SAP ID:

SHRI VILEPARLE KELAVANI MANDAL'S
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC ACCREDITED with "A" GRADE (CGPA : 3.18)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

COURSE CODE: DJ19DSC501 **DATE:**

COURSE NAME: Machine Learning - II **CLASS: AY 2022-23**

LAB EXPERIMENT NO.1 AIM

60009210033

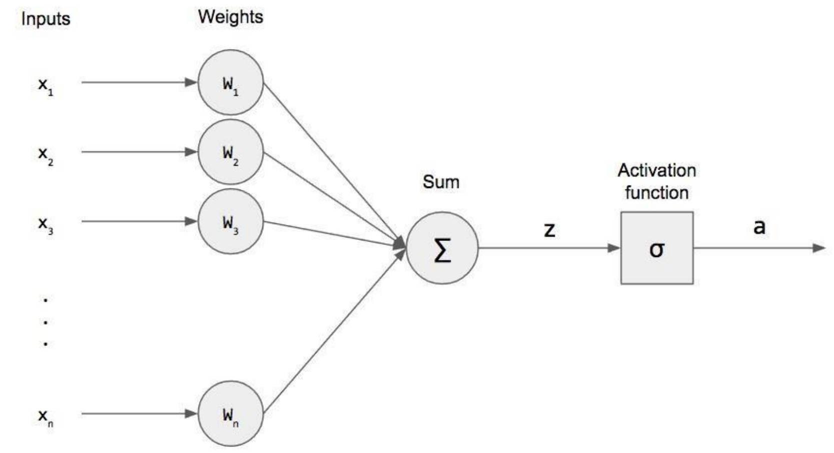
Jhanvi Parekh

D11

Implement Boolean gates using perceptron – Neural representation of Logic Gates.

THEORY:

Perceptron is a Supervised Learning Algorithm for binary classifiers.



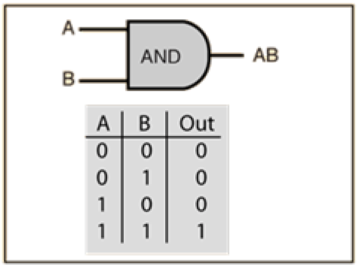
For a particular choice of the weight vector w and bias parameter b, the model predicts output ŷ for the corresponding input vector x.

$$\begin{aligned} \hat{y} &= \Theta\left(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b\right) \\ &= \Theta(\mathbf{w} \cdot \mathbf{x} + b) \\ \text{where } \Theta(v) &= \begin{cases} 1 & \text{if } v \geqslant 0 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

The input values, i.e., x1, x2, and bias is multiplied with their respective weight matrix that is W1, W2, and W0. The corresponding value is then fed to the summation neuron where the summed value is calculated. This is fed to a neuron which has a non-linear function(sigmoid in our case) for scaling the output to a desirable range. The scaled output of sigmoid is 0 if the output is less than 0.5 and 1 if the output is greater than 0.5. The main aim is to find the value of weights or the weight vector which will enable the system to act as a particular gate.

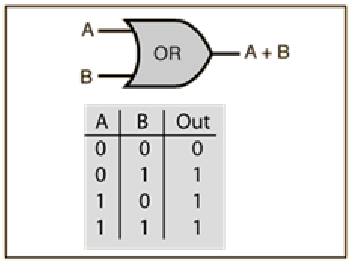
Boolean gates – Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on a certain logic.

1) AND



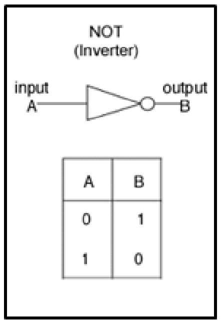
A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

2) OR



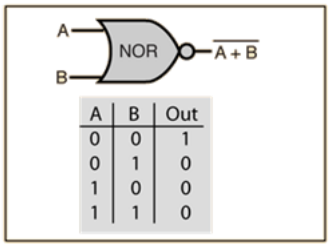
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

