{EPITECH}

MYTORCH

A CLASH OF KINGS

MYTORCH

binary name: my_torch_generator/my_torch_analyzer
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, objfiles,…), 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 safely plot against your foes and exchange with your allies, thanks to your great innovations in 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 battling against your foes, you first need to find the optimal strategy to defeat them. Fortunately, you can simulate your battles by using the best strategy game known to men: chess.

Project

For this project, you need to make two binaries:
- A neural network generator: it must be able to generate a new neural network from a configuration file
- A chessboard analyzer: it can be launched either in training mode, or in evaluation mode.

For this project, you **MUST** provide a machine-learning-based solution! Any other kind of solution will be rewarded by a nice 0

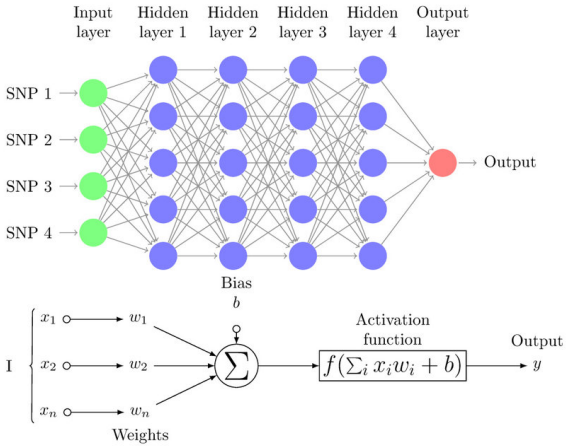
In completion with your binaries, you **SHOULD** provide a professional documentation explaining the results of your benchmarks.

You **MUST** keep every script and training datasets you used to train your network, so that we could *theoretically* 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!

You don't have to push your datasets, as they can get quite heavy. However, you must be able to show them during the Defense

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

The Generator



Before training an artificial neural network, you need to think about its architecture (or more generally, its hyperparameters). Because there is no perfect architecture and you might want to try different configurations, you **MUST** create a netural network generator that will generate a new neural network based on a configuration file.

```
Terminal
~/B-CNA-500> ./my_torch_generator --help
USAGE
    ./my_torch_generator config_file_1 nb_1 [config_file_2 nb_2...]

DESCRIPTION
    config_file_i   Configuration file containing description of a neural network we want
to generate.
    nb_i          Number of neural networks to generate based on the configuration file.
```

You are free to format your configuration file as you wish. Think carefully about what information you need to generate an artificial neural network!

```
Terminal
~/B-CNA-500> ls
basic_network.conf my_torch_generator
~/B-CNA-500> ./my_torch_generator basic_network.conf 3 && ls
basic_network_1 basic_network_2 basic_network_3 basic_network.conf my_torch_generator
```

You are free to store your neural network any way you want.

The Analyzer

Making a blank neural network won't help you go really far. Therefore, you need to create a binary that, given a neural network, will either train it, or use it to make a prediction.

For this project, your artificial neural network must take a chess board as input, and outputs the state of the game:

- ✓ "Checkmate": A player has won
- ✓ "Check": A player has the other player's king checked
- ✓ "Stalemate": There is a draw
- ✓ "Nothing": Nothing in particular

