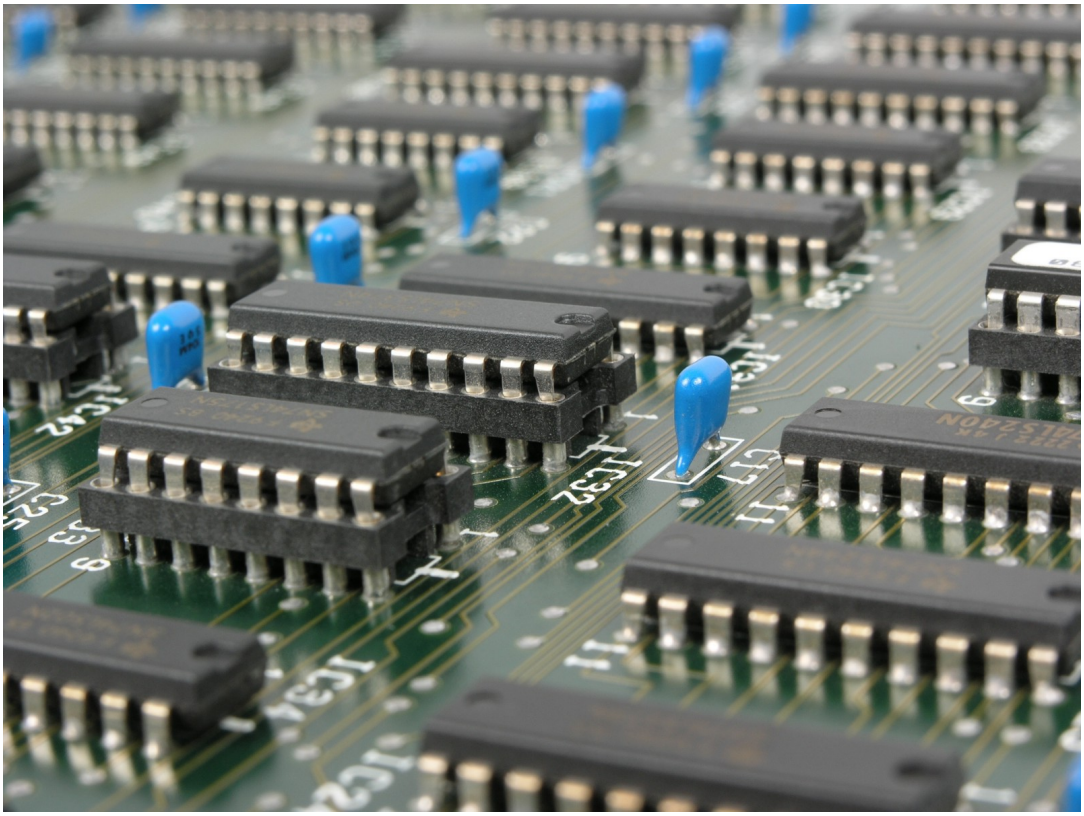
Francisc
Camillo
Past tangent, 3D graphic artist, IoT enthusiast, Data Science Newbie
Jul 8, 2017 · 4 min read

Neural Representation of Logic Gates



https://static.pexels.com/photos/39290/mother-board-electronics-computer-board-39290.jpeg

Logic gates namely AND, OR, NOT are some of the building blocks of every technological breakthrough for the past decade specially for hardware.

For this project, we are going to represent Logic Gates using the basics of Neural Network. I've created a perceptron using numpy that implements this Logic Gates with the dataset acting as the input to the perceptron.

Part 1: Logic Gates

First, we must familiarize ourselves about logic gates.

'***A logic gate is an elementary building block of a digital circuit. Most logic gates have two inputs and one output. At any given moment, every terminal is in one of the two binary conditions low (0) or high (1), represented by different voltage levels. The logic state of a terminal can, and generally does, change often, as the circuit processes data. In most logic gates, the low state is approximately zero***

volts (0 V), while the high state is approximately five volts positive (+5 V)*. as stated from techtarget*

The most common logic gate are AND, OR, NOT. The logic gate AND only returns 1 if both inputs are 1 else 0, logic gate OR returns 1 for all inputs with 1, and will only return 0 if both input is 0 and lastly logic gate NOT returns the invert of the input, if the input is 0 it returns 1, if the input is 1 it returns 0. To make it clear the image below shows the truth table for the basic gates.

A	B	Z
0	0	0
0	1	0
1	0	0
1	1	1

AND

A	B	Z
0	0	0
0	1	1
1	0	1
1	1	1

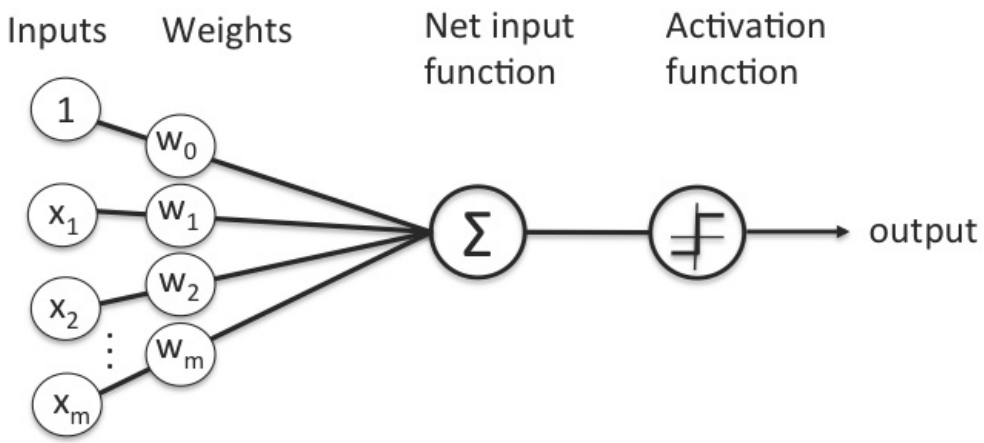
OR

A	Z
0	1
1	0

NOT

The columns A and B are the inputs and column Z is the output. So, for the inputs A = 0, B = 0 the output is Z = 0.

