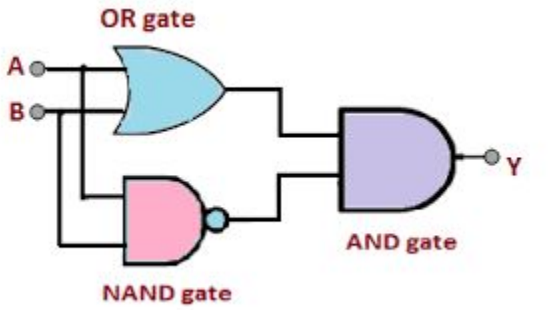
Design XOR Gate

CS550 - Machine Learning and Business Intelligence



Submitted by: Divya Pandey(19665)
Instructor: Dr. Henry Chang

Table of Content

Introduction

- ★ The XOR (Exclusive OR) gate is a logical operation that outputs true only when the inputs differ (one is true and the other is false).

- ★ The XOR gate problem was one of the early demonstrations of the power of neural networks and was used to show that neural networks could solve problems that traditional rule-based systems could not

1.

Design

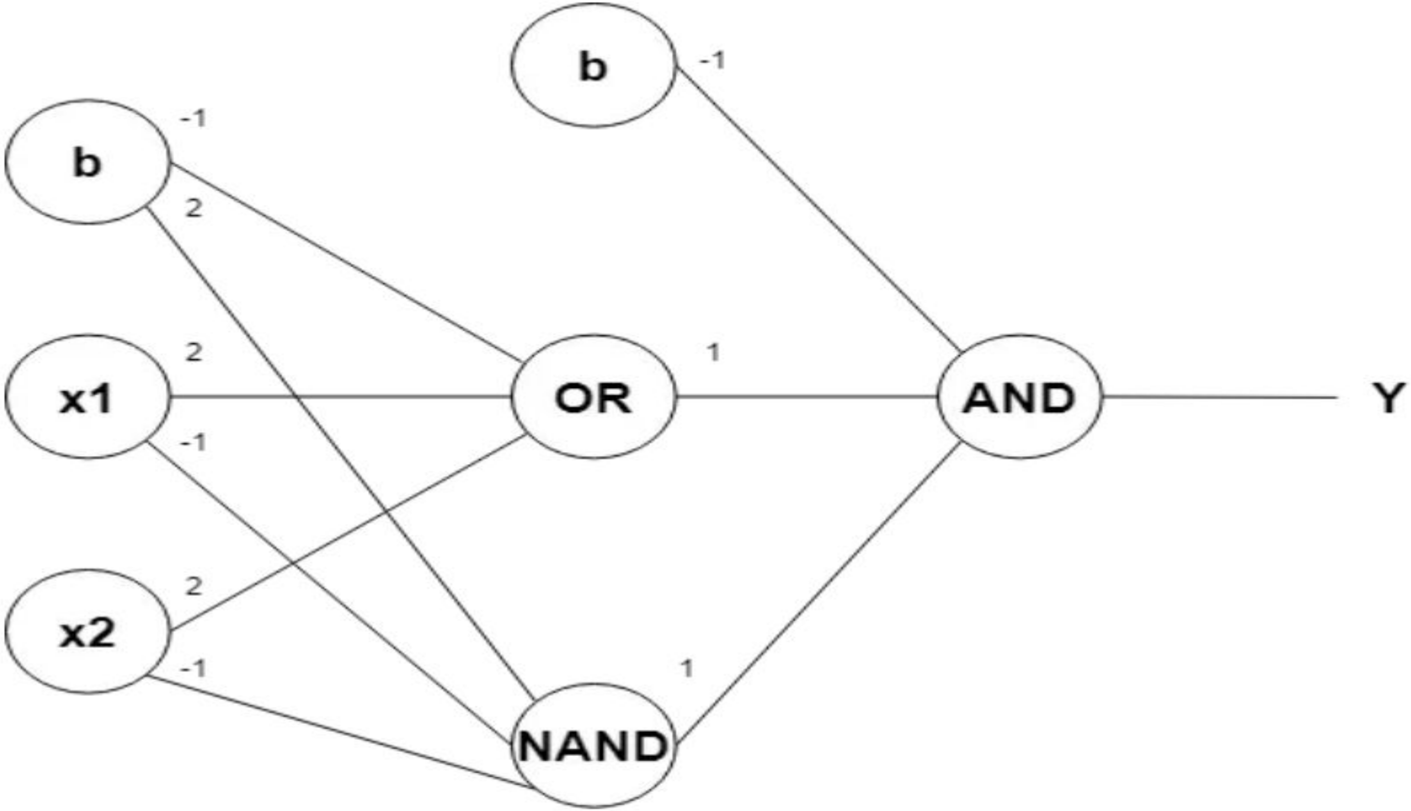
A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

