Bayesian Hyperparameters Optimization

Matteo Petrone
matteo.petrone@stud.unifi.it

Lorenzo De Luca
lorenzo.deluca4@stud.unifi.it

September 2019

1 Introduction

In this paper we are going to analyze the use of a Global Optimization algorithm. Specifically, it is a Beyesian algorithm, based on evaluation of a surrogate model of the objective function.
We used it to choose hyperparameters in a Neural Network.
In the next paragraphs we briefly define theoretical aspects of Bayesian Optimization, and then we evaluate experimental results.

2 Bayesian Optimization

The main purpose of Global Optimization is to detect a global minimum of a function $f : S \to \mathbb{R}$, with $S \subseteq \mathbb{R}^n$, that is:

$$\min_{x \in S} f(x)$$

in particular we want to determine

$$f^* = \min_{x \in S} f(x)$$

and

$$x^* = \arg\min_{x \in S} f(x) : f(x^*) \leq f(x) \, \forall x \in S$$

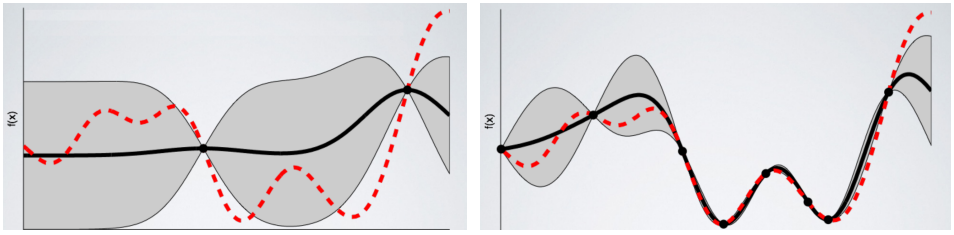
Therefore solving the problem means to identify the best solution among many, avoiding to choose local minimum.
To do this there are two kinds of methods:

1. **Deterministic Methods**, that provide an absolute guarantee of success but still require restrictive assumptions about the function.

2. **Stochastic Methods**, that generally require weaker hypotheses and ensure a probabilistic convergence to the global optimum.

The Bayesian algorithm used in this paper belongs to the second class.
The Hyperparameter Optimization is linked to an evaluation of the objective function, that is an extremely expansive job. In fact, to obtain a value of that function, we first need to train the model on a training set, and then evaluate it on a validation set.
It is important to underline that this problem become intractable if we have a very large number of hyperparameters or too deep Neural Networks, that would require too much time to train.

In general, it works finding next hyperparameter set to evaluate on real objective function. These, however, are firstly assessed on a surrogate function, that is easier to optimize than the objective function. The purpose of this method is to make the surrogate model better increasing observations.
In this way the surrogate model improves and the algorithm become aware of those regions that is worth exploring (see Figure 1, where surrogate model is the black function, trust region in grey and the real objective function in red).



(a) After 2 evaluations (b) After 8 evaluations

Figure 1: Surrogate model estimation

2.1 Why we chose Bayesian Optimization

Literature provides us a very large number of possibilities to treat this kind of problem, however our choice fell back on a Bayesian approach, and the main reason is due to efficiency. Our algorithm select next hyperparameters keeping track of past evaluations, using it to create a probabilistic model that associates hypeparameters to a probability represented by:

$$P(score|hyperparameters)$$

Then it takes long time choosing the hyperparameters so that less calls are made to the objective function. Others methods such as random search, make a choice in a completely uninformed way, respect to older selections. For this reason they are not very interesting in cases like ours.

3 Our Study

To pratically study the Bayesian Optimization, we optimized *learning-rate* and *weight decay* of a Convolutional Neural Network that classifies images from a well-known dataset (see 4.1).

3.1 Experiments

We provided two different domains to the Bayesian optimization method for both hyperparameters: $[10^{-4},10^{-2}]$ for learning-rate and $[0,10^{-2}]$ for weight decay, bounding values that literature considers better. The Bayesian algorithm starts with 5 random attempts to find best point to start and then it countinues for 25 iterations. For each iteration, the training process of the network consists in 120 epochs.
To reach more consistent result, before every training process, the weights of the network are initialized to default values.
We underline that objective function in the implementation of Bayesian optimization is maximized, therefore loss of the validation set is inverted, changing sign.
To evaluate performance we took for each training/validation process the minimum value of `validation loss`, and the coorrisponding values, of that iteration, of `validation accuracy`, `training loss` and `training accuracy`. To print these performaces is required to edit line 85 in Networks.py (see section 4) and select a module of a number ≤ 100.

4 Implementation

Then code, available on GitHub at the following link, consists in 3 main files:

data.py : where all data are splitted in training-set and validation-set.

