CentraleSupélec

université PARIS-SACLAY

Deep Learning Project Report

Quentin Goulas, Lucas Hocquette

ICE P24

December - 2024

Table des matières

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.

The source code for this project can be found at this repository [link]

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 [5] :

- 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, 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, 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 $\mathbf{p}_0$ 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

The convergence of the Particle Swarm Optimization algorithm cannot be proven in the general case (although some refinements can provide convergence to a local optimum [1]). Yet, empirical data shows the efficiency of this algorithm in a majority of cases.

2.4 Bayesian Optimization HyperBand algorithm [2]

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 BOHB optimizer framework —similar to ours— that we integrated to our framework. Furthermore, this library enables asynchronous processing to try out multiple configurations simultaneously, therefore drastically increasing computing efficiency.

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.

The accuracy function of the classifier is chosen as the proportion of accurate classifications from the model on a validation dataset.

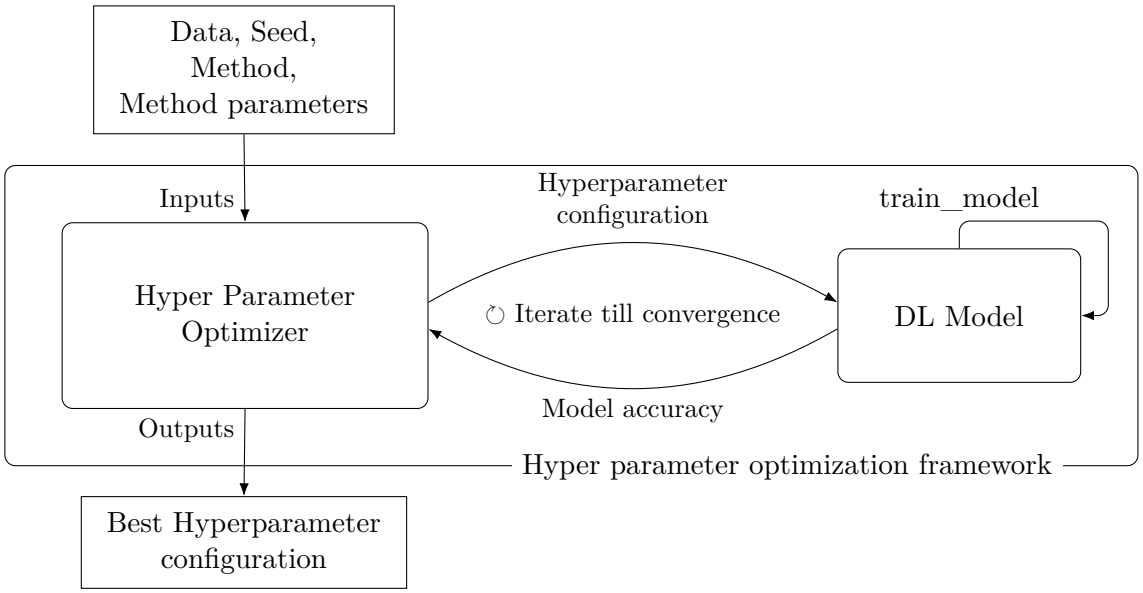

FIGURE 1 – Hyper parameter optimization workflow

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 Experimental setup

To try out the different hyperparameter optimizers, we search for the optimal configurations of a LeNet-5 type network [4]. More precisely, we look for the optimal number of filters for the three convolutional layers C1, C3 and C5 as well as the size of the network's dense layer F6. A schematic of the LeNet-5 architecture is provided in Figure 2.

