Bayesian optimization for weight initialization of gradient-based methods

Colin Majoor, Francesco Salvi, Martin Dahl
CS-439 Optimization for Machine Learning, EPFL

I. Introduction

In this project, we investigate if Bayesian optimization with Gaussian processes can be used in combination with Stochastic Gradient Descent (SGD) as a form of weight initialization, with applications to simple models with a small number of parameters. Traditionally, weights are initialized randomly by sampling from a weight space hypercube. Using gradient-based methods, this seldom lead to a global optimum, because the model loss functions are usually non-convex and contain several local minimas. With Bayesian optimization (BO), instead, a loss function can be surveyed before running SGD, allowing the starting point to be selected more strategically. In particular, we are interested in answering the following questions:

- **RQ1**: Can BO+SGD help in converging to the optimum, when it lies in a narrow long valley?
- **RQ2**: Can BO+SGD help in finding the global optimum, avoiding local ones?

For this purpose, we will analyze three small optimization problems by using SGD on top of weight initialization by Bayesian optimization: the Rosenbrock [1] and Rastrigin [2] functions, commonly used as a benchmark for optimization algorithms because of their peculiar characteristics that make them tricky to optimize [3], and small neural network applied on the Titanic dataset [4]. In all three cases, the results of this strategy will be compared to a weight initialization by random sampling, used as a baseline. In §II, we summarize Bayesian Optimization and the previous related works. Then, in §III we introduce our toy problems and explain our methodology. Finally, in §IV we present and discuss our results.

II. Background

A. Bayesian optimization

Bayesian optimization is a gradient-free optimization method, that uses a Bayesian statistical model to optimize a so-called *black-box function* f, usually expensive to evaluate, by subsequently evaluating a set of carefully-selected points [5], [6]. The method models the target f as a random function, initially placing a *prior distribution* over it and then updating it iteratively using the sampled points, getting thus a *posterior distribution*. Such distribution, finally, is used at each step to obtain an *acquisition function* that determines the next point to to test on f, balancing exploration and exploitation. The process is described schematically in Algorithm 1, adapted from *Peter I. Frazier's* [7].

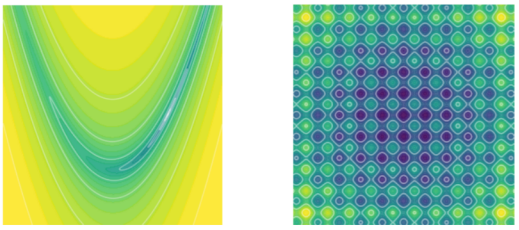
Three common choices for the acquisition function are the Upper Confidence Bound (UCB), the Expected Improvement (EI) and the Probability of Improvement (POI). UCB recommends points with the largest confidence bound, EI points with the largest expected improvement and POI points with the largest probability of improvement. Furthermore, these functions can be tuned to prioritize exploration or exploitation. An exploration-tuned acquisition function will take points potentially far from the current optimum, not always adhering to the main strategy, while an exploitation-tuned acquisition function which will try to leverage the current optimum, locally maximizing the function.

Algorithm 1 Basic pseudo-code for Bayesian optimization

```
Place a Gaussian process prior on f
Observe f at R points according to an initial space-filling
experimental design. Set n = R.
while n ≤ R + B do
    Update the posterior probability distribution on f
    using all available data.
    Let x_n be a maximizer of the acquisition function over
    x, where the acquisition function is computed using
    the current posterior distribution.
    Observe y_n = f(x_n).
    Increment n.
end while
return Point evaluated with largest f(x).
```

B. Related work

It proved difficult finding papers attempting to initialize weights using Bayesian optimization. One candidate is *Nadir Murru et al.* [8], using a kalman filter Bayesian approach. Results showed that the suggested method for weight initialization decreased the average number of required iterations to convergence compared to random weight initialization. In other approaches, such as in *Babak Shakibi et al.* [9], a subset of the model parameters are predicted, showing that there is redundancy among the weights and allowing to interpolate a majority of them. Similarly, *Michal Drozdzal et al.* [10] attempts to predict all weights using only a single forward pass. In all, these works can be seen as a part of meta-learning; learning to learn.



(a) Rosenbrock (b) Rastrigin

Figure 1: Plots of the Rosenbrock and Rastrigin functions. Darker means lower value.

III. Methods

A. Rosenbrock function

$$f(x, y) = (a - x)^2 + b(y - x^2)^2 \tag{1}$$

The Rosenbrock function [1] is shown in Figure 1a and defined in (1), with a and b parameters usually set to 1 and 100. The function is characterized by the presence of a narrow flat valley, easy to individuate but with a minimum that is difficult to reach. Therefore, it is a good working problem to study our **RQ1**.

The Rosenbrock is optimized using learning rates $1e-5$ and $1e-4$ for 1000 and 100 iterations of gradient descent respectively. The outcome is averaged over 1000 trials. For initialization, random is compared to Bayesian optimization with the UCB acquisition function set to high exploitation ($\kappa = 0.1$). 5 exploration and 5 exploitation iterations were performed. Results are seen in 2.

B. Rastrigin function

$$f(\