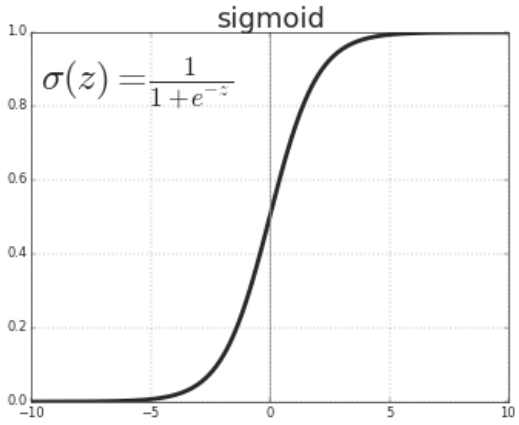
Part 2: Perceptron



Schematic of Rosenblatt's perceptron.

A perceptron is the basic part of a neural network. A perceptron represents a single neuron on a human's brain, it is composed of the dataset (Xm) , the weights (Wm) and an activation function, that will then produce an output and a bias. The datasets (inputs) are converted into an ndarray which is then matrix multiplied to another ndarray that holds the weights. Summing up all matrix multipy and adding a bias will create the net input function, the output would then passed into an activation function that would determine if the neuron needs to fire an output or not. Most common activation function used for classification used is a sigmoid function, which is a great function for classification (Although sigmoid is not the leading activation function for middle layers of neural networks [ahem ReLU / Leaky ReLU] it still is widely used for final

neural networks [eheM ReLU / Leaky ReLU] it still is widely used for final classifications.)



The image above is the curve for the sigmoid function. The output of the model is passed into the sigmoid as “z”. To make it simple a sigmoid function returns 0–0.4 if the result of a model (z) is a negative number and 0.5–1 if the model is positive.

3. Code

For the code, we’ll start off by importing numpy. Numpy is a library with built in mathematical functions that is great for doing matrix multiplies and scientific programming. (Learn more about numpy here: http://www.numpy.org/)

```
1  import numpy as np
```

gistfile1.txt hosted with ❤ by **GitHub** **view raw**

Then we’ll create a perceptron function that will act as the perceptron in the image shown before. The function will take in a weight matrix, the bias and the dataset matrix. “np.dot(x, weight)” matrix multiplies the dataset and the weights from the input and then “np.add()” adds the output of the matrix multiply to the bias. “1/(1+np.exp(-model)” represents the activation function, passing the model into a sigmoid function.

Note: Since our goal for this project is to represent a perceptron as a logic gate, we are going to round the output of the activation function to make the output only 1 or 0, but for actual purposes, rounding the output is a big NO, the small information given by the decimals helps add information for the next neuron that will handle the information. Not to mention about Vanishing and Exploding Gradient, but that’s another story

```
def perceptron(weight, bias, x):
    model = np.add(np.dot(x, weight), bias)
    logit = 1/(1+np.exp(-model))
    print('Type: {}'.format(logit))
    return np.round(logit)
```

Logic.py hosted with ❤ by GitHub view raw

After creating the perceptron we need to fill in the inputs for it. The function compute does it, since the dataset and weights aren't ndarrays yet we'll do it here. It takes in the logictype ("logic_and", "logic_or", etc) for labeling the compute being done by the machine, weightdict, a dictionary that holds all the weights and biases, and lastly the dataset parameter takes in the dataset

```
def compute(logictype, weightdict, dataset):
    weights = np.array([ weightdict[logictype][w] for w in weightdict[logictype].keys()[::-1
    output = np.array([ perceptron(weights, weightdict['bias'][logictype], val) for val in d
    print(logictype)
    return logictype, output
```

Logic.py hosted with ❤ by GitHub view raw

The compute function will return the result of the perceptron function for each dataset in an array

The gist below is a sample output of the Logic gate AND and the Logic gate OR.

```
Logic Function: AND
X0 X1 X2 Y
1   0   0   0.0
1   0   1   0.0
1   1   0   0.0
1   1   1   1.0
Logic Function: OR
X0 X1 X2 Y
1   0   0   0.0
1   0   1   1.0
1   1   0   1.0
```

That concludes the Neural Logic Project. Some notes, the projects weights has been made manually for the sake of introducing the basic function of a perceptron, although optimization would be the best answer to find the correct weights for this problem, so that the neural network could correctly answer the problem if the inputs becomes larger.

To Learn more you could visit the repository : https://github.com/fjcamillo/Neural-Representation-of-Logic-Functions/blob/master/Logic.py

Machine Learning | Neural Network | Logic Gates | Perceptron