Building a Neural Network from Scratch

By Renzo Honorato

I spent the last week trying to build a Neural Network all by myself, without libraries that make for you. Mostly because I wanted a better understanding of how they worked, as I didn't really get the high level implementations.

You don't have to skim through this article to decide whether you wanna read it or not. Here's what you'll learn:

1. A solid mathematical overview of how Neural Networks work.
2. How to implement a simple NN to add two bits.
3. What the heck is going on under the hood when you use a Neural Network.

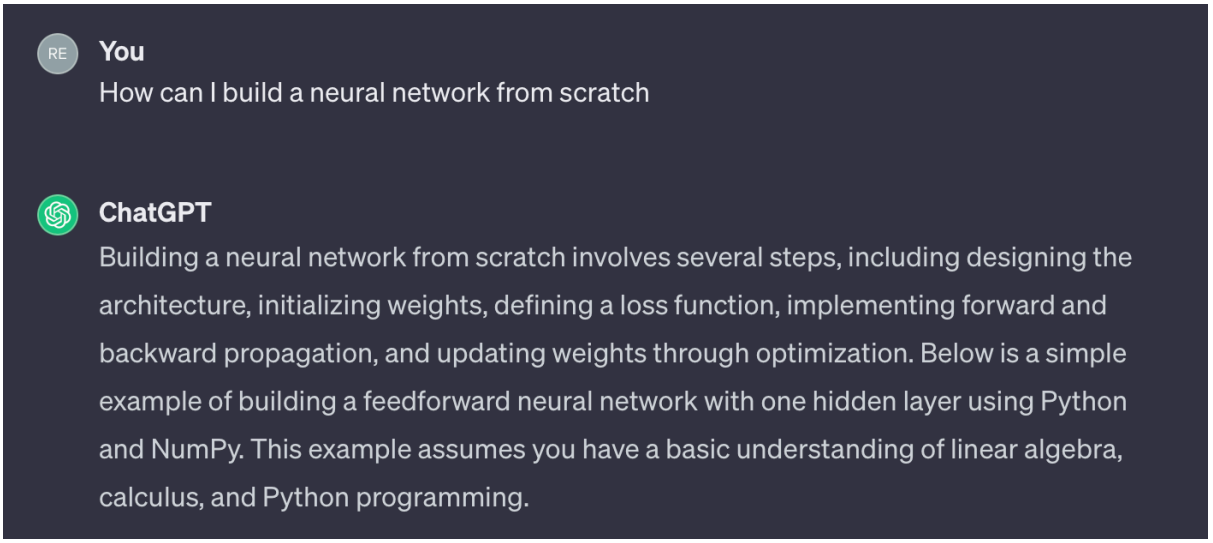
Disclaimer:

I will only show the code for simple examples for using NNs to solve simple problems, to really create a deeper understanding of how they work. At some point this whole article series will be in my github account, so follow to learn a lot more in the next issues!

The Beginning

My journey began where most good stories begin: I was lost and hopeless. I had spent the last several weeks trying to learn AI to do all those cool things (by "cool things" I really mean "classification problems", and yes, that is actually really cool, right?), but I just didn't get what was going on and no one actually got to the actual **low level implementation**!

So, inspired by **this 3B1B video** (which I highly recommend, by the way) I decided that I would build my own neural network to classify numbers. And what better place to start than ChatGPT?



The first prompt I made to ChatGPT to start building my NN.

And do I regret that prompt? That is the beginning of many sleepless nights and days into a story of perceptrons and errors(many, many errors). “no pain, no gain”, I guess.

After that I started following **the book introducing Neural Networks by Michael Nielsen.** The beginning was a huge challenge. I just couldn’t catch up and actually take the mathematical concepts and implement them on my own. Which is why I have boiled all the main concepts down below to help you.

Here is all you need to know to start:

A Neural Network is made out of many layers of **perceptrons,** which, for our purposes, you can think of as just a number. A simple binary perceptron works like this:

- You feed it n inputs, but each input has its own “level of importance” (i.e. weights) that will determine whether or not the perceptron is “on”, if the sum of the inputs multiplied by their weights is less than a certain number b, then the perceptron is off, if the sum is larger than b then the perceptron is on, simple as that!

