Hyperparameter Optimization of Deep Neural Networks: Combining Hyperband with Bayesian Model Selection

Hadrien Bertrand*[1,2], Roberto Ardon[2], Matthieu Perrot[3], and Isabelle Bloch[1]

[1]LTCI, Télécom ParisTech, Université Paris-Saclay, France
[2]Philips Research, France
[3]L'Oréal R&I, France

Abstract

One common problem in building deep learning architectures is the choice of the hyper-parameters. Among the various existing strategies, we propose to combine two complementary ones. On the one hand, the Hyperband method formalizes hyper-parameter optimization as a resource allocation problem, where the resource is the time to be distributed between many configurations to test. On the other hand, Bayesian optimization tries to model the hyper-parameter space as efficiently as possible to select the next model to train. Our approach is to model the space with a Gaussian process and sample the next group of models to evaluate with Hyperband. Preliminary results show a slight improvement over each method individually, suggesting the need and interest for further experiments.

Keywords: Deep Learning, Gaussian Process, Bayesian Optimization, Hyperparameter Optimization

1 Introduction

The most common strategy when trying to solve a task using deep learning is to start from a well-known architecture such as AlexNet [1] or VGG-16 [2], which is usually enough to obtain decent results. The question then is how to improve them. One possibility is to acquire more data but this can be very costly depending on the type of data and the task (for example, segmentation of particular organs in medical images).

The other possibility is to change the architecture of the network. This requires either understanding the effect of each part of the network and how the different parts work together, something obscure for both beginners and experts in the field, or trying many different architectures in the hope of finding a better one.

Mathematically, we can pose the problem as follows. The hyper-parameter space $\mathcal{X}$ can be seen as a hypercube where each dimension is a hyper-parameter. Any particular combination x in $\mathcal{X}$ determines a unique network f that can be trained then evaluated according to a loss function $\mathcal{L}$. The goal is to find as fast as possible the combination x_* that minimizes the loss:

$$x_* = \underset{x \in \mathcal{X}}{\operatorname{argmin}} \, \mathcal{L}(f(x)) \tag{1}$$

Each evaluation of a combination is costly as it requires training a neural network. Moreover, hyper-parameters can be discrete or even categorical. It is impossible to try all combinations as the number of possibilities grows exponentially with each additional hyper-parameter. Therefore smarter policies are required and this has given rise to the field of hyper-parameter optimization. It has raised considerable development effort in recent years due to its importance in deep learning.

Let us make clearer the notion of speed in this context. The search is allocated a total budget B in minutes that each policy is free to allocate however it wants. What matters is the best model that could be found in this time. The choice of having a budget in time means that the model will not be trained in epoch as is the most common, but in minutes. This is a practical choice that allows to estimate accurately the total time that will take the search, but it will have the effect of favouring small networks. Indeed, two models trained an equal amount of time will not have seen an equal amount of data if one model is bigger and thus slower than the other.

The next section is an overview of the field, with particular attention paid to its role in deep learning.

*hadrien.bertrand@telecom-paristech.fr

We then present in Section 3 two methods, Bayesian optimization and Hyperband, which we will combine. These methods are evaluated on a standard classification dataset and results are discussed in Section 4. We conclude by a discussion of the limitations of the proposed methods and possible research directions in Section 5.

2 State of the Art

The simplest method for hyper-parameter optimization is random search [3]. It consists in picking combinations randomly in a pre-defined hyper-parameter space until a good enough model is found or resources are exhausted. It is easy to implement and to parallelize. And it is surprisingly efficient. The reason is that for most tasks there are only a handful of hyper-parameters that have a significant effect on the loss, meaning that the relevant hyper-parameter space is actually of low-dimensionality. As a consequence it is likely when picking combinations to choose one with optimal values for the important dimensions.

It is not known in advance which hyper-parameters are relevant to the task, otherwise we would discard the ones with low impact when constructing the hyper-parameter space. A better algorithm should be able to spend an appropriate amount of attention to the hyper-parameters in proportion of their impact on the task at hand.

One such method is Bayesian optimization. A recent review of the field can be found in [4]. The idea is to construct a probabilistic model of the loss function and update it sequentially as new combinations are tried. An acquisition function will then take into account the uncertainty of the model to guide exploration and select new combinations. Further details are given in Section 3.1.

