Hyperparameters tuning for Convolutional Neural Network based on Global Optimization

Jacopo Bartoli
jacopo.bartoli@stud.unifi.it

Iacopo Erpichini
iacopo.erpichini@stud.unifi.it

***Abstract*—The aim of this experiment is to analyze the performance of a simple Global Optimization algorithm used to perform an hyperparameters optimization of Convolutional Neural Network. In particular we use an implementation of Bayesian algorithm and we will compare its performance to a quasi-random search algorithm.**

***Index Terms*—CNN, Bayesian, quasi-random, hyperparameters optimization**

I. Introduction

The training of a neural network consists in optimizing an objective function known as loss, which measures how much the predictions of the network are like the desired outputs. These outputs depend on a certain number of parameters and **hyperparameters**.

The sub-optimal parameters (or weights) can be learned through a training phase by taking in to account gradient of the loss function with respect to them. With this we can apply optimization techniques such gradient method and derivatives. An alternative could be exploiting an approximation of the function in the second order and apply quasi-Newton methods.

These techniques are not available for the selection of hyperparameters, which must be already chosen before the training phase begins. This happen because they often represent some feature of the net's structure.

The choice of hyperparameters can be formulated as a global optimization problem, $f^* = \min_{x \in S} f(x)$ with $f : \mathrm{S} \to \mathbb{R}$ and $\mathrm{S} \subseteq \mathbb{R}$, because we want detect a global minimum of a function.

In this case evaluating the objective function is very expensive (in fact, it does require network training), and for which there is no efficient method for calculating the gradient or the Hessian matrix. In this study a relatively simple network training is considered to compare a global optimization algorithm with a quasi-random search method.

II. Hyperparameters tuning

In this paper we have analyzed two different techniques for hyperparameters optimization.

The first methods analyzed to fulfill this task was the Bayesian algorithm. This is a very used global optimization algorithm. This method is based on the idea of working on a surrogate function instead of the objective one because it is easier to optimize. The surrogate function is chosen in regard to a prior probability distribution. Then using as input some evaluation of the objective function the algorithms calculate a posterior probability distribution. This last one is used to select the next point which we need to evaluate and then the new values are used for refine the posterior distribution.

The other technique used in this work was a quasi-random search based on Sobol sequences. These are low discrepancy random sequences that use a base of two to form successively finer uniform partition of the space, and then reorder the coordinate in each dimension. The main difference from a standard random search is that the solution space is covered more evenly as we can see from 1 and 2. For example this will avoid the selection of solution points really close to each other (cluster of point), that can't lead to an improvement of the objective function.

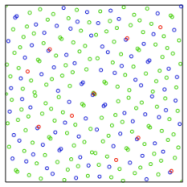

Fig. 1. Sobol sequences

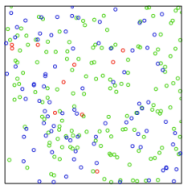

Fig. 2. Uniformly Random Points

III. Architecture

The neural network that we used is inspired by LeNet5 [1]. As we said before the purpose of this project is not to obtain the best accuracy on the dataset but is to study the behaviour of two Global Optimization methods. To achieve our goal we used a simple neural network that reach a relative low accuracy on the dataset. Thanks to this we could see in a better way the improvement that our hyperparameter optimization produced. Here we can see the structure of our network:

A. Tools

The entire project was written in Python 3.6, using the open-source Pythorch library, based on Torch. It allowed us

