CS550 - Machine Learning and Business Intelligence

Project :Design XOR Gate

Prince Rajasekaran
19658
Instructor: Dr. Chang, Henry

Table of Content

Introduction

A logical gate that exclusively ORs two input signals is known as an XOR gate (Exclusive OR gate).

Only when the two input signals are different does it output a value of 1.

In other words, the XOR gate will produce 1 if either one input is high (1) and the other is low (0), otherwise, it will produce 0.

Theory

The truth table for an XOR gate with inputs A and B and output Y can be represented as

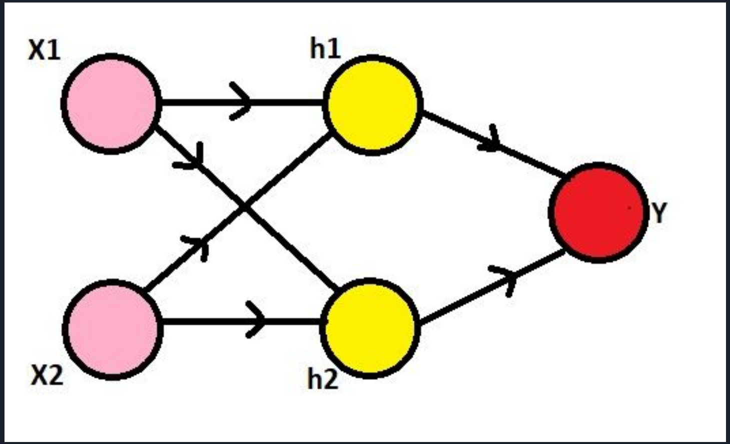
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

They are mainly used in circuit design for error detection and data processing and in encryption algorithms

Theory

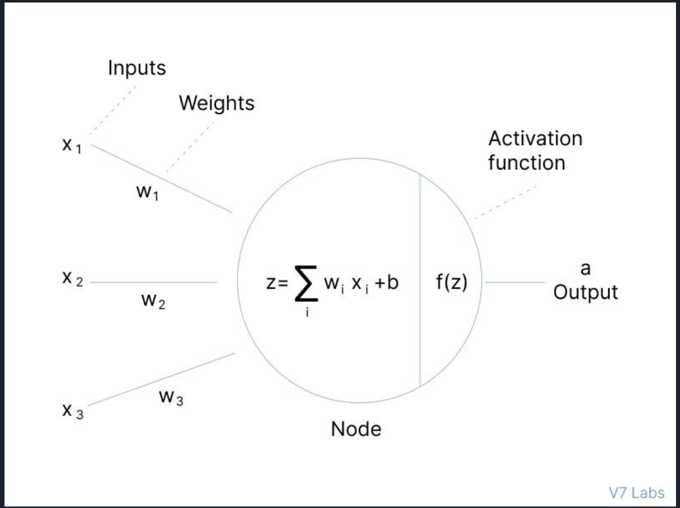
A particular kind of machine learning model called a neural network is motivated by the structure and operation of the human brain.

It is made up of many interconnected processing nodes or neurons that cooperate to address a particular issue.



Theory

Each neuron's output from a neural network is subjected to a mathematical function known as an activation function. It gives the network non-linearity, enabling it to simulate intricate, non-linear relationships between inputs and outputs.



Theory

Let’s see how to train the gates :

- Forward Pass training algorithm

```
Z := ( W0 * C + W1 * X + W2 * Y >= T )
    where T := 1.0

if ( W0 * C + W1 * X + W2 * Y >= T )
then output is 1
else output = 0
```

Theory

Training the gates :

Set the input (X,Y) to some possible value such as (0,0), (0,1), (1,0), or (1,1).

Forward process :

Calculate the output Z for the given input (X,Y).

Backward process :

1. Adjust the weight
2. If the output Z is too low, increase the weights that had inputs that were "1"
3. If the output Z is too high, decrease the weights that had inputs that were "1"
4. Continue looping this process till each possible input combination gives the right output

Implementation

Project: Design XOR Gate

```
      +----------------------------------------------------------------+
      |     "NAND"    +--------+                     "XOR" Function    |
      | 1.0-->1.0--->|        |                                       |
      |              |        |                                       |
X --->|----> ???? -->| Neuron |-->Z2--                                |
      |\             |        |      \                                |
Y---->|-\--> ???? -->|        |       \ "AND"    +--------+           |
      |\ \           +--------+        ->????-->|        |           |
      | \ \                                     | Neuron |-->Z3-->|-->Z
      |  \ \  "Or"   +--------+       -->????-->|        |           |
      |   \ -->???-->|        |      /          +--------+           |
      |    \         | Neuron |-->Z1--                                |
      |     -->???-->|        |                                       |
      |              +--------+                                       |
      +----------------------------------------------------------------+
```