The problem of hyper-parameter optimization can be seen from the perspective of a multi-armed bandit, where each combination is an arm and there is only a finite amount of resources (training time) to allocate to each arm. This has given rise recently to Hyperband [5], see Section 3.2 for details.

Neuro-evolutionary algorithms have also been applied to hyper-parameter optimization. Modern approaches on the subject [6] [7] allow a great flexibility on the architecture but require a lot of resources. They also do not stop training prematurely like Hyperband.

Recently reinforcement learning based approaches have been developed. One neural network is trained to construct other neural networks optimal for a given task [8] [9]. Like evolutionary algorithms, the main advantage is the greater variability of the tested architectures but at a higher cost in resources to train the controller network.

One major advantage of the last two approaches is the ability to use conditional hyper-parameters, i.e. hyper-parameters that must be chosen only if another hyper-parameter was chosen at a particular value. Bayesian Optimization and Hyperband have no simple way to integrate such hyper-parameters.

3 Methods

3.1 Bayesian Optimization

The procedure of Bayesian optimization is as follows (see [4] for a review of the topic and [10] for an approach closer to ours): we model the loss function of our hyper-parameter space using a probabilistic model, here a Gaussian process. We then compute an acquisition function, which takes into account the model posterior to select the next combination to evaluate.

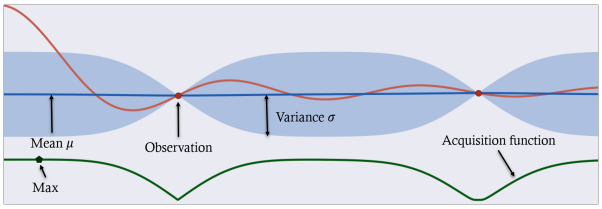

Figure 1: Bayesian Optimization on a one dimensional function. The orange curve is the true loss function, the blue one is the prediction of the Gaussian process. The green curve is the acquisition function. The next evaluation will be at the maximum of the green curve.

A Gaussian process [11] is a supervised learning model that gives a Normal distribution on each predicted point. It is specified by its covariance function, which we choose to be a Squared Exponential Kernel:

$$k(x, x') = \exp\left(-\frac{||x - x'||_2}{2l^2}\right) \quad (2)$$

Here l is a vector of the same dimensionality as the input x. It specifies the length scale of each dimension. It is found at training time by optimizing the log-marginal likelihood:

$$\log p\left(y|x, l\right) = -\frac{1}{2}y^T K^{-1} y - \frac{1}{2}\log|K| - \frac{n}{2}\log 2\pi \quad (3)$$

where K is the Gram matrix of all training points, such that $K_{ij} = k(x_i, x_j)$.

Our chosen acquisition function is the Expected Improvement [12], which uses the improvement function I [4]:

$$I(x) = (y_* - \mu(x))\,\mathbb{1}\,(y_* > \mu(x)) \tag{4}$$

where x represents a given configuration, y_* is the minimum loss found so far and μ is the mean returned by the Gaussian process. It will be 0 if the predicted mean is lesser than the best loss found so far (which means there is no improvement), otherwise its proportional to the gap between the predicted mean and best loss. From there, the Expected Improvement is:

$$EI(x) = \mathbb{E}[I(x)] = s(x)[u\Phi(u) + \phi(u)] \tag{5}$$

with

$$u = \frac{y_* - \mu(x)}{\sigma(x)} \tag{6}$$

where σ is the variance returned by the Gaussian process, and Φ and ϕ are the normal cumulative distribution and density function, respectively.

This function offers a good compromise between exploration of regions with high uncertainty and exploitation. Moreover, it does not add any hyper-parameter to tune. Once we have computed EI for each configuration, we do not pick the argmax as classically done in the literature but we normalize the results to make a distribution from which we sample the next configurations to test.

One important aspect of this method is that the training time of the model is a dimension of the hyper-parameter space. This means that the method is free to choose how long to train each model. In practice we found that it prefers longer training time, even though it looks like a lower training time is enough to get a good estimate of the worth of the model and would allow the evaluation of more models. This is a flaw of the method, which lacks an explicit notion of budget and a way to best exploit it.