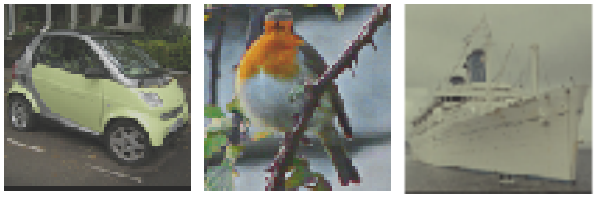
networks.py : here there is the implementation of the network and the methods that permit the training and validation phases.

evaluation.py : where there is the main method, the evaluation function for the Bayesian Optimization. [Run this file to test]

4.1 Dataset

The dataset used in the paper is STL-10. It includes 10 classes of RGB images, large 96x96 each. In Figure 2 there are 3 exemples of classes taken from STL-10. Specifically, the dataset is composed by:

- 500 training images per class
- 800 test images per class
- 100000 non-labeled images for unsupervised learning (we did not use them)



(a) car (b) bird (c) ship

Figure 2: Class example images from STL-10

4.2 Convolution Neural Network Architecture

The used network model is composed by 9 total layers and take an input image of dimensions 96x96x3. First and second layer are convolution with a 3x3 filter which produce 16 feature maps. With these parameters, the result of these two convolution is yet a 96x96 image. Third layer consists in a MaxPool, which halves the image. Fourth and fifth layers are different respect to the firsts two only for renerated feature maps: these return 32 of them. Sixth layer is a MaxPool exactly like the third. The last three layers consist in a linear trasformation, useful for the classification task.

4.3 Development Environment and Tools

The entire project was written in Python 3.6, using the open-source Pythorch library, based on Torch. It allowed us to implement the CNN.
The entire computational phase took place remotely, through SSH protocol, on a Ultron PC with six GPU Nvidia Tesla K80.

5 Risultati

Below are presented some graphs that examine the results obtained. We used tensorboardX to read the quality of the training/validation processes. In particular in the following graph (Figure 3) there is represented the trend of accuracy of the iteration of the Bayesian method which produced the highest score, which was the 16th iteration. Table 1 shows the numerical result: iteration, hyperparameter value (learning rate and weight decay), training loss, validation loss and validation accuracy.

6 Conclusioni

From obtained results we can observe that the Bayesan optimizer, after some iteration, can improve the quality of the model choosing better hyperparameters for it. The max accuracy obtained was 27.78%, far from the world record of 74.33% reached in 2015 (link); however this was not our goal.

Iterations	LearningRate	WeightDecay	TrainLoss	ValLoss	ValAccuracy(%)
1	0.001	0.0	1.4992	2.4179	24.05
2	0.000339	0.001	1.5348	1.9103	27.10
3	0.001	0.001	2.3028	2.3026	9.87
4	0.00001	0.001	1.9383	1.9782	19.82
5	0.00001	0.0	1.9112	1.9398	19.35
6	0.001	0.000381	2.3028	2.3026	9.87
7	0.000582	0.001	2.3027	2.3026	9.42
8	0.000354	0.000652	1.4981	1.8744	24.8
9	0.000181	0.001	1.5827	1.8597	25.7
10	0.000284	0.000843	1.5496	1.8805	26.3
11	0.000406	0.000455	1.4944	1.9312	26.2
12	0.000264	0.000686	1.5245	2.0316	25.75
13	0.00001	0.000211	1.9152	1.9610	20.65
14	0.000453	0.000744	1.5206	1.8499	26.12
15	0.000724	0.000170	2.3027	2.3026	9.42
16	0.000482	0.000588	1.5442	2.0037	27.78
17	0.000213	0.000174	1.5273	1.9227	25.52
18	0.000399	0.000267	1.5109	1.8547	26.4
19	0.001	0.000716	2.3028	2.3026	9.87
20	0.000394	0.000832	2.3026	2.3026	9.42
21	0.000195	0.0	1.5280	2.2242	25.95
22	0.000788	0.000880	2.3027	2.3026	9.42
23	0.000661	0.0	1.4828	2.5396	24.7
24	0.00001	0.000515	1.9887	2.0711	19.97
25	0.000779	0.000392	2.3027	2.3026	9.42

Table 1: Results obtained with the bayesian algorithm: each iteration corresponds to a training of 120 epochs. The validation accuracy column is highlighted in light blue and the best result is green.

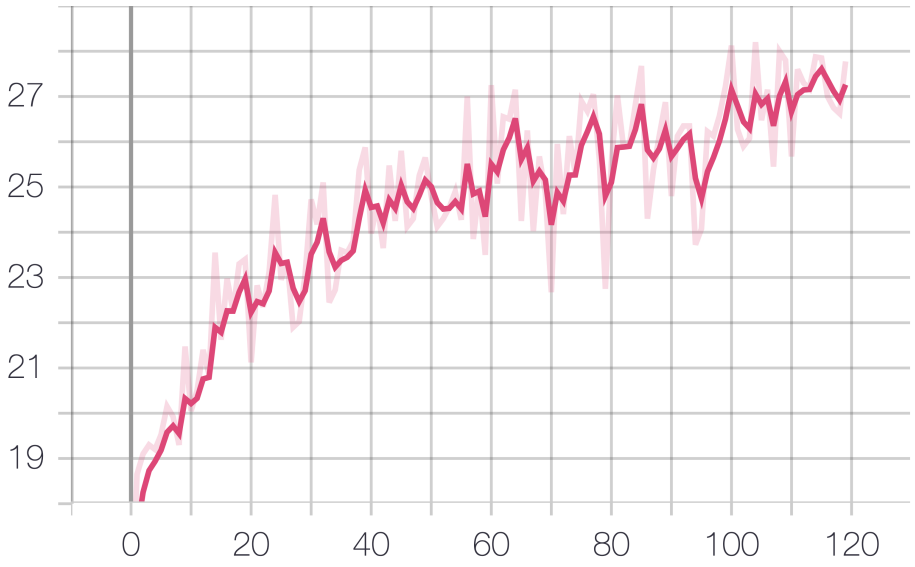


Figure 3: Validation accuracy at 16th iteration during the 120 epochs