XOR Gate

The boolean representation of an XOR gate is;

> $x1x`2 + x`1x2$

Design



Design

Multilayer Perceptptron

```
      +-------------------------------------------------------------+
      |      "NAnd"  +---------+                   "XOr" Function  |
      | 1.0-->2.0--->|         |                                   |
      |              |         |                                   |
X --->|----> -0.75-->| Neuron  |-->Z2--                            |
      |\             |         |       \                           |
Y---->|-\--> -0.75-->|         |        \ "And"   +---------+      |
      |\ \           +---------+         ->0.75-->|         |      |
    |  \  \                                     | Neuron  |-->Z3-->|-->Z
      |  \ \  "Or"   +---------+        -->0.75-->|         |      |
      |   \ -->1.5-->|         |        /         +---------+      |
      |    \         | Neuron  |-->Z1--                            |
      |      -->1.5-->|         |                                   |
      |              +---------+                                   |
      +-------------------------------------------------------------+
                          With Bias
```

```
     XOR
-----------
X  Y  |  Z
-----------
0  0  |  0
0  1  |  1
1  0  |  1
1  1  |  0
```

Implementation

Design **AND** Gate to calculate the values of **W1,W2 and Y**

Z := (W1 * X + W2 * Y >= T)

where T := 1.0.

Desired "**And**" Function

X	**Y**	**Z**
0	0	0
0	1	0
1	0	0
1	1	1

Implementation

Loop 1:

W1=W2=0

Z := (0 * X + 0 * Y >= T)

X	**Y**	**Z**
0	0	0
0	1	0
1	0	0
1	1	0

Implementation

Loop 2:

W1=W2=0.5

Z := (0.5 * X + 0.5 * Y >= T)

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Implementation

Design **OR** Gate to calculate the values of **W1,W2 and Y**

Z := (W1 * X + W2 * Y >= T)

where T := 1.0.

Desired "**OR**" Function

X	**Y**	**Z**
0	0	0
0	1	1
1	0	1
1	1	1

Implementation

Loop 1:

W1=W2=0

Z := (0* X + 0 * Y >= T)

X	**Y**	**Z**
0	0	0
0	1	0
1	0	0
1	1	0

Implementation

Loop 2:

W1=W2=0.5

Z := (0.5* X + 0.5 * Y >= T)

X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Implementation

Loop 3:

W1=W2=1.0

Z := (1.0* X + 1.0 * Y >= T)

X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Implementation

Design **NAND** Gate to calculate the values of **W1,W2 and Y**

Z := (W0 * C + W1 * X + W2 * Y >= T)

where T := 1.0.