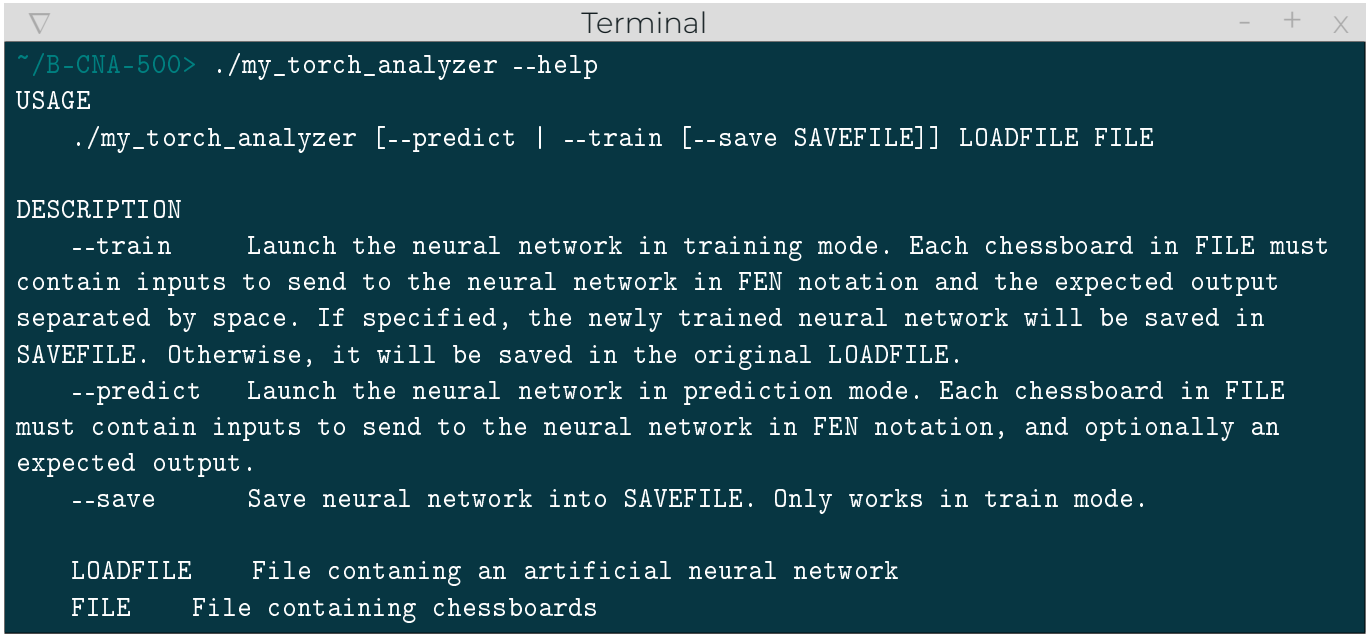
Only these 4 states are required. If you feel confident, you could add who check(mate)s whom. In this case, add the color of the winning player after the state. For example: "Checkmate White" means that the white player has won.

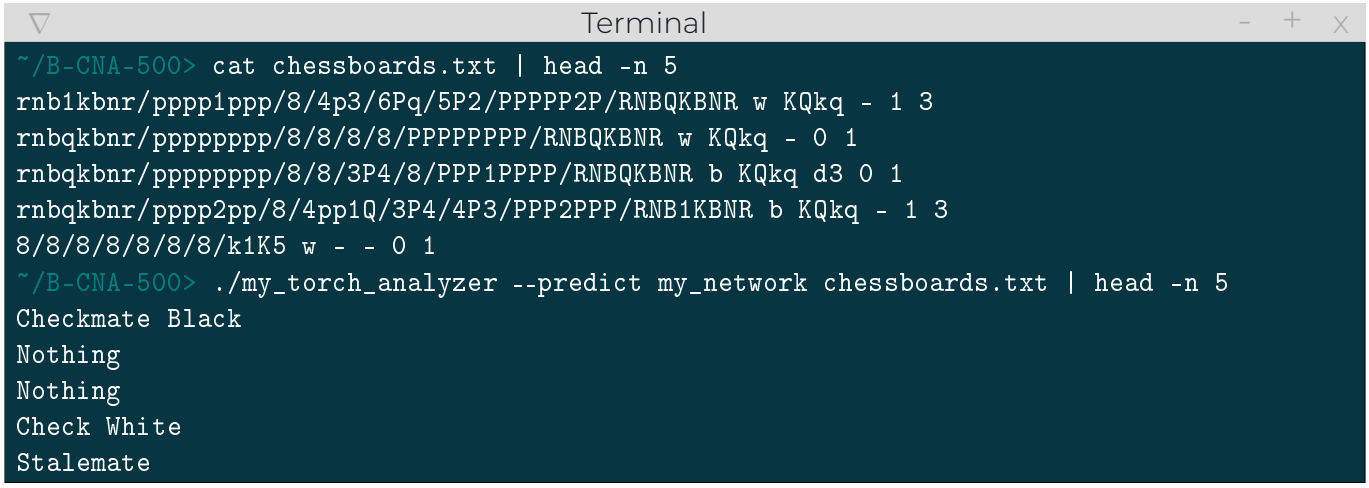
The chessboards will be represented using the Forsyth–Edwards Notation.

You are free to design you neural network as you like, but you will be asked to justify your **hyperparameters** during the Defense.

Your design choices matter! Don't forget to benchmark

In predict mode, your program **MUST** analyze every chessboard in the given file, and give its prediction in the same order as they are stored. In train mode, there are no limitations.


