

---

# Deep Learning Project Report

Quentin Goulas, Lucas Hocquette

*ICE P24*

---

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 finally 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 computationally 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 for 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, Bayesian Optimizer Hyper Band 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. We also provide the pseudo-codes for the methods.

## 2.1 Grid search

This method consists in “brute forcing” the best hyperparameter configuration, i.e by trying all possible hyperparameter configurations on the hyperparameter space and returning the best configuration. This method is extremely computationnaly heavy, with the number of possible configurations increasing exponentially as the number of tested hyperparameters increases.

---

**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

This method consists in randomly selecting a proportion of all the possible hyperparameter configurations and returning the best configuration after trying all configurations in the sample. This method reduces slightly the computational time compared to grid search. However, this method does not solve the curse of dimensionality problem. As a matter of fact, the only method to solve this problem is to use a sampling proportion of the order  $p^d$  with the  $d$  the number of hyperparameters. This technique keeps computation time constant but dramatically reduces the exhaustivity of the method and success probability of the algorithm.

---

**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**  
     $r \sim \mathcal{U}([0, 1])$ ;  
    **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

The particle swarm optimization algorithm is a metaheuristic optimization algorithm which assumes next to no information on the functions to optimize. Modelled on the evolution of bird flocks,

it consists in evaluation the model accuracy on a small set of hyperparameter configurations, then choosing a new set of configurations on the basis of the best configuration seen overall and by the best configuration seen by the particles individually.

---

**Algorithm 3:** Particle Swarm Optimization (PSO) algorithm

---

**Input:** Hyperparameter space  $H$ , training set  $D$ , validation set  $V$ , swarm size  $S$ ,  $\phi_l$  local step size,  $\phi_g$  global step size,  $w$  inertia, precision  $\epsilon$

**Output:** Best hyperparameter configuration  $h^*$  and accuracy  $a^*$

Initialize the personal best position vector  $\mathbf{p}_0$  to  $\mathbf{x}_0$ ;

Initialize the position vector (randomly taken from  $H$ )  $\mathbf{x}_0$  of the swarm;

Initialize the speed vector  $\mathbf{v}_0$  to  $\mathbf{0}$ ;

Initialize and compute accuracy vector  $\mathbf{a}_0$  of the swarm  $\mathbf{x}_0$ ;

$t \leftarrow 0$ ;

$a^*, h^* \leftarrow \max(\mathbf{a}_0), \mathbf{x}_0[\text{argmax}(\mathbf{a}_0)]$ ;

**repeat**

    Compute  $\mathbf{a}_t$  of the swarm  $\mathbf{x}_t$ ;

$\tilde{a}, \tilde{h} \leftarrow \max(\mathbf{a}_t), \mathbf{x}_t[\text{argmax}(\mathbf{a}_t)]$ ;

**if**  $\tilde{a} > a^*$  **then**

$a^*, h^* \leftarrow \tilde{a}, \tilde{h}$ ;

**end**

$t \leftarrow t + 1$ ;

    Take  $u_l$  and  $u_g$  uniformly from  $[0, \phi_l]$  and  $[0, \phi_g]$ ;

$\mathbf{v}_t \leftarrow \mathbf{v}_{t-1}.w + (\tilde{h} - \mathbf{x}_{t-1}).u_l + (h^* - \mathbf{x}_{t-1}).u_g$ ;

$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} + \mathbf{v}_t$ ;

**until**  $\|\mathbf{v}_t\| < \epsilon$ ;

**return**  $h^*, a^*, t$

---

## 2.4 Bayesian Optimization HyperBand algorithm

This method uses a probabilistic model of the accuracy function given the different tested configurations and iteratively returns the next configuration to try out by choosing the maximizer of expected improvement. The probabilistic model used here is the Tree Parzen Estimator which uses a kernel density estimator to model the accuracy function of the model. To further improve the performance of the algorithm, the Hyper Band technique is used to dynamically allocate the amount of resources for each training. For this project, we rely on the HpBandSter library [\[clickable link\]](#) which provides a full optimizer framework —similar to ours— to apply BOHB which we integrated to our framework. Furthermore, this library enables asynchronous processing to try out multiple configurations simultaneously, therefore drastically increasing convergence speed.

## 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 convolutional layers (C1, C3, C5) 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 ob-

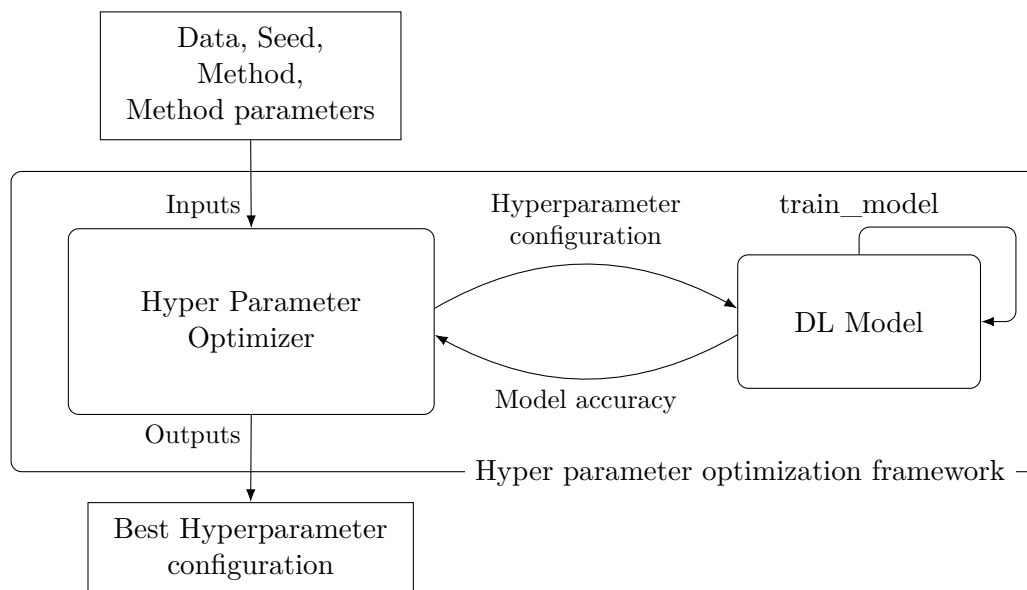


FIGURE 1 – Hyper parameter optimzation workflow

jects across hyperparameter spaces (which are hyperparameter configurations with multiple accessible values) and hyperparameter arrays (used to represent multiple hyperparameter configurations simultaneously).

## 4 Results