Deep Learning Project Report

Quentin Goulas, Lucas Hocquette

ICE P24 SD9

ST7 - 2023

This project report aims to present the development framework, methodology and results obtained during the Deep Learning course project. This project focussed on hyperparameter optimization methods for deep learning models.

This report comprises four sections : an introduction to the project its premises, a presentation of the considered algorithms for this project, a description of the deployed system architecture and finall a discussion on the obtained results.

1 Project introduction

In the context of deep learning, hyperparameters are model parameters that aren't learnt by the model yet impact the learning capacity of the model. Hyperparameters include, but are not limited to :

— model architecture : number of hidden layers, size of the hidden layers, layer type (dense, convolutional, recurrent, ...), activation function of the layer

— preprocessing : normalization, data augmentation, ...

— regularization : Lasso, Tykhonov, dropout layers, ...

— loss function choice : euclidean distance, cross-entropy loss, ...

The choice of good hyperparameters is crucial to have a well-functioning model where having too simple a model may induce underfitting whilst building too complex a model might lead to overfitting. Yet, choosing the appropriate hyperparameters implies training an important number of models with different hyperparameters configuration, which induces high computing costs. Finding efficient methods to obtain the optimal hyperparameter configuration for a given DL task is therefore a topic of choice for the deep learning literature.

The most used Hyper Parameter Optimization (HPO) methods include :

— Babysitting : try different hyperparameter configurations, assess the accuracy of the different models, manually adjust the configurations and repeat till convergence. This method can be computationnaly intensive and labor intensive

— Grid search : try all the possible hyperparameter configurations and select the best one. This method requires very important computational and time resources, as well as a discrete hyperparameter space to evaluate

— Random search : try a proportion of randomly chosen hyperparameters in the possible hyperparameter space and select the best performing configuration among the tested configuration

— Gradient-based optimization : uses a gradient-descent approach to find the next hyperparameter configuration to evaluate. This requires a loss function that is differentiable in the hyperparameter variables

— Particle Swarm Optimization (PSO) : uses the particle swarm optimization algorithm to find the optimum fo a given function

— Genetic Algorithms which uses the genetic heuristic to find the optimal hyperparameters

— Bayesian methods using a modelled prior on the model loss function to converge to the best hyperparameter configuration

This project aims at implementing a HyperParameter Optimization framework to compare different hyperparameter optimizers. For this project, we will focus on Grid Search, Random Search and Particle Swarm Optimization (said to be the best hyperparameter optimizing method in the current literature)

2 Algorithm descriptions

We describe here the algorithms that have been implemented throughout this project : Grid search, Random search, Particle swarm optimization.

2.1 Grid search

This method consists in “brute forcing”

Algorithm 1: Grid search algorithm

Input: Hyperparameter space H, training set D, validation set V
Output: Best hyperparameter configuration h^* and accuracy a^*
for *each hyperparameter configuration* h *in* H **do**
    Train model M with hyperparameters h on D;
    Evaluate model M on V and compute accuracy a;
    if $a > a^*$ **then**
        $h^* \leftarrow h$;
        $a^* \leftarrow a$;
    end
end
return h^*, a^*

2.2 Random search

Algorithm 2: Random search algorithm

Input: Hyperparameter space H, training set D, validation set V, sampling proportion p
Output: Best hyperparameter configuration h^* and accuracy a^*
for *each hyperparameter configuration* h *in* H **do**
    if *random number* $r < p$ **then**
        Train model M with hyperparameters h on D;
        Evaluate model M on V and compute accuracy a;
        if $a > a^*$ **then**
            $h^* \leftarrow h$;
            $a^* \leftarrow a$;
        end
    end
end
return h^*, a^*

