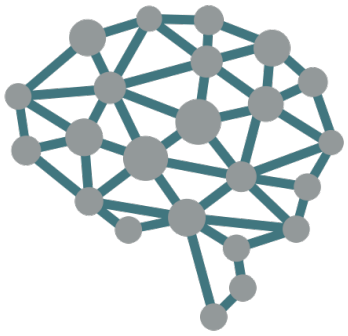
B5 - Computer Numerical Analysis

B-CNA-500

MyTorch Bootstrap

an introduction to artificial neural networks



In 1948, in a paper named Intelligent Machinery, Alan TURING introduced a type of neural networks named B-type unorganized machine that he considered as the simplest possible model of the nervous system.

The objectives of this bootstrap are:

- to give you a first approach to artificial neural networks ;
- to have you implement the algorithms at the heart of the training process.

Libraries that handles the implementation of artificial neural networks are forbidden.
You must code everything from scratch

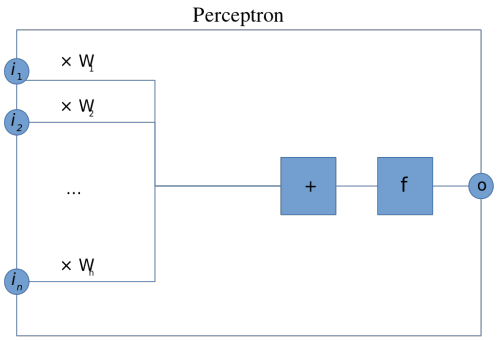
You can use libraries to handle linear algebra and to display the learning curves.

THE PERCEPTRON

Invented in 1943, by Warren McCULLOCH and Walter PITTS, the perceptron is a **single layer linear classifier**.

The first hardware implementation was built in 1957 by Frank ROSENBLATT.
It was one of the first neural network to be implemented.

The perceptron mimics a neuron inside a human brain and provides a single output.
It consists in a single node. Its activation depends on the values of its **inputs** and its **activation function**.
Each connection contains a weight (also called parameter) which is learned during the **training phase**.



$$o = f\left(\sum_{k=1}^{n} i_k \cdot W_k\right)$$

For this part, your goal is to create a perceptron, that:

- takes an arbitrary number of inputs,
- produces a single output (1 or 0) on the standard output.

Terminal

```
~/B-CNA-500> ./my_perceptron --help
USAGE
    ./my_perceptron [--new NB_INPUTS | --load LOADFILE] [--save SAVEFILE] --mode
[train | predict] FILE

DESCRIPTION
    --new       Creates a new perceptron with NB_INPUTS inputs.
    --load      Loads an existing perceptron from LOADFILE.
    --save      Save the perceptron's state into SAVEFILE. If not provided, the
state of the perceptron will be displayed on standard output.
    FILE      a file containing a list of inputs (and expected outputs) that the
perceptron needs to evaluate (either for training, or predicting).
```

Step I: The basics

A perceptron is a simple linear classifier. It is only defined by:

- its inputs
- its biases
- its activation function
- its learning rate

Basically, that means that your perceptron will just make a weighted sum of all the inputs, and send the result to its activation function. If the result is above a certain threshold, the output will be 1, otherwise it will be 0.
If some of these concepts are a bit vague, try to do some research before going further !

For this first part, create a program that can generate a new perceptron with random biases. You must be able to store its state into a save file.

Step II: Your first perceptron

Generate a perceptron that can take 2 inputs, and mimics the behaviour of an AND gate.

Obviously, this problem could easily be solved programatically. You need to make your perceptron "*learn*" how to mimic an AND gate

First, you need to generate your training data. As there are only two inputs that can each take only 2 values (0 or 1), there are only 4 cases to handle. You can make a script that can generate 2 random booleans (0 or 1), send them to your perceptron, and check if the output is correct. Easy !

Remember the truth tables ?

If you got lucky and generated a perceptron that already computes an AND gate perfectly without training, try to make it evaluate a OR gate instead !

At this stage, your perceptron should be able to be sometimes right, sometimes wrong. You can compute the "error" as the difference between the expected output and the real output.
Of course, in order to make it learn and get more efficient, you need to update your biases depending on whether the output is correct or not.
Your perceptron will need a way to correct its biases depending on the computed error. Therefore, you need to implement a **learning algorithm**.

You may want to research what is a *stochastic gradient descent*

Step III: Automation

As the previous example was pretty simple, a lot of steps could be done by hand. In order to facilitate debugging and learning, try to automate all this process using scripts !

You could also consider visualizing any data that might be useful using a plotting library.

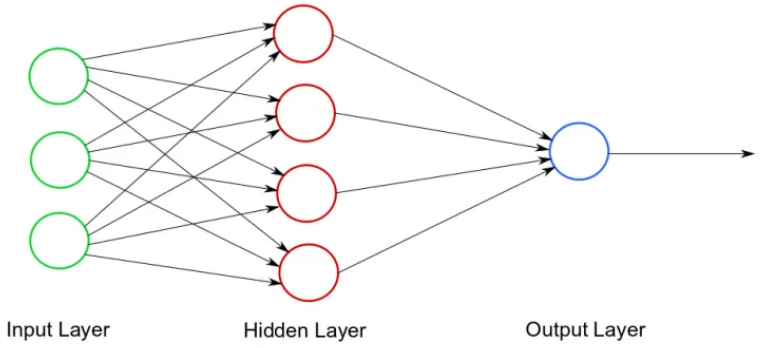
During the learning phase, checking the percentage of accuracy of your model could be an interesting data to know

A README.txt is always nice

THE MULTILAYER PERCEPTRON

Unfortunately, the results of Frank's perceptron were not as good as expected and the idea was abandoned. Moreover, a perceptron only allows you to compute linear functions.

A multilayer perceptron (MLP) is a modern feedforward artificial neural network.
It consists of fully connected neurons with a nonlinear kind of activation function, organized in at least three layers.



MLPs are able to distinguish data that is not linearly separable.
Pros:

- MLP models can solve complex non-linear problems ;
- they work well with both small and large input data ;
- they quickly provide some predictions after the training.

Cons:

- computations are time-consuming and complex ;
- it's hard to predict how much the dependent variable affects each independent variable ;
- the model functioning depends on the quality of training.

> *Modern feedforward networks are trained using the* ***backpropagation*** *method and are colloquially referred to as the* vanilla neural networks.
>
> *Wikipedia*

Now, try to generalize your single perceptron, such that you can have multiple layers of multiple perceptrons.

You may need to implement a way to visualize how efficient your neural network gets over time

There is no perfect answer on which value shoud you use for your learning rate or what activation function to use, but some are better than other. Try to do some research, and see what works best for you !

TO GO FURTHER

Resources to dig and interactive learning tools to play with:

- backpropagation
- tensorflow playground
- mladdict