Perceptrons, Logical Functions, and the XOR problem

Deep Learning Pills #2

Francesco Cicala · Follow

Published in Towards Data Science

6 min read · Aug 31, 2018

Today we will explore what a Perceptron can do, what are its limitations, and we will prepare the ground to overreach these limits! Everything supported by graphs and code.

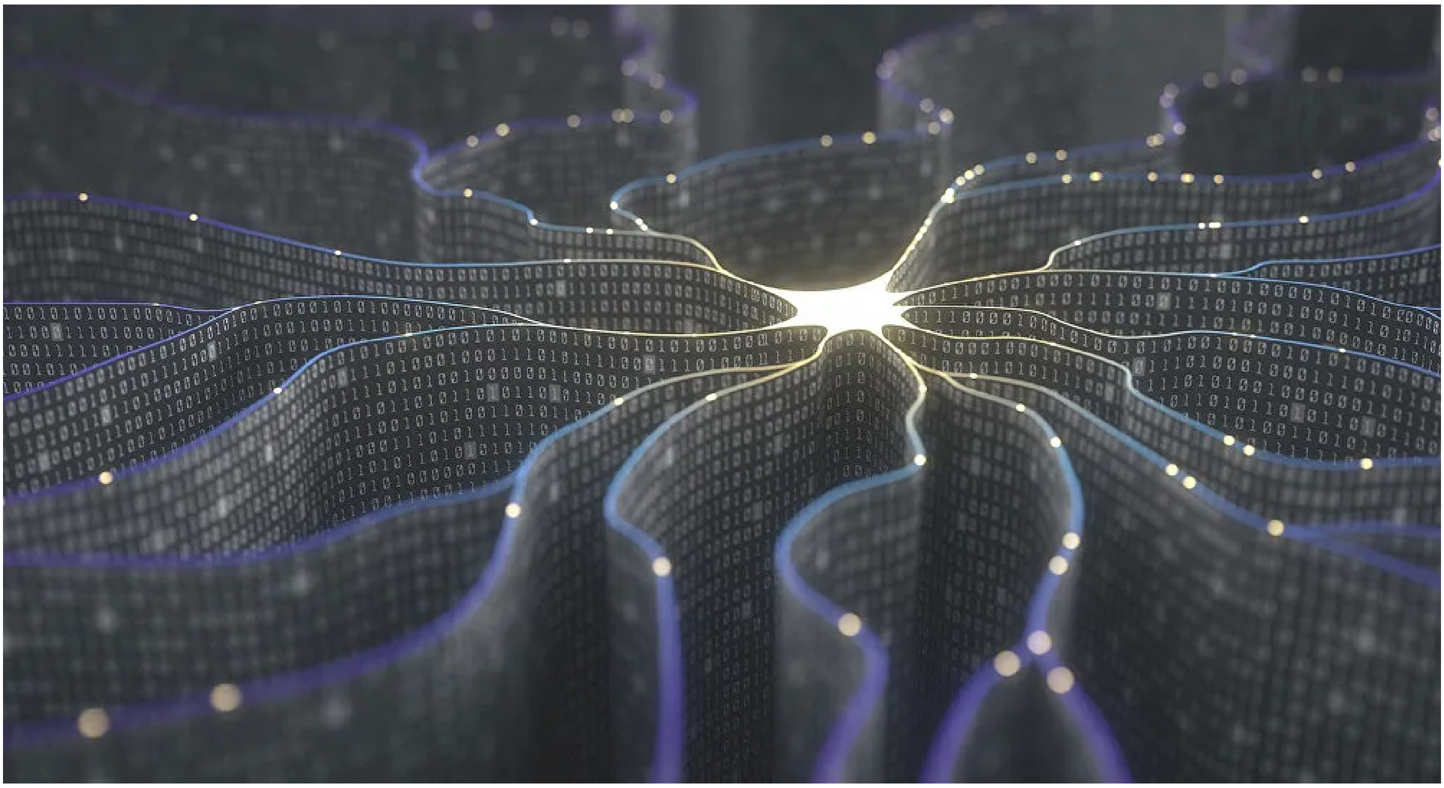
Elements from Deep Learning Pills #1

In Part 1 of this series, we introduced the Perceptron as a model that implements the following function:

$$\begin{aligned}\hat{y} &= \Theta(w_1 x_1 + w_2 x_2 + \ldots + w_n x_n + b) \\ &= \Theta(\mathbf{w} \cdot \mathbf{x} + b)\end{aligned}$$

$$\text{where} \quad \Theta(v) = \begin{cases} 1 & \text{if } v \geqslant 0 \\ 0 & \text{otherwise} \end{cases}$$