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 10, 28, 28]             760
         MaxPool2d-2          [-1, 10, 14, 14]               0
            Conv2d-3          [-1, 20, 10, 10]           5,020
         MaxPool2d-4            [-1, 20, 5, 5]               0
            Linear-5                  [-1, 10]           5,010
================================================================
Total params: 10,790
Trainable params: 10,790
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 0.09
Params size (MB): 0.04
Estimated Total Size (MB): 0.15
----------------------------------------------------------------
```

Fig. 3. Network used for evaluate CIFAR-10 dataset.

to implement the CNN. For our text we used a Google's service called Colab that allows to run a Jupyter Notebook on a provided CPU or GPU. This is the link at the notebook on Colab.

B. Bayesian Optimization

For the Bayesian optimization algorithm we used a python implementation provided by Bayesian Optimization library. This particular implementation uses Gaussian processes.

C. Quasi-Random Search

For the quasi random search we used a python library called Optunity. It has an implementation of Sobol sequences, and allows us to make the quasi-random through our hyperparameters space.

IV. Dataset

The first experiment was conducted on MNIST dataset [2].

It contains images of handwritten numbers annotated and has inside 60,000 images of train and 10,000 of tests, the images are 1x28x28[1] pixels.

Fig. 4. Example of elements in MNIST.

Several scientific publications have focused on the goal of obtaining a low error rate on the classification. In one of these, concerning a work based on the use of a hierarchical system of convolutional neural networks, an error rate of 0.23% is reported. This tells us that this dataset is very didactic but it is not useful for our goal.

With this dataset our simple network achieve an extremely high level of accuracy and we were not able to point out the difference on accuracy with our hyperparameters tuning techniques. So we decided to switch dataset to try to underline the improvement of our tuning methods.

In order to do that we tried the CIFAR-10 dataset [3], it consists of 60,000 3x32x32 pixel colour (3-channels) images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images. The classes are entirely mutually exclusive.

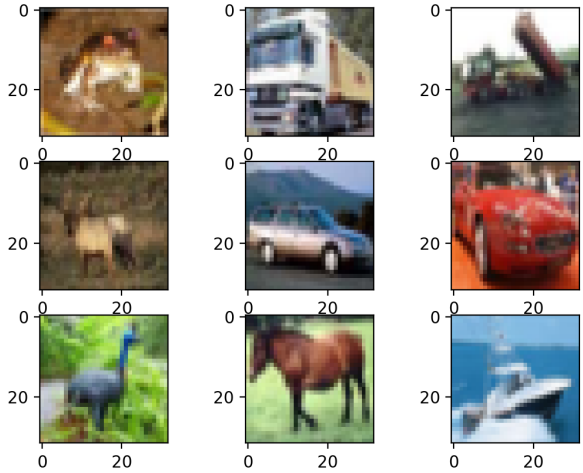

Fig. 5. Example of elements in CIFAR-10.

V. Implementation

The code is available on GitHub and on Google Colab at the following links: GitHub Code, Google Colab Code.

The code's structure is the same but in Jupyter Notebook there are some comments for a better understanding of the code and the selection of the parameters needed for replicate our tests or try something different. In the GitHub repository there is a *ReadMe* that explain how to install the libraries needed to run the code.

On GitHub we divided our project into 3 files:

- ***test.py***: Here is possible to change the parameters for tests and the function from the libraries that implements the bayesian and quasi random algorithm.
- ***network.py***: We have the function to create, train and evaluate the network and also we have a local script for print a summary of the network.
- ***dataset_management.py***: Here we use the function from pytorch for extract the database used for our experiments.

VI. Experiment

In our experiment we focused on the optimization of only two hyperparameters: weight decay and learning rate. The first one is a value, usually between 0 and 1, that multiply the

[1]Channels x Width x Height.

square sum of weight in the loss function in order to not get out of hand and compromise the learning algorithm. The second one represents the amount that weight are updated during the training phase.

In our tests we used 25 attempt for both quasi-random search and Bayesian method to maximize the objective function. In particular in the Bayesian one, we use 5 attempts for the choice of the starting point of the algorithm and 20 for the algorithm itself. In order to reduce the stochastic component of these tests we run them 10 times each and made the mean and the standard deviation of the target values.

In particular the function that we want to minimize is the loss function. Our implementation uses the negative log-likelihood loss. During our tests the network is trained with 50 epochs.

The hyperparameters space that we have used in our test is:
hyperparameters = {"learning_rate": (0.0001, 0.1), "weight_decay": (0, 0.1)}

VII. Results

As we can see in I the accuracy of both models are pretty even. The loss score is slightly better with the Bayesian algorithm. This behaviour can be assigned to the few number of point evaluated.

TABLE I
Mean Accuracy and Loss

	Mean Loss	**Std Dev**	**Mean Accuracy**	**Std Dev**
Quasi-random	1.0868	0.0158	63.0020%	0.6896%
Bayesian	1.0792	0.0267	62.9667%	1.1370%

In Table II we report an example of 25 point evaluated with a single run of Bayesian algorithm. For the quasi-random search the table structures is the same. In yellow we can see the five initial iteration for the algorithm and in green we have the best loss with the relative accuracy on the validation set. This result show us the best hyperparameters identified in that run.

VIII. Conclusion

In our tests both the hyperparameters tuning techniques, achieved the similar results. Probably this can be assigned to the low number of points chosen to evaluate the objective function. However we can assume that with more points the Bayesian algorithm will score a better result than the quasi-random search.

In the end, observing our tests, we can say that both methods are a good way to chose hyperparameters.

References

[1] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, november 1998. Gzipped PostScript, 75 pages, 709897 bytes

[2] Yann LeCun, Courant Institute, NYU, Corinna Cortes, Google Labs, New York, Christopher J.C. Burges, Microsoft Research, Redmond. "MNIST handwritten digit database."

[3] The CIFAR-10 and CIFAR-100 are labeled subsets of the 80 million tiny images dataset. They were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10

TABLE II
BAYESIAN EXPERIMENT

Bayesian 25 evaluation, 5 init point, 50 epochs per evaluations				
Iter	**Loss**	**Learning Rate**	**Weight Decay**	**Accuracy Validation**
1	1.4682993084002451	0.045734877141134014	0.029746750991550254	0.4812
2	1.4911971714845889	0.010761720432692988	0.034991773553814325	0.4747
3	1.8028555911058073	0.08611842338433033	0.05069879330905539	0.3705
4	1.7239089824591474	0.039102745150275946	0.07347505811478032	0.3895
5	1.2944789612369172	0.005042318359375003	0.00036658203125	0.5594
6	2.2894782154423416	0.0001	0.0	0.1255
7	2.3019540142861143	0.0001	0.1	0.1259
8	1.289038735210516	0.091580693359375	0.00073783203125	0.55
9	1.254048059700401	0.066855443359375	0.00049033203125	0.5697
10	1.3045590318691958	0.08038329355153166	0.01411036179664859	0.5316
11	1.8104093264622294	0.02799025696437753	0.02453245238966922	0.3618
12	1.277805628670249	0.069946099609375	0.00076876953125	0.5897
13	1.347356940910315	0.08273763827659827	0.0	0.5493
14	1.1217950855850414	0.043444626312879	0.0	0.6133
15	1.5347245832917038	0.1	0.023979266567333894	0.4636
16	2.2963736421743017	0.0001	0.059028117300117575	0.1095
17	1.5772969677190112	0.03783213155032981	0.044952283457541556	0.4423
18	1.303392969119321	0.029388142459315356	0.014257077767293345	0.5354
19	1.2059591818766988	0.05213680595825444	0.007619124130867758	0.5752
20	1.1369534339874414	0.05239800549599426	0.0	0.613
21	1.1826573185100677	0.034982903183658286	0.0	0.5906
22	1.1821330299802646	0.046958993236411416	0.0	0.5909
23	1.7686269473118388	0.06334330558678612	0.06606360258636525	0.361
24	2.004098408541102	0.1	0.06920598148063337	0.2875
25	1.5713350696928183	0.06819162068021895	0.03296104203062853	0.4281