputP)
else :
outputP = 0
error = output-outputP
weights[0] += error * input1 * lr
weights[1] += error * input2 * lr
weights[2] += error * bias * lr
outputP=1/(1+numpy.exp(-outputP))
print(outputP)
In [13]:
for i in range(50) :
Perceptron(1,1,1) #True or true
Perceptron(1,0,1) #True or false
Perceptron(0,1,1) #False or true
Perceptron(0,0,0) #False or false
Type here to search
23°C Partly cloudy
ENG
12:28 AM
29-09-2023


localhost:8888/notebooks/AIML_EXP_1(Calculate%20the%20output%20of%20a%20simple%20neuron).ipynb
jupyter AIML_EXP_1(Calculate the output of a simple neuron) Last Checkpoint: 9 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
weights[1] += error * input2 * lr
weights[2] += error * bias * lr
outputP=1/(1+numpy.exp(-outputP))
print(outputP)
In [13]:
for i in range(50) :
Perceptron(1,1,1) #True or true
Perceptron(1,0,1) #True or false
Perceptron(0,1,1) #False or true
Perceptron(0,0,0) #False or false
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
In [ ]:
Type here to search
23°C Partly cloudy
ENG
12:28 AM
29-09-2023


localhost:8888/notebooks/AIML_EXP_1(Calculate%20the%20output%20of%20a%20simple%20neuron).ipynb
jupyter AIML_EXP_1(Calculate the output of a simple neuron) Last Checkpoint: 9 minutes ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
weights[1] += error * input2 * lr
weights[2] += error * bias * lr
outputP=1/(1+numpy.exp(-outputP))
print(outputP)
In [13]:
for i in range(50) :
Perceptron(1,1,1) #True or true
Perceptron(1,0,1) #True or false
Perceptron(0,1,1) #False or true
Perceptron(0,0,0) #False or false
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
In [ ]:
Type here to search
23°C Partly cloudy
ENG
12:28 AM
29-09-2023

Expt. 1- 3

Expl - Calculate the output of a simple neuron

PAGE No.
DATE 14/9/23

* Post lab Questions -

Q1) Comment on the similarites between biological neuron & artificial neuron.

-> Biological neurons & artificial neurons share several key similarities -

1. Information Processing - Both process information
2. Input integration - Both integrate incomming signals
3. weighting of inputs - Both assign weights to inputs.
4. Activation / Threshelding - Both use a threshold to decide when to transmit.
5. output Transmission - Transmint information downstream.
6. Learning & Adaptation - Both can adapt & learn from experience
7. Network connectivity - Both are part of larger networks.
8. Non - Linear Transformations - Both can perform non-linear transformations, allowing them to capture complex patterns.

Q2) What do you mean by activation function? State and explain its types

→ An activation function in neural networks is a mathematical function applied to the weighted sum of inputs to a neuron to determine its output. It introduces non-linearity to the model, allowing neural networks to approximate complex relationships in data. Here are some common types of activation functions-

1. Sigmoid Function
2. Hyperbolic Tangent Function
3. Rectified Linear Unit (ReLU)
4. Leaky ReLU
5. Parametric ReLU (PReLU)
6. Exponential Linear Unit (ELU).
7. Swish
8. Gated Recurrent Unit (GRU).

→

PAGE No.
DATE / /

Q3) Implement the above code considering sigmoid function.

→ code -

```
import numpy as np

# Define the output variable
output = 0.5

# Apply the sigmoid function to the output
outputP = 1/(1+np.exp(-outputP)) #sigmoid function

# Print the result
print("Output after applying sigmoid:", outputP)
```