Implementation

Step 1 : design your own AND gate, OR gate and NAND gate using these rules

Using the forward/backward process :

- Forward process
  Calculate the output Z for the given input (X,Y).
- Backward process
  Adjust weights
  + If the output Z is too low, increase the weights by 0.5
    which had inputs that were "1".
  + If the output Z is too high, decrease the weights by 0.5

Using the step activation function :

Z := (W1 * X + W2 * Y >= T)
   where T := 1.0

Implementation

1. **AND**

Desired "And"

Function

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

Loop 1
W1=W2=0

Function

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

Loop 2
W1=W2=0.5

Function

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

We got the desired result here in the loop 1

Hence the formula will be:

Z := (0.5 * X + 0.5 * Y >= 1)

Implementation

1. OR

Desired "OR" Function

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

Loop 1
W1=W2=0

Function

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

Loop 2
W1=W2=0.5

Function

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

Loop 3
W1=W2=1.0

Function

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

We got the desired result here in the loop 3

Hence the formula will be:

Z := (1.0 * X + 1.0 * Y >= 1)

Implementation

Desired
"NAND"

Function
C X Y | Z
----
1 0 0 | 1
1 0 1 | 1
1 1 0 | 1
1 1 1 | 0

Loop 1
W0 =0.0
W1=W2=0.5

Function
C X Y | Z
----
1 0 0 | 0
1 0 1 | 0
1 1 0 | 0
1 1 1 | 1

Loop 2
W0 =0.5
W1=W2=0.5

Function
C X Y | Z
----
1 0 0 | 0
1 0 1 | 1
1 1 0 | 1
1 1 1 | 1

Loop 3
W0 =1.0
W1=W2=0.5

Function
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
Function

C X Y | Z
----
1 0 0 | 1
1 0 1 | 1
1 1 0 | 1
1 1 1 | 1

Loop 5
W0 =1.0
W1= -0.5,
W2= 0.0

Function
C X Y | Z
----
1 0 0 | 1
1 0 1 | 1
1 1 0 | 0
1 1 1 | 0

Loop 6
W0 = 1.0
W1= 0.0,
W2=0.0

Function
C X Y | Z
----
1 0 0 | 1
1 0 1 | 1
1 1 0 | 1
1 1 1 | 1

Loop 7
W0 =1.0
W1=0.0, W2=-0.5

Function
C X Y | Z
----
1 0 0 | 1
1 0 1 | 0
1 1 0 | 1
1 1 1 | 0

Implementation

Loop 8
W0 = 1.5
W1= 0.0, W2=
-0.5
Function

C X Y | Z
----
1 0 0 | 1
1 0 1 | 1
1 1 0 | 1
1 1 1 | 1

Loop 9
W0 =1.5
W1=-0.5,
W2=-0.5

Function
C X Y | Z
----
1 0 0 | 1
1 0 1 | 1
1 1 0 | 1
1 1 1 | 0

Implementation

The formula for

"AND"

Z := (0.5 * X + 0.5 * Y >= 1)

"OR"

Z := (1.0 * X + 1.0 * Y >= 1)

"NAND"

Z := (1.5 * C -0.5 * X -0.5* Y >= 1)

Implementation

Proving that the designed XOR Gate works :

- **X=1, Y=1**
- **X=1, Y=0**
- **X=0, Y=1**
- **X=0, Y=0**

Test 1 , X=1, Y=1	Desired Result
Z1: = (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0) = 1.5-0.5-0.5 := 0 Z2: = (1* X + 1* Y >= 1.0) = 1 + 1 := 1 Z3: = (0.5* Z1 + 0.5* Z2 >= 1.0) = 0.5*0 + 0.5*1 := 0	XOR ----------- X Y | Z ----------- 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

Implementation

Test 2: X=1, Y=0	Desired Result
Z1: = (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0) = 1.5-0.5*1-0.5*0 := 1 Z2: = (1* X + 1* Y >= 1.0) = 1*1 + 1*0 := 1 Z3: = (0.5* Z1 + 0.5* Z2 >= 1.0) = 0.5*1 + 0.5*1 := 1	XOR ----------- X Y | Z ----------- 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