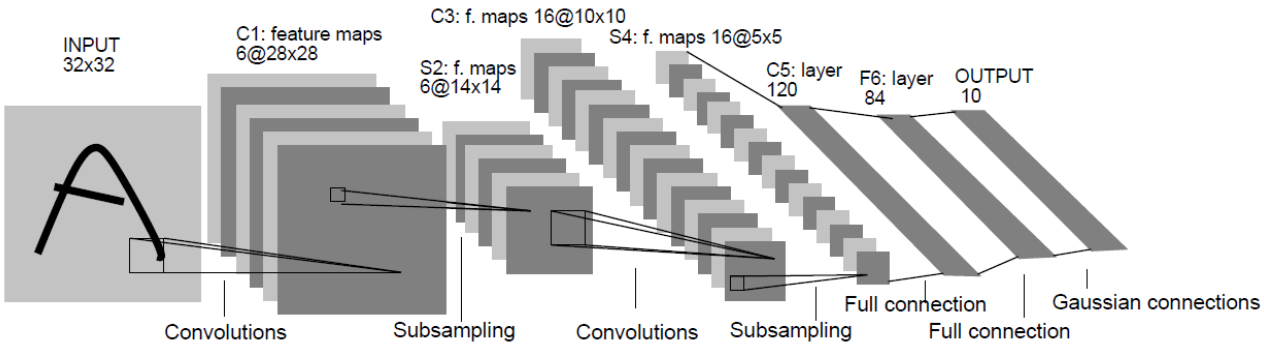

FIGURE 2 – LeNet-5 architecure with the hyperparameter configuration used in the original paper

We use the LeNet-5 architecture to recognize different images from the CIFAR-10 dataset [3], divided into 10 categories : { airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck}. The CIFAR-10 comprises 60 000 coloured images of 32×32 pixels with 6 000 images per category and split into a 50 000 image training set and a 10 000 image test set.

Model trainings for this project have been entirely done on local GPUs (GeForce MX230 and GeForce RTX3070 Laptop) with a batch of size 64. Training the classifier on CIFAR-10 as provided by `torchvision` led to an epoch duration from 20 to 40s depending on model complexity.

We use the following parameters for each algorithm :

Algorithm	Parameters
Grid search	$\emptyset$
Random search	p = 0.5
Particle Swarm Optimization	$S = 5, \phi_l = 2,\ \phi_g = 2,\ w = 0.5$
BOHB	$min_{epochs} = 10,\ max_{epochs} = 40,\ N_{iterations} = 20$

FIGURE 3 – Algorithm parameters

Unless indicated otherwise, the number of epochs for every trainings is set at 40.

For every method, we consider the following hyperparameter space to search :

Hyperparameter	Range
F6	[125,325]
Number of C1 filters	[1 :32]
Number of C3 filters	[1 :32]
Number of C5 filters	[40 :130]

FIGURE 4 – Hyperparameter space

We select this hyperparameter space by using the initial hyperparameter configuration used in LeNet-5 and explore neighbouring configurations to it.

Although the implemented algorithms enable increased efficiency compared to a babysitting or grid search approach, each optimization run took considerable amounts of time, allowing minimal repetitions to validate the results, leaving significant space to interpretation of the results and little quantitative results. We still present the results in the following section.

5 Results

Running the algorithms returns us the following results :

Algorithm	Hyperparameter config				Number of training epochs	Validation accuracy
	F6	C1	C3	C5		
PSO	324	26	19	40	$20 \times 5 \times 40$	66.8%
BOHB	21	24	23	35	40×40	67.0%
Grid Search	-	-	-	-	$200 \times 32 \times 32 \times 90 \times 40$	-
Random Search	-	-	-	-	$p \times 200 \times 32 \times 32 \times 90 \times 40$	-

FIGURE 5 – Optimal hyperparameter configurations

Due to the extreme resource consumption of grid search and random search, we weren't able to do full runs for these algorithms.

From these results, we observe that both PSO and BOHB provide good solutions for hyperparameter optimization, with reduced costs compared to Grid Search or Random Search. Although BOHB seems to have an edge on PSO in terms of convergence speed as well as reliabilty, we believe this

experiment on its own isn't sufficient to confidently compare BOHB and PSO in terms of performance and reliability.

Références

[1] A. P. Engelbrecht et F. van den Bergh. "A Convergence Proof for the Particle Swarm Optimiser". In : *Fundamenta Informaticae* (Januray 2010). doi : 10.3233/FI-2010-370.

[2] Stefan Falkner, Aaron Klein et Frank Hutter. "BOHB : Robust and Efficient Hyperparameter Optimization at Scale". In : *Proceedings of the 35th International Conference on Machine Learning*. Stockholm, Sweden, 2018.

[3] Alex Krizhevsky. *Learning Multiplye Layers of Features from Tiny Images*. Technical Report. Avr. 2009, p. 60.

[4] Yann LeCun et al. "Gradient-Based Learning Applied to Document Recognition". In : *Proceedings of the IEEE*. T. 86. Nov. 1998, p. 2278-2324. doi : 10.1109/5.726791.

[5] Li Yang et Abdallah Shami. "On Hyperparameter Optimization of Machine Learning Algorithms : Theory and Practice". In : *Neurocomputing* 415 (juill. 2020), p. 295-316. doi : 10.1016/j.neucom.2020.07.061.