{EPITECH}

MY_TORCH

A CLASH OF KINGS

MY_TORCH

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).

With your cryptographic innovations, you can now plot against foes and trade with allies—everything’s set to claim 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 strongest army 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 deliver 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, it is **MANDATORY** to provide a machine-learning-based solution trained with supervised learning!

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

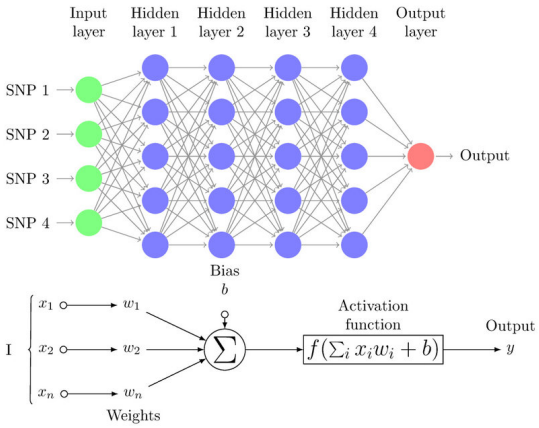
You **MUST** keep every script and training datasets you used to train your network, so that we could *theoretically* generate a new neural network and train it the same way as you did.
You **MUST** push a pre-trained neural network that can be loaded with your analyzer, and its name **MUST** start with "**my_torch_network**".

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, libraries handling neural networks (pytorch, tensorflow, …) are NOT allowed.

The Generator

Before training an artificial neural network, you need to think about its hyperparameters.



Because there is no perfect configuration and you might want to try different ones, you **MUST** create a generator that will create a neural network from 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. Likewise, you are free to store your neural network any way you want. 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.nn basic_network_2.nn basic_network_3.nn basic_network.conf
my_torch_generator
```

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.

Your neural network takes 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.

Try to go step by step! First, your AI could only differenciate two states: "Check" (checks and checkmates) and "Nothing" (stalemates and nothing). Then, as your training algorithm gets more and more sophisticated, try to make your AI differenciate the 4 states. And finally, in case of a check or checkmate, try to see if your AI could tell if it's the whites or the blacks that have the advantage!

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

Feel free to design you neural network as you like. However, you'll be asked to justify your **hyper-parameters**.

Your design choices matter! Don't forget to benchmark

In prediction 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 containing 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_torch_network_basic.nn chessboards.txt |
head -n 5
Check
Nothing
Nothing
Check
Nothing
~/B-CNA-500> ./my_torch_analyzer --predict my_torch_network_medium.nn chessboards.txt |
head -n 5
Checkmate
Nothing
Nothing
Check
Stalemate
~/B-CNA-500> ./my_torch_analyzer --predict my_torch_network_full.nn chessboards.txt |
head -n 5
Checkmate Black
Nothing
Nothing
Check White
Stalemate
```

Optimizations

You **SHOULD** implement a way to optimize the learning of your neural network.

Overfitting

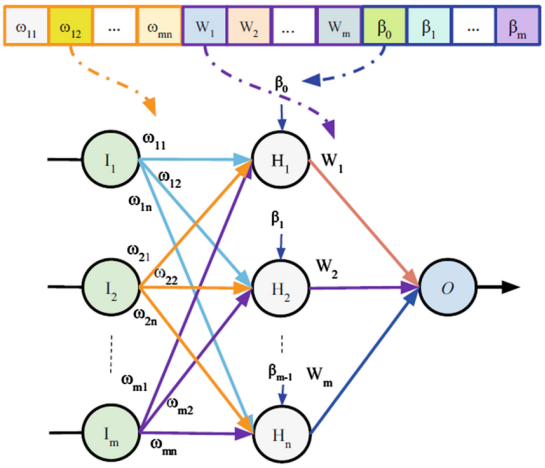
One major issue with machine learning is overfitting. Your program should try some ways to avoid that during the training process (for example using cross-validation or regularization processes)

Hyperparameter optimizations

During the training phase, you may encounter a slight 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 and Metaheuristics



Documentation

You should be aware by now that writing and maintaining professional documentation is extremely important. You could provide a README, some benchmarks, as well as any document that can help you justify in the Defense your design choices.
The more *useful* and *relevant* documentation you provide, the better!

A “benchmark” without any visual proof is not worth anything!

Bonus

- ✓ Optimize the speed learning using parallel computing (multithreading, multicore programming, GPGPU, ...)
- ✓ A display of multiple learning curves on the same graph (useful to compare models)
- ✓ Evaluate the learning phase with multiple metrics.
- ✓ A whole chess AI

v 2.3.0