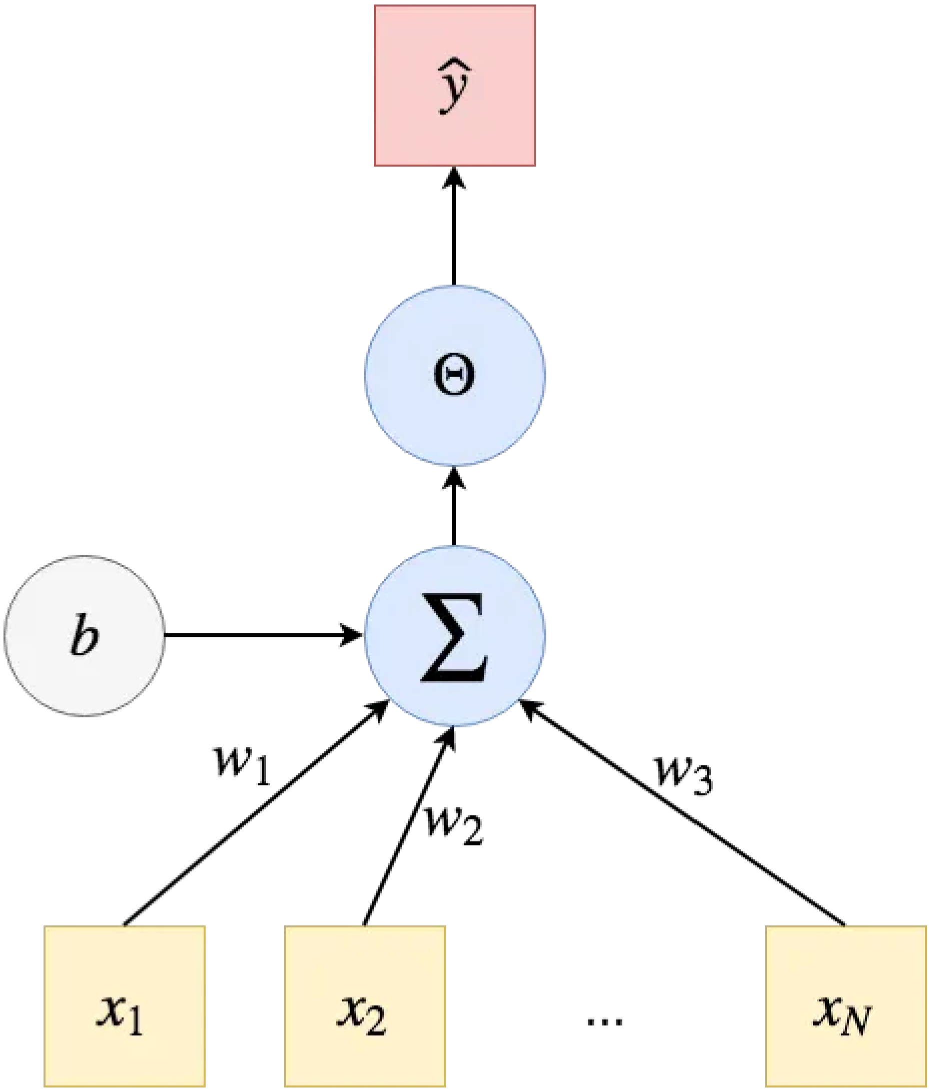
For a particular choice of the parameters *w* and *b*, the output ŷ only depends on the input vector *x*. I'm using ŷ ("y hat") to indicate that this number has been produced/predicted by the model. Soon, you will appreciate the ease of this notation.