3.2 Hyperband

The idea behind Hyperband [5] is to pick a group of models uniformly in $\mathcal{X}$, train them for a certain time, then discard the worst performing ones, and repeat until a few combinations remain. The problem is how to decide when to evaluate the models. If this is done too early, we cannot differentiate the good from the bad ones, and if too late, we waste valuable training time on bad models.

Hyperband addresses this by creating brackets, each with the same total budget. The brackets differ in when they start to evaluate models. The first one will do so after, e.g. 1 minute, the next bracket after 3 minutes and so on. As a consequence, the first bracket will try many more models than the last one, but might discard some good models too early.

The method is governed by only two hyper-parameters: the maximum amount of resource that can be given to a single model R and the proportion of models discarded at each evaluation η. We chose $R = 27$ minutes and $\eta = 3$, meaning that we keep 1/3 of models at each evaluation. The time resource parameter R was chosen so low because the networks we are training are small and the task is simple enough so that 27 minutes is sufficient for most of them to converge.

3.3 Combining approaches

Bayesian optimization and Hyperband being orthogonal approaches, it seems natural to combine them. Hyperband chooses the configurations to train uniformly and intervenes only during training. On the other side, Bayesian optimization picks configurations carefully by modelling the loss function, then let them train without interruption.

As a result, Hyperband does not improve the quality of its selection with time, and Bayesian optimization regularly loose time training bad models.

The combination is straightforward and fixes those two problems. Model selection is done by Bayesian optimization as described in Section 3.1, then Hyperband train them as described in Section 3.2.

The proposed algorithm is as follows: we pick the first group of configurations uniformly and evaluate them with Hyperband. All subsequent selections are done by training a Gaussian process with a squared-exponential kernel on all evaluated models. We then compute the expected improvement of all untested combinations, normalize it to make a probability distribution from which we sample the next group of configurations.

4 Experiments and Results

We compare the three methods presented above: Bayesian optimization, Hyperband, and Hyperband using Bayesian optimization. The algorithms were implemented in Python using scikit-learn [13] for the Gaussian processes and Keras [14] as deep learning framework.

Comparison was done on the CIFAR-10 dataset [15], which is a classification task on 32x32 images. The

image set was split in 50% for the training set used to train the neural networks, 30% for the validation set used for the Bayesian model selection and the rest as test set used for the reported results below.

Each method had a budget of 4000 minutes of training on a NVIDIA TITAN X. The choice to constrain in time instead of epoch means that the quantity of data seen by the models depends on the GPU in addition to the consequences pointed in the introduction.

The chosen architecture was a standard convolutional neural network with varying number of layers, number of filters and filter size per layer. Other hyper-parameters involved in the training method are: the learning rate, the batch size and the presence of data augmentation. In the end there were 6 hyper-parameters to tune for a total of 19 200 possible configurations.

The metric that matters when comparing methods is the minimum loss of all models tried at a given time, and is illustrated in Fig. 2 for the two individual methods and their combination (one run each). These results suggest that Bayesian optimization performs slightly worse than both other methods, and that Hyperband with Bayesian selection finds a better model quicker than Hyperband. However, due to the random nature of the methods, many runs would be needed to draw definite conclusions from Fig. 2.

It might be more informative to look at the running median of the loss of all models trained at a given time as in Fig. 3. Even if in the end only the best model matters, the running median gives us an idea of the quality of the models tried. Both methods using Bayesian optimization start with a higher median which decreases with time, while Hyperband has a lower median which stays flat and is eventually beaten by Hyperband with Bayesian selection.

One possible explanation is that Bayesian optimization is very exploratory at first and most boundaries of our hyper-parameter space are bad models. It then becomes better as the variance of the Gaussian process diminishes and better combinations are selected.

The fact that Bayesian optimization performs worse in both losses might mean that it is more important to try many models and discard them very quickly over trying less combinations but training them longer.

5 Discussion

Due to the part of chance in all three methods, one run is not enough to draw definite conclusions on which of them is best. Ideally we would have used cross-validation on each method and compared the methods on other tasks but lack of time prevented us from doing so and this is left for future work.

In any case, the results for Hyperband and Bayesian Optimization were consistent with the literature. With a low budget, Hyperband will perform better on average than Bayesian Optimization and should be preferred. It also has the advantage of being easier to implement and to parallelize.