2.3 Particle Swarm Optimization

Algorithm 3: Particle Swarm Optimization (PSO) algorithm

Input: Hyperparameter space H, training set D, validation set V, swarm size S, ϕ_l local step size, ϕ_g global step size, w inertia, precision ϵ
Output: Best hyperparameter configuration h^* and accuracy a^*

```
Initialize the personal best position vector p_0 to x_0;
Initialize the position vector (randomly taken from H) x_0 of the swarm;
Initialize the speed vector v_0 to 0;
Initialize and compute accuracy vector a_0 of the swarm x_0;
t ← 0;
a*, h* ← max(a_0), x_0[argmax(a_0)];
repeat
    Compute a_t of the swarm x_t;
    ã, h̃ ← max(a_t), x_t[argmax(a_t)];
    if ã > a* then
        a*, h* ← ã, h̃;
    end
    t ← t + 1;
    Take u_l and u_g uniformly from [0, φ_l] and [0, φ_g];
    v_t ← v_{t-1}.w + (h̃ − x_{t-1}).u_l + (h* − x_{t-1}).u_g;
    x_t ← x_{t-1} + v_t;
until ‖v_t‖ < ε;
return h*, a*, t
```

3 Development framework

This project will be entirely developed on Python through the Pytorch deep learning library. Moreover, the CIFAR-10 dataset will be used throughout this study to train a LeNet5-type model architecture for image recognition. Our hyperparameters of interest are the output sizes of the the first two convolutional layers (C1, C3) and the output size of the first dense layer (F6)

For this project, we decide to build a HyperParameterOptimizer (HPO) object responsible for navigating the input hyperparameter space and organizing the different hyperparameter configuration tests. The HPO object then calls a dedicated library to generate a model with the appropriate hyperparameters and launches the trainings. The system architecture is given in Figure 1

To improve scalability, hyperparameter configurations are parsed as dictionaries, with name-value pairs of hyperparameter name and corresponding values. This enables to manipulate the same objects across hyperparameter spaces (which are hyperparameter configurations with multiple accessible values) and hyperparameter arrays (used to represent multiple hyperparameter configurations simultaneously).

4 Results

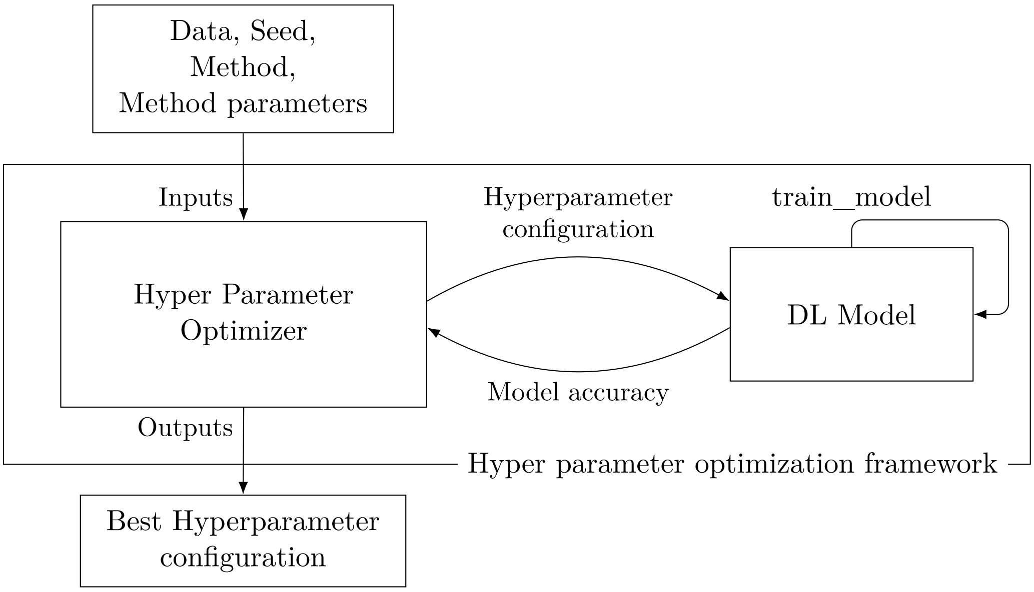

FIGURE 1 – Hyper parameter optimzation workflow