The bias C for NAND is 1.0

Desired **"NAND"** Function

X	**Y**	**Z**
0	0	1
0	1	1
1	0	1

Implementation

Loop 1 :

W0=0.0

W1=W2=0.5

Z := (0 * 1.0 + 0.5 * X + 0.5 * Y >= T)

C	X	Y	Z
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Implementation

Loop 2:

W0=0.5

W1=W2=0.5

Z := (0.5* 1.0 + 0.5 * X + 0.5 * Y >= T)

C	X	Y	Z
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Implementation

Loop 3:

W0=1.0

W1=W2=0.5

Z := (1.0* 1.0 + 0.5 * X + 0.5 * Y >= T)

C	X	Y	Z
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Implementation

Loop 4:

W0=1.0

W1=W2=0.0

Z := (1.0* 1.0 + 0.0 * X + 0.0 * Y >= T)

C	X	Y	Z
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Implementation

Loop 5:

W0=1.0

W1=W2=-0.5

Z := (1.0* 1.0 + -0.5 * X + -0.5 * Y >= T)

C	X	Y	Z
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Implementation

Loop 6:

W0=1.5

W1=W2=-0.5

Z := (1.5* 1.0 + -0.5 * X + -0.5 * Y >= T)

C	X	Y	Z
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Test

- What is the formula for

Z1 := X "AND" Y

Z1 := (0.5 * X + 0.5 * Y >= 1.0)

- **What is the formula for**

Z1 := X "OR" Y

Z1 := (1.0 * X + 1.0 * Y >= 1.0)

- **What is the formula for**

Z1 := X "NAND" Y

Bias is +1.5 , C = 1; W0 = 1.5; W1=W2 = -0.5

Z2 := (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0)

Z2 := (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0)

Z2 := (- 0.5 * Y >= 0.5 * X + -1.5 * 1.0 + 1.0)

Z2 := (-0.5 * Y >= 0.5 * X – 0.5)

Z2 := (Y <= -X + 1.0)

Test

- **What is the formula for**

Z1 := X "Or" Y

Z2 := X "NAND" Y

Z := Z3 := Z1 "AND" Z2

Z := (X "OR" Y) "AND" (X "NAND" Y)

Z := (1.0 * X + 1.0 * Y >= 1.0) "AND" (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0)

Z := (0.5 * (1.0 * X + 1.0 * Y >= 1.0) + 0.5 * (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0) >= 1.0)

Z := (0.5 * (1.0 * X + 1.0 * Y >= 1.0) + 0.5 * (1.5 + -0.5 * X + -0.5 * Y >= 1.0) >= 1.0)

Test

- **Step 4: Please prove that your designed XOR Gate work**
  - **X=1, Y=1**
  - **X=1, Y=0**
  - **X=0, Y=1**
  - **X=0, Y=0**

Z1 := X "Or" Y

Z2 := X "NAND" Y

Z := Z3 := Z1 "AND" Z2

Z := (X "OR" Y) "AND" (X "NAND" Y)

Z := (1.0 * X + 1.0 * Y >= 1.0) "AND" (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0)

Z := (0.5 * (1.0 * X + 1.0 * Y >= 1.0) + 0.5 * (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0) >= 1.0)

Z := (0.5 * (1.0 * X + 1.0 * Y >= 1.0) + 0.5 * (1.5 + -0.5 * X + -0.5 * Y >= 1.0) >= 1.0)

Test

Take X=1 ,Y=1

Z := (0.5 * (1.0 * 1.0 + 1.0 * 1.0 >= 1.0) + 0.5 * (1.5 + -0.5 * 1.0 + -0.5 * 1.0 >= 1.0) >= 1.0)

Z := (0.5 * (1.0 + 1.0 >= 1.0) + 0.5 * (1.5 + -0.5 + -0.5 >= 1.0) >= 1.0)

Z := (0.5 * (2.0 >= 1.0) + 0.5 * (0.5 >= 1.0) >= 1.0)

Z := (0.5 * (true) + 0.5 * (false) >= 1.0)

Z := (0.5 * 1 + 0.5 * 0 >= 1.0)

Z := (0.5 + 0.0 >= 1.0)

Z := (false)