At the end, each policy has trained many models from which we select only one. However we have a few high performing models with very similar error rates but a good diversity in architecture. Selecting the top five or ten and making an ensemble with them would give a very robust classifier. We can even consider applying model distillation on the ensemble to keep the advantages of a small network.

The approach of modelling the hyper-parameter space could also be used for transfer learning. One application would be for the same task, but with a bigger dataset containing more data and classes. It can be expected that the model of the hyper-parameter space learned before would be informative for the new dataset and would allow us to find a good model faster.

Closer to the authors' interest would be similar tasks, for example in a medical imaging context, where the methods would have been applied to the classification of images from a particular modality, say MRI as in [16], and we would now like to apply them to the classification of images from another modality, ultrasound for example.

References

[1] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, pp. 1097–1105, 2012.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[3] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, pp. 281–305, Feb. 2012.

[4] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proceedings of the IEEE*, vol. 104, pp. 148–175, Jan 2016.

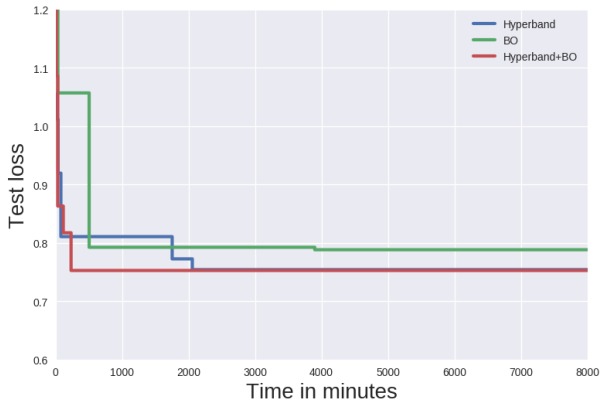

Figure 2: Loss of the best model found at a given time by each method.

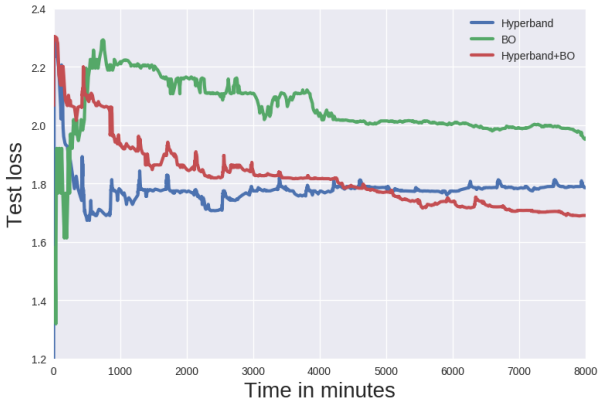

Figure 3: Running median of the loss of all tested models for each method.

[5] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Efficient hyperparameter optimization and infinitely many armed bandits," *CoRR*, vol. abs/1603.06560, 2016.

[6] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. Le, and A. Kurakin, "Large-Scale Evolution of Image Classifiers," *arXiv:1703.01041 [cs]*, Mar. 2017.

[7] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, A. Navruzyan, N. Duffy, and B. Hodjat, "Evolving Deep Neural Networks," *arXiv:1703.00548 [cs]*, Mar. 2017. arXiv: 1703.00548.

[8] B. Baker, O. Gupta, N. Naik, and R. Raskar, "Designing Neural Network Architectures using Reinforcement Learning," *arXiv:1611.02167 [cs]*, Nov. 2016.

[9] B. Zoph and Q. V. Le, "Neural Architecture Search with Reinforcement Learning," in *International Conference on Learning Representations*, 2017.

[10] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems 25*, pp. 2951–2959, 2012.

[11] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[12] D. R. Jones, "A Taxonomy of Global Optimization Methods Based on Response Surfaces," *J. of Global Optimization*, vol. 21, no. 4, pp. 345–383, 2001.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] F. Chollet, "Keras." `https://github.com/fchollet/keras`, 2015.

[15] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Technical report, University of Toronto*, 2009.

[16] H. Bertrand, M. Perrot, R. Ardon, and I. Bloch, "Classification of MRI data using deep learning and Gaussian process-based model selection," in *IEEE International Symposium on Biomedical Imaging*, 2017.