Using that, let’s try to solve a simple problem:

Suppose you’re tired of deciding whether or not you want to go out every time. So you decide to make a Neural Network that will decide that for you. It takes 3 inputs:

1. Is the weather good?
2. Is it near?
3. Does my partner wanna go with me?

And imagine that you really really want to have your boyfriend/girlfriend go with you, so you assign it weight 5. Also, you don't have much money to pay for public transportation and it's not convenient, so you assign it weight 2. And you also give the weather a weight 2 because you have a good umbrella.

If the sum is less than 5 you don't go and the program spits out "No", if it is more than or equal to 5 you go and the program spits out "Yes".

So, in this example we have 1 neuron that takes 3 inputs.

Here is a simple python script that implements that:

```
# This function takes only 1s and 0s as arguments!
def do_i_want_to_go_out(good_weather, near, my_partner_is_going):
  result = my_partner_is_going*5 + near*2 + good_weather*2
  return  "Yes" if result >= 5 else "No"
```

I know this may seem a little dumb! But it is the building block for more complex things.

Now that you know what a perceptron is, let's try to build something a bit more useful with it.

A NAND gate is a logical gate that takes 2 inputs and fires only if both are no "on", here is truth table for a NAND gate:

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

Truth table for a NAND gate.

To create a NAND gate we will create a class called “binary_perceptron” that will follow this rule to output either a 0 or a 1:

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Let's start by defining our perceptron class:

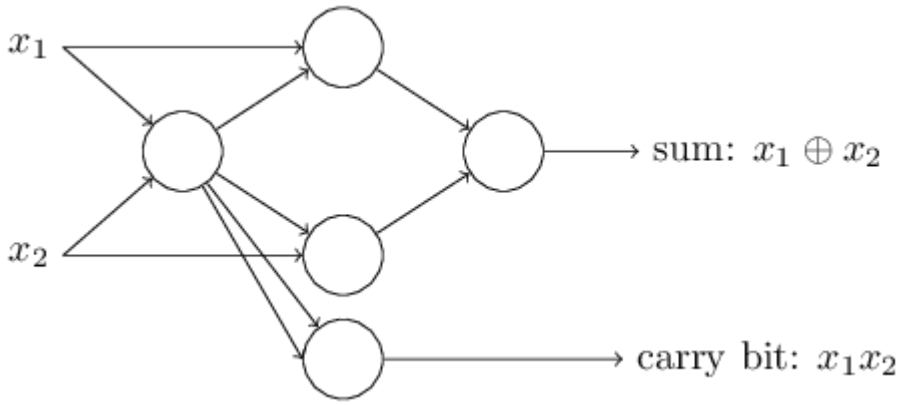
```
class Perceptron:
  def __init__(self, weights, bias):
    self.weights = weights
    self.bias = bias

  def activation(self, inputs):
    weighted_sum = np.dot(inputs, self.weights) + self.bias
    return 1 if weighted_sum > 0 else 0
```

Notice that this is just a generic binary perceptron. It isn't acting as a NAND gate yet. Let's create a nand_gate function now:

```
def nand_gate(x1, x2):
  perceptron = Perceptron([-2, -2], 3)
  return perceptron.activation([x1, x2])
```

This function acts exactly as a band_gate. Using that we can create a neural network that adds two bits!

To do that, we will use the following architecture that keeps track of a carry bit and result of the sum:



To implement that, we are just going to call the nand_gate function and feed some inputs. The code might be a little confusing, but follow the diagram above and it will make sense!

```
# This function receives 2 bits and returns their and a carry bit.
def bitwise_adder(x1, x2):
  gate1 = nand_gate(x1, x2)
```

```
    gate2 = nand_gate(x1, gate1)
    gate3 = nand_gate(x2, gate1)
    result = nand_gate(gate2, gate3)
    carry_bit = nand_gate(gate1, gate1)

    return [result, carry_bit]
```

This is already getting too long, so I'll leave the rest for next week!

But before you go, I want you to notice that from NAND gates you can perform any computational operation, meaning that from perceptrons you can also perform any kind of computational operation! **That is the power of AI**, the only challenge is helping it figure out how to adjust the numbers in the neural net in order to solve the problem.

But that is a story for another day.

For now, that's it, but before you leave you should read this:

1. If you want to go further in your ML knowledge, **stay tuned for the next articles,** there I will go much more in depth in the technology.
2. Follow me on X for uncomplicated news on tech, education and economics: @renzo_honorato
3. Connect with me on LinkedIn so we can chat! I would love to learn more about you.
4. Last but not least, if you liked this content, like and leave a comment with your thoughts!

Thanks for getting to this point and I'll see you next week! :)