Z := 0

Test

Take X=1 ,Y=0

Z := (0.5 * (1.0 * 1.0 + 1.0 * 0.0 >= 1.0) + 0.5 * (1.5 + -0.5 * 1.0 + -0.5 * 0.0 >= 1.0) >= 1.0)

Z := (0.5 * (1.0 + 0.0 >= 1.0) + 0.5 * (1.5 + -0.5 + -0.0 >= 1.0) >= 1.0)

Z := (0.5 * (1.0 >= 1.0) + 0.5 * (1.0 >= 1.0) >= 1.0)

Z := (0.5 * (true) + 0.5 * (true) >= 1.0)

Z := (0.5 * 1 + 0.5 * 1 >= 1.0)

Z := (0.5 + 0.5 >= 1.0)

Z := (true)

Z = 1

Test

Take X=0 ,Y=1

Z := (0.5 * (1.0 * 0.0 + 1.0 * 1.0 >= 1.0) + 0.5 * (1.5 + -0.5 * 0.0 + -0.5 * 1.0 >= 1.0) >= 1.0)

Z := (0.5 * (0.0 + 1.0 >= 1.0) + 0.5 * (1.5 + -0.0 + -0.5 >= 1.0) >= 1.0)

Z := (0.5 * (1.0 >= 1.0) + 0.5 * (1.0 >= 1.0) >= 1.0)

Z := (0.5 * (true) + 0.5 * (true) >= 1.0)

Z := (0.5 * 1 + 0.5 * 1 >= 1.0)

Z := (0.5 + 0.5 >= 1.0)

Z := (true)

Z = 1

Test

Take X=0 ,Y=0

Z := (0.5 * (1.0 * 0.0 + 1.0 * 0.0 >= 1.0) + 0.5 * (1.5 + -0.5 * 0.0 + -0.5 * 0.0 >= 1.0) >= 1.0)

Z := (0.5 * (0.0 + 0.0 >= 1.0) + 0.5 * (1.5 + -0.0 + -0.0 >= 1.0) >= 1.0)

Z := (0.5 * (0.0 >= 1.0) + 0.5 * (1.5 >= 1.0) >= 1.0)

Z := (0.5 * (false) + 0.5 * (true) >= 1.0)

Z := (0.5 * 0 + 0.5 * 1 >= 1.0) Z := (0.0 + 0.5 >= 1.0)

Z := (false)

Z = 0

Test

OR **NAND** **XOR**

X Y | Z1 X Y | Z2 X Y | Z3

---------- ---------- ----------

0 0 | 0 0 0 | 1 0 0 | 0

0 1 | 1 **AND** 0 1 | 1 ==> 0 1 | 1

1 0 | 1 1 0 | 1 1 0 | 1

1 1 | 1 1 1 | 0 1 1 | 0

From Above Calculations, Hence **“OR”** AND “**NAND**” GATE Operations Output is **XOR** GATE.

Enhancement Ideas

- We can use try implementing XOR gate using different neural network architectures such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs) to see if they improve the performance of the XOR gate.

Conclusion

- Overall, the XOR problem is an important problem in neural networks because it has been used extensively to demonstrate the power and limitations of different neural network architectures and techniques, and is still relevant today as a benchmark problem for evaluating the performance of new neural network models.

GitHub Link

★ https://github.com/divyapandey03/Machine-Learning/tree/main/Logic%20Circuit%20Design/Design%20XOR%20Gate

References

★ *Dukor, S. (2020, March 07). Neural representation of and, or, not, XOR and XNOR logic gates (Perceptron algorithm). Retrieved March 30, 2023, from https://medium.com/@stanleydukor/neural-representation-of-and-or-not-xor-and-xnor-logic-gates-perceptron-algorithm-b0275375fea1*

★ *https://hc.labnet.sfbu.edu/~henry/sfbu/course/machine_learning/neural_network/slide/ann.html#gate*