Terminal
~/B-CNA-500> ./my_torch_analyzer --help
USAGE
./my_torch_analyzer [--predict | --train [--save SAVEFILE]] LOADFILE FILE
DESCRIPTION
--train Launch the neural network in training mode. Each chessboard in FILE must contain inputs to send to the neural network in FEN notation and the expected output separated by space. If specified, the newly trained neural network will be saved in SAVEFILE. Otherwise, it will be saved in the original LOADFILE.
--predict Launch the neural network in prediction mode. Each chessboard in FILE must contain inputs to send to the neural network in FEN notation, and optionally an expected output.
--save Save neural network into SAVEFILE. Only works in train mode.
LOADFILE File contaning an artificial neural network
FILE File containing chessboards


Terminal
~/B-CNA-500> cat chessboards.txt | head -n 5
rnb1kbnr/pppp1ppp/8/4p3/6Pq/5P2/PPPPP2P/RNBQKBNR w KQkq - 1 3
rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1
rnbqkbnr/pppppppp/8/8/3P4/8/PPP1PPPP/RNBQKBNR b KQkq d3 0 1
rnbqkbnr/pppp2pp/8/4pp1Q/3P4/4P3/PPP2PPP/RNB1KBNR b KQkq - 1 3
8/8/8/8/8/8/8/k1K5 w - - 0 1
~/B-CNA-500> ./my_torch_analyzer --predict my_network chessboards.txt | head -n 5
Checkmate Black
Nothing
Nothing
Check White
Stalemate

Optimization

You **MAY** want 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, your cost function may converge toward a *local minima*.
Playing with the learning rate and randomness could help unstuck your ai, but you could use a better approach in order to optimize the learning method.

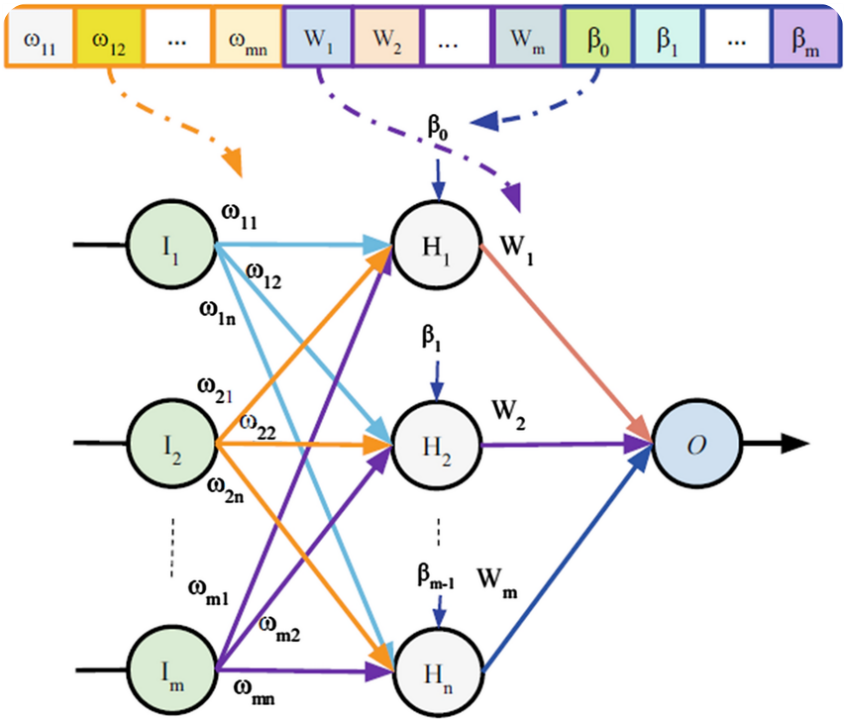
Instead of trying to find yourself how to fine-tune the hyperparameters, you may want to look for Hyperparameter Optimization.

For example, you could add a **Genetic Algorithm**!

Optimized speed learning

You could 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*, ...)


ω11
ω12
...
ωmn
W1
W2
...
Wm
β0
β1
...
βm
β0
I1
I2
Im
H1
H2
Hn
O
ω11
ω12
ω1n
ω21
ω22
ω2n
ωm1
ωm2
ωmn
β1
βm-1
W1
W2
Wm

Documentation

You should be aware by now that writing and maintaining professional documentation is extremely important. You should at least provide a README, some benchmarks, as well as any document 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.

v2