3) NOT



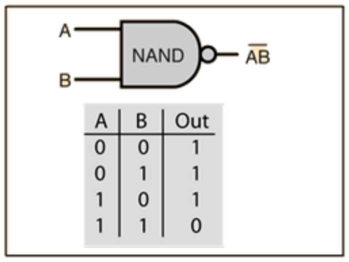
A	B
0	1
1	0

4) NOR



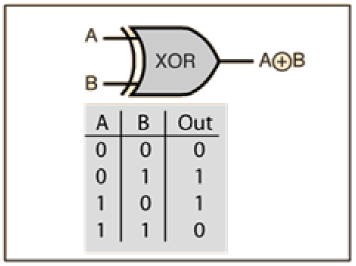
A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

5) NAND



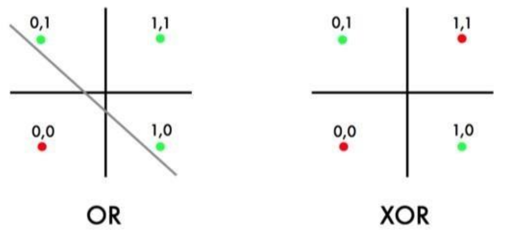
A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

6) XOR



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

For the XOR gate, the output can't be linearly separated. Uni layered perceptrons can only work with linearly separable data. But in the following diagram drawn in accordance with the truth table of the XOR logical operator, we can see that the data is NOT linearly separable.



To solve this problem, we add an extra layer to our vanilla perceptron, i.e., we create a Multi Layered Perceptron (or MLP). We call this extra layer as the Hidden layer. To build a perceptron, we first need to understand that the XOR gate can be written as a combination of AND gates, NOT gates and OR gates in the following way:

$$XOR(x_1, x_2) = AND(NOT(AND(x_1, x_2)), OR(x_1, x_2))$$

Hidden layers are those layers with nodes other than the input and output nodes. An MLP is generally restricted to having a single hidden layer. The hidden layer allows for non-linearity. A node in the hidden layer isn't too different to an output node: nodes in the previous layers connect to it with their own weights and biases, and an output is computed, generally with an activation function.

1. **Implement the 6 Boolean gates above using perceptrons. Inputs = x1, x2 and bias, weights should be fed into the perceptron with single Output = y**
2. **Display final weights and bias of each perceptron.**
3. **What are the limitations of the perceptron network?**
   Ans: Only linearly separable functions may be learned by perceptrons. This indicates that a perceptron will not be able to learn the decision boundary accurately if the data cannot be divided into two classes using a single hyperplane (a straight line in two dimensions or a flat plane in higher dimensions).
   Binary Output: Perceptrons only work well for binary classification applications since they only produce binary outputs (0 or 1). A single perceptron cannot solve many real-world issues that call for multi-class categorization without extensions.
   Absence of Hidden Layers: Because perceptrons only have one layer, they are unable to simulate intricate interactions between features. Perceptrons lack the depth that deep learning networks with numerous hidden layers exhibit for a variety of tasks.
   Inability to Solve Non-Linear situations: Non-linear decision boundaries are a common feature of real-world situations. Perceptrons can only be used to solve problems with linear decision boundaries since they are unable to capture these non-linear interactions.
   Perceptrons can have low generalization and accuracy because they are susceptible to noisy data and outliers.

CONCLUSION:

Base all conclusions on your actual results; describe the meaning of the experiment and the implications of your results.

In the "Boolean Gate via Perceptron" experiment, several Boolean logic gates, such as AND, OR, NOT, NOR, NAND, and XOR gates, are simulated and replicated using perceptrons, which are straightforward artificial neural network units.

AND Gate:

Outputs true (1) only if both input values are true (1).

Implementation: In a perceptron model, this can be achieved by assigning appropriate weights and a bias such that the perceptron only fires when both inputs are activated.

OR Gate:

Outputs true (1) if at least one input value is true (1).

Implementation: Like the AND gate, perceptrons can be configured to act as OR gates by adjusting weights and biases.

NOT Gate:

Inverts the input value. True (1) becomes false (0), and vice versa.

Implementation: In this case, a single perceptron can be trained to act as a NOT gate by setting appropriate weights and biases.

NOR Gate:

Outputs true (1) only if both input values are false (0).

Implementation: NOR gates can be represented using perceptrons by modifying the weights and bias accordingly.

