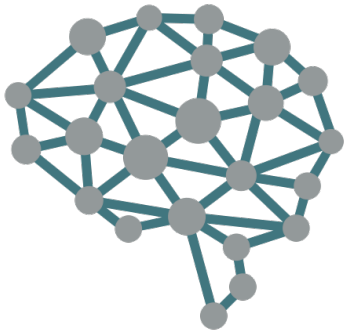
B5 - Computer Numerical Analysis

B-CNA-500

MyTorch

A Clash of Kings



MyTorch

binary name: my_torch
language: everything working on "the dump"
compilation: when necessary, via Makefile, including re, clean and fclean rules

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

Context

Now that you can easily plot against your foes and safely exchange with your allies, thanks to your great innovations in graph theory and cryptography, everything is in place for you to declare your supremacy over the Iron Throne.

To do so, you need to send your army to crush your enemies with fire and blood. However, you know from experience that even the best army in the world can be defeated if they are not properly guided through the use of the deadliest weapon of all: strategy.

Before starting the battles against your foes, you first need to find the best strategy to defeat them. Fortunately, you can train using the best strategy game known to men: chess.

PROJECT

You need to create neural network that can take a chess board as input, and outputs the status of the game: either a player wins (checkmate), a player has the other player's king checked, there is a draw (stalemate) or the game is still on.

For this project, you **MUST** provide a machine-learning-based solution!

PROJECT'S MAIN PARTS

This project will be separated into 2 main parts:

- the neural network:
  a group of neurons that are grouped into multiple layers linked together ;
- the optimization:
  improve the learning process either in learning speed, or in the method used to learn.

You **MUST** provide **at least**:

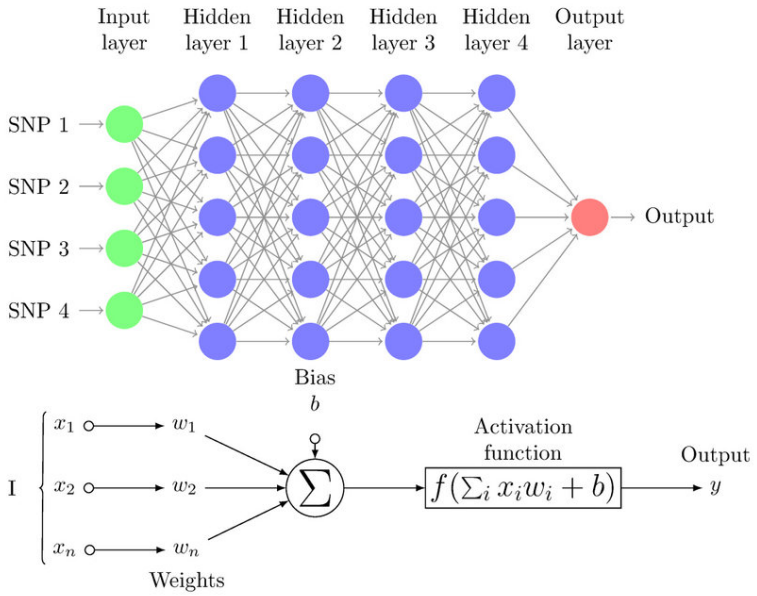
- A binary that can generate a new random generic neural network.
- A binary that takes an existing neural network, and either trains it on a dataset (training phase) or runs it on a dataset to check the output (prediction phase)
- A documentation explaining how your project works, how to launch the binaries, how you trained your model, the results of your benchmarks, etc.
- Your training datasets

You can have only one binary that does everything, but be careful to provide all the needed informations on how to use it!

You MUST send every script and training datas, so that we could theoretically easily generate a new neural network and train it the same way as you did. Providing only a pre-trained neural networks could be considered cheating!

Obviously, you are not allowed to use libraries that already handle neural networks for you (pytorch, tenserflow, etc)

The Neural Network



For this first part, you need to create a neural network that will take in its input layers every cell of a chessboard, and will output the state of the game depending on whose turn it is.

Every cell on the chessboard will be represented using a variant of the Forsyth–Edwards Notation to facilitate visualization:

- The pieces will be represented in the same way as the FE notation
- An empty cell will be represented by a dot
- Each line of the board will end with a LF (n)

You are free to design you neural network as you like it, but you MUST justify your architecture during the Defense (number of layers, number of neurons per layer, activation function, etc).

Your design choices matter! Providing benchmarks can be a good start for a justification

Starting the project with a Perceptron may be a good start

Here is an example of how you could launch your project (feel free to add more flags, cut the binary into multiple binaries/scripts, whatever fits your needs best):

Terminal

```
~/B-CNA-500> ./my_torch --help
USAGE
    ./my_torch [--new IN_LAYER [HIDDEN_LAYERS...] OUT_LAYER | --load LOADFILE]
[--train | --predict] [--save SAVEFILE] FILE

DESCRIPTION
    --new       Creates a new neural network with random weights.
Each subsequent number represent the number of neurons on each layer, from left
to right. For example, ./my_torch -new 3 4 5 will create a neural network with
an input layer of 3 neurons, a hidden layer of 4 neurons and an output layer of 5
neurons.

    --load     Loads an existing neural network from LOADFILE.
    --train     Launches the neural network in training mode. Each board in FILE
must contain inputs to send to the neural network, as well as the expected output.
    --predict   Launches the neural network in predictin mode. Each board in FILE
must contain inputs to send to the neural network, and optionally an expected
output.
    --save     Save neural network internal state into SAVEFILE.

    FILE    FILE containing chessboards
```

Optimization

This part is **mandatory**, you need to implement a way to optimize your neural network, either in the learning process, or in the learning speed rate.

Optimized breeding

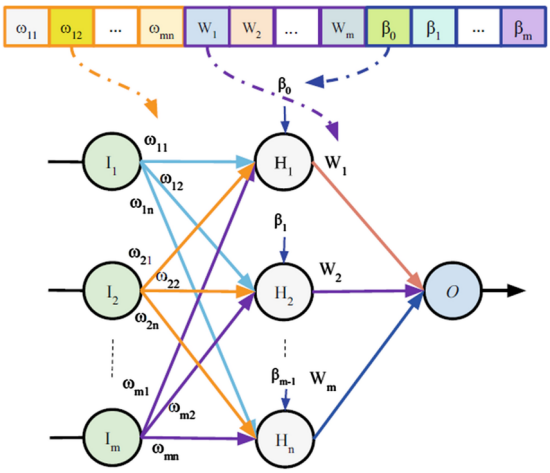
During the learning phase, you may encounter a *minor* issue: when training your neural network, it can converge towards a "good" solution, but it may not be the optimal one.
Playing with the learning rate and randomness could the trick, but you could use a different approach in order to optimize the learning method.

Add a **Genetic Algorithm** to add more diversity in your solutions!

Optimized speed learning

You can also speed up the (long!) process of training by parallelizing the computations.
There are a lot of ways to parallelize the training process. Think of:

- Multicore programming
- GPGPU (*CUDA*, *OpenCL*, …)



Documentation

You should be aware by now that writing professional documentation is extremely important.
Alongside the README that mainly inform us how to launch everything in your project, you must add everything that can help you justify in the Defense your design choices.

A benchmark without any visual proof is not worth anything!

The more *useful* documentation you provide, the better!

Bonus

- Optimize both the breeding AND the speed learning.
- A display of multiple learning curves on the same graph (useful to compare models)
- A whole chess AI
- Evaluate the learning phase with multiple metrics.