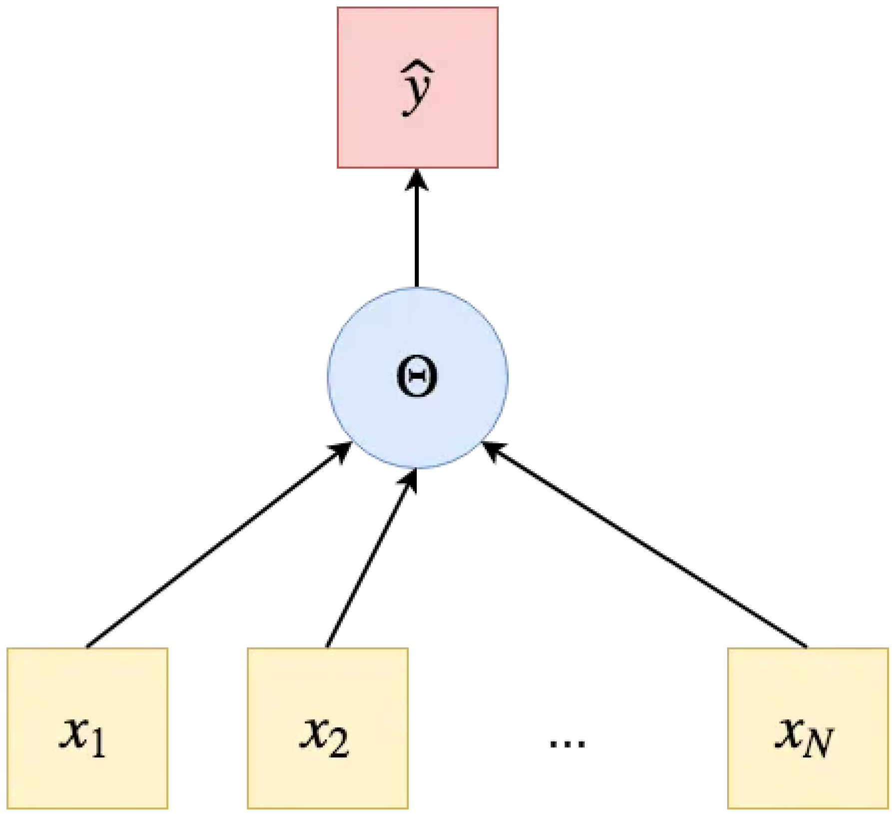
Computational Graph

To visualize the **architecture** of a model, we use what is called **computational graph**: a directed graph which is used to represent a math function. Both variables and operations are nodes; variables are fed into operations and operations produce variables.

The computational graph of our perceptron is:



The Σ symbol represents the linear combination of the inputs x by means of the weights w and the bias b. Since this notation is quite heavy, from now on I will simplify the computational graph in the following way:



What can a perceptron do?

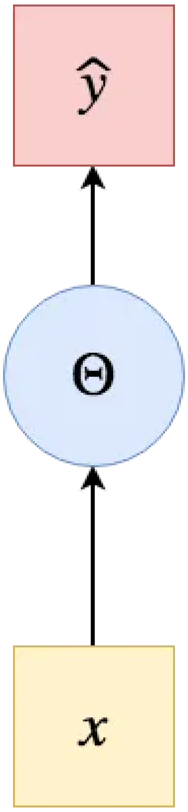
I am introducing some examples of what a perceptron can implement with its *capacity* (I will talk about this term in the following parts of this series!). Logical functions are a great starting point since they will bring us to a natural development of the theory behind the perceptron and, as a consequence, **neural networks**.

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

NOT logical function

Let's start with a very simple problem:

> *Can a perceptron implement the NOT logical function?*



NOT(x) is a 1-variable function, that means that we will have one input at a time: N=1. Also, it is a **logical function**, and so both the input and the output have only two possible states: 0 and 1 (i.e., False and True): the Heaviside step function seems to fit our case since it produces a binary output.

With these considerations in mind, we can tell that, if there exists a perceptron which can implement the NOT(x) function, it would be like the one shown at left. Given two parameters, *w* and *b*, it will perform the following computation:
$\hat{y} = \Theta(wx + b)$

The fundamental question is: do exist two values that, if picked as parameters, allow the perceptron to implement the NOT logical function? When I say that a *perceptron implements a function,* I mean that for each input in the function's domain the perceptron returns the same number (or vector) the function would return for the same input.
Back to our question: those values exist since we can easily find them: let's pick $w = -1$ and $b = 0.5$.

```python
import numpy as np

def unit_step(v):
        """ Heavyside Step function. v must be a scalar """
        if v >= 0:
                return 1
        else:
                return 0

def perceptron(x, w, b):
    """ Function implemented by a perceptron with
                weight vector w and bias b """
        v = np.dot(w, x) + b
        y = unit_step(v)
        return y

def NOT_percep(x):
        return perceptron(x, w=-1, b=0.5)

print("NOT(0) = {}".format(NOT_percep(0)))
print("NOT(1) = {}".format(NOT_percep(1)))
```