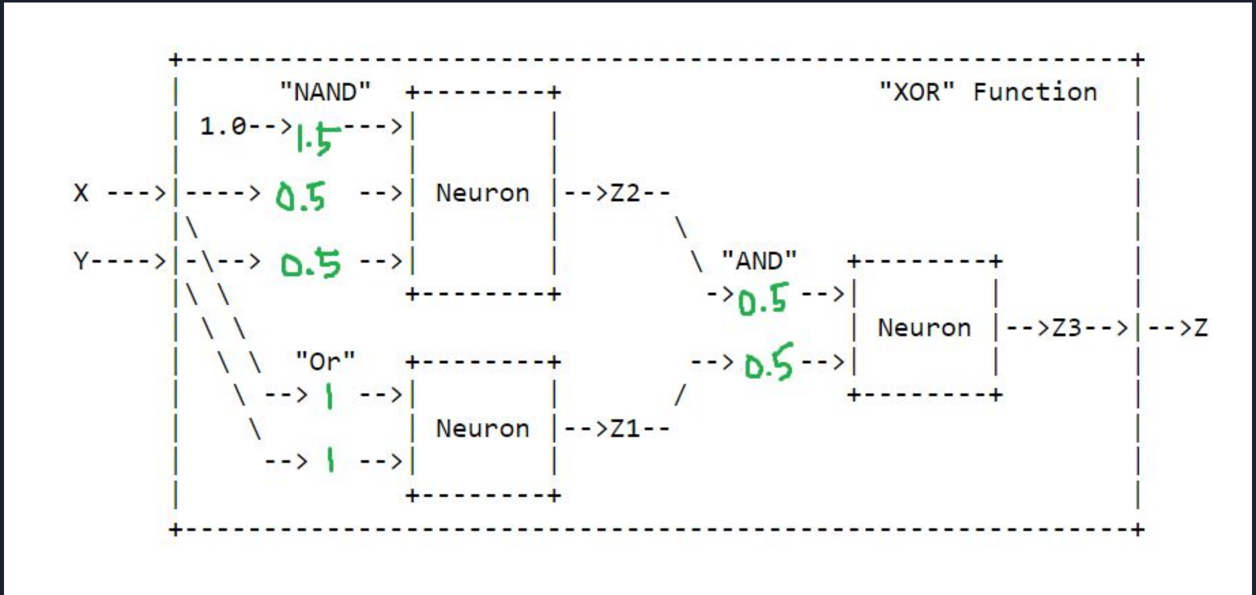
Implementation

Test 3: **X=0, Y=1**	**Desired Result**
Z1: = (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0) = 1.5*1.0-0.5*0-0.5*1 := 1 Z2: = (1* X + 1* Y >= 1.0) = 1*0 + 1*1 := 1 Z3: = (0.5* Z1 + 0.5* Z2 >= 1.0) = 0.5*1 + 0.5*1 := 1	**XOR** ----------- **X Y | Z** ----------- **0 0 | 0** **0 1 | 1** **1 0 | 1** **1 1 | 0**

Implementation

Test 4: **X=0, Y=0**	**Desired Result**
Z1: = (1.5 * 1.0 + -0.5 * X + -0.5 * Y >= 1.0) = 1.5-0.5*0-0.5*0 := 1 Z2: = (1* X + 1* Y >= 1.0) = 1*0 + 1*0 := 0 Z3: = (0.5* Z1 + 0.5* Z2 >= 1.0) = 0.5*1 + 0.5*0 := 0	XOR ----------- X Y | Z ----------- 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

Conclusion:


"NAND"
"XOR" Function
1.0-->
1.5
X --->
0.5
Y---->
0.5
Neuron
-->Z2--
"AND"
0.5
0.5
Neuron
-->Z3-->
-->Z
"Or"
1
1
Neuron
-->Z1--

Conclusion

The overall goal of the project is to implement the Forward and Backward Pass Algorithm and design an XOR gate using a neural network. As a result, we now have the fundamental procedures for carrying out the Forward and Backward process with the activation function.

Along with designing logic gateways with neural networks, we also have a variety of other applications, such as robotics, natural language processing, image and speech recognition, and more.

References

DeepAI. (2019, May 17). *Feed Forward Neural Network*. DeepAI. Retrieved April 3, 2023, from https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network

Wright, L. G., Onodera, T., Stein, M. M., Wang, T., Schachter, D. T., Hu, Z., & McMahon, P. L. (2022, January 26). *Deep physical neural networks trained with backpropagation*. Nature News. Retrieved April 3, 2023, from https://www.nature.com/articles/s41586-021-04223-6

What is an artificial neural network? here's everything you need to know. Digital Trends. (2019, January 6). Retrieved April 3, 2023, from https://www.digitaltrends.com/computing/what-is-an-artificial-neural-network/