NAND Gate:

Outputs false (0) only if both input values are true (1).

Implementation: Similar to AND and OR gates, perceptrons can be configured to behave as NAND gates.

XOR Gate:

Outputs true (1) if the number of true (1) inputs is odd.

Implementation: XOR gates are more complex than the previous gates and cannot be directly represented using single perceptrons. However, they can be constructed using combinations of multiple perceptrons and layers in a neural network.

Link: https://colab.research.google.com/drive/1CWKKQezMNrzhyeuqxy4ZvQuEIDTh9M7k?usp=sharing

Boolean Gate via Perceptron

And Gate

```
import numpy as np

def unitStep (y_in):
if y_in>=0:



    return 1



  else:



    return 0



def perceptron(x,w,b):
y_in=np.dot(w,x) + b
y_hat=unitStep(y_in)
return y_hat



def And_fun(x):
w=np.array([1,1])


  b=-2


  return perceptron(x,w,b)


test1=np.array([0,0])
test2=np.array([0,1])
test3=np.array([1,0])
test4=np.array([1,1])


print(And_fun(test1))
print(And_fun(test2))
print(And_fun(test3))
print(And_fun(test4))
```

```
     0
     0
     0
     1
```

OR Gate

```
import numpy as np

def unitStep (y_in):
if y_in>=-0:
return 1   else:
return 0




def perceptron(x,w,b):
y_in=np.dot(w,x) + b
y_hat=unitStep(y_in)
return y_hat


def or_fun(x):
w=np.array([1,1])   b=-1


  return perceptron(x,w,b)


test1=np.array([0,0])
test2=np.array([0,1])
```

```
test3=np.array([1,0])
test4=np.array([1,1])


print(or_fun(test1))
print(or_fun(test2))
print(or_fun(test3))
print(or_fun(test4)) 0
     1
     1
     1
```

Not gate

```
import numpy as np

def unitStep (y_in):
if y_in<=0:
return 1   else:
return 0


def perceptron(x,w,b):
y_in=np.dot(w,x) + b
y_hat=unitStep(y_in)
return y_hat


def not_fun(x):   w=1
b=0   return
perceptron(x,w,b)


test1=np.array([0])
test2=np.array([1])


print(not_fun(test1))
print(not_fun(test2))


     1
     0
```

NAND GATE

```
import numpy as np

def unitStep (y_in):
if y_in<=0:
return 1   else:
return 0


def perceptron(x,w,b):
y_in=np.dot(w,x) + b
y_hat=unitStep(y_in)
return y_hat


def nand_fun(x):
w=np.array([1,1])   b=-1

  return perceptron(x,w,b)


test1=np.array([0,0])
test2=np.array([0,1])
test3=np.array([1,0])
test4=np.array([1,1])


print(nand_fun(test1))
print(nand_fun(test2))
print(nand_fun(test3))
print(nand_fun(test4))

     1
     1
     1
     0
```

NOR Gate

```
import numpy as np

def unitStep (y_in):
if y_in<=0:
return 1   else:
return 0


def perceptron(x,w,b):
y_in=np.dot(w,x) + b
y_hat=unitStep(y_in)
return y_hat


def nor_fun(x):
w=np.array([1,1])   b=0
return perceptron(x,w,b)


test1=np.array([0,0])
test2=np.array([0,1])
test3=np.array([1,0])
test4=np.array([1,1])


print(nor_fun(test1))
print(nor_fun(test2))
print(nor_fun(test3))
print(nor_fun(test4))
```

```
     1
     0
     0
    0 EX-
```

OR gate

```
import numpy as np

def unitStep (y_in):
if y_in==-1:
return 1   else:
return 0


def perceptron(x,w,b):
y_in=np.dot(w,x) + b
y_hat=unitStep(y_in)
return y_hat


def nor_fun(x):
w=np.array([1,1])   b=-2


  return perceptron(x,w,b)


test1=np.array([0,0])
test2=np.array([0,1])
test3=np.array([1,0])
test4=np.array([1,1])


print(nor_fun(test1))
print(nor_fun(test2))
print(nor_fun(test3))
print(nor_fun(test4))
```

```
     0
     1
     1
     0
```