And we get:

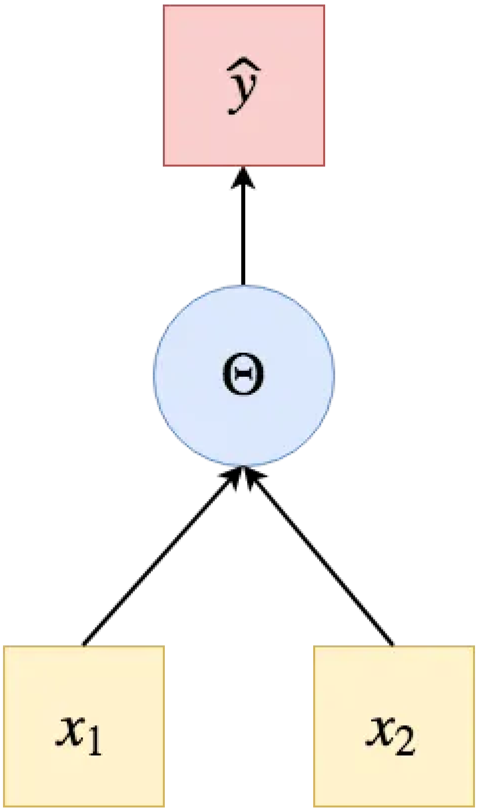
```
NOT(0) = 1
NOT(1) = 0
```

We conclude that the answer to the initial question is: yes, a perceptron can implement the NOT logical function; we just need to **properly set its parameters.** Notice that my solution isn't unique; in fact, solutions, intended as (w, b) points, are infinite for this particular problem! You can use your favorite one ;)

AND logical function

The next question is:

> *Can a perceptron implement the AND logical function?*



The AND logical function is a 2-variables function, *AND(x1, x2)*, with binary inputs and output.

This graph is associated with the following computation:
$\hat{y} = \Theta(w1*x1 + w2*x2 + b)$

This time, we have three parameters: *w1*, *w2,* and *b*.
Can you guess which are three values for these parameters which would allow the perceptron to solve the **AND problem**?

SOLUTION:
w1 = 1, w2 = 1, b = -1.5

```python
def AND_percep(x):
    w = np.array([1, 1])
    b = -1.5
    return perceptron(x, w, b)

# Test
example1 = np.array([1, 1])
example2 = np.array([1, 0])
example3 = np.array([0, 1])
example4 = np.array([0, 0])

print("AND({}, {}) = {}".format(1, 1, AND_percep(example1)))
print("AND({}, {}) = {}".format(1, 0, AND_percep(example2)))
print("AND({}, {}) = {}".format(0, 1, AND_percep(example3)))
print("AND({}, {}) = {}".format(0, 0, AND_percep(example4)))
```


And it prints:

```
AND(1, 1) = 1
AND(1, 0) = 0
AND(0, 1) = 0
AND(0, 0) = 0
```

OR logical function

OR(x1, x2) is a 2-variables function too, and its output is 1-dimensional (i.e., one number) and has two possible states (0 or 1). Therefore, we will use a perceptron with the same architecture as the one before. Which are the three parameters which solve the OR problem?

SOLUTION:
$w1 = 1, w2 = 1, b = -0.5$

```python
def OR_percep(x):
    w = np.array([1, 1])
    b = -0.5
    return perceptron(x, w, b)

# Test
example1 = np.array([1, 1])
example2 = np.array([1, 0])
example3 = np.array([0, 1])
example4 = np.array([0, 0])

print("OR({}, {}) = {}".format(1, 1, OR_percep(example1)))
print("OR({}, {}) = {}".format(1, 0, OR_percep(example2)))
print("OR({}, {}) = {}".format(0, 1, OR_percep(example3)))
print("OR({}, {}) = {}".format(0, 0, OR_percep(example4)))
```

or.py hosted with ❤ by GitHub view raw

```
OR(1, 1) = 1
OR(1, 0) = 1
OR(0, 1) = 1
OR(0, 0) = 0
```

XOR — ALL (perceptrons) FOR ONE (logical function)

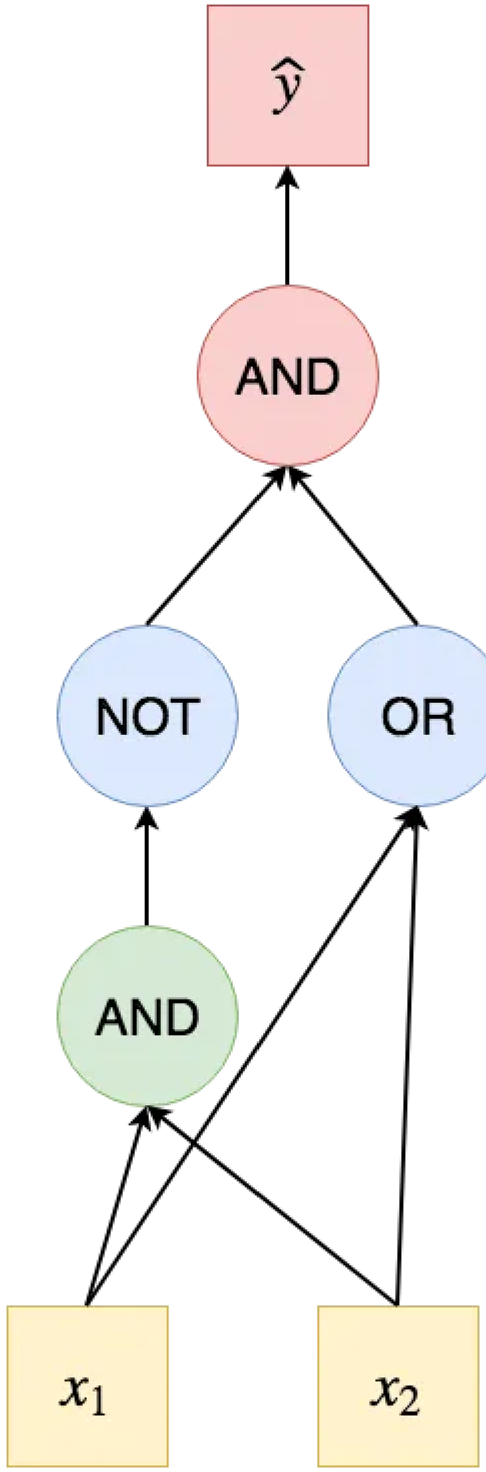
We conclude that a single perceptron with an Heaviside activation function can implement each one of the fundamental logical functions: NOT, AND and OR. They are called *fundamental* because any logical function, no matter how complex, can be obtained by a combination of those three. We can infer that, **if we appropriately connect the three perceptrons we just built, we can implement any logical function!** Let's see how:

> *How can we build a network of* ***fundamental logical perceptrons*** *so that it implements the XOR function?*

A	B	XOR
0	0	0
0	1	1
1	0	1
1	1	0

SOLUTION:

$$XOR(x1, x2) = AND(NOT(AND(x1, x2)), OR(x1, x2))$$



```
def XOR_net(x):
    gate_1 = AND_percep(x)
    gate_2 = NOT_percep(gate_1)
    gate_3 = OR_percep(x)
    new_x = np.array([gate_2, gate_3])
    output = AND_percep(new_x)
    return output

print("XOR({}, {}) = {}".format(1, 1, XOR_net(example1)))
print("XOR({}, {}) = {}".format(1, 0, XOR_net(example2)))
print("XOR({}, {}) = {}".format(0, 1, XOR_net(example3)))
print("XOR({}, {}) = {}".format(0, 0, XOR_net(example4)))
```


And the output is:

```
XOR(1, 1) = 0
XOR(1, 0) = 1
XOR(0, 1) = 1
XOR(0, 0) = 0
```


Some of you may be wondering if, as we did for the previous functions, it is possible to find parameters' values for a single perceptron so that it solves the XOR problem all by itself.

I won't make you struggle too much looking for those three numbers, because it would be useless: the answer is that they do not exist. **Why?** The answer is that the XOR problem is not **linearly separable**, and we will discuss it in depth in the next chapter of this series!

I will publish it in a few days, and we will go through the linear separability property I just mentioned. I will reshape the topics I introduced today within a geometrical perspective. In this way, every result we obtained today will get its natural and intuitive explanation.

If you liked this article, I hope you'll consider to give it some claps! Every clap is a great encouragement to me :) Also, feel free to get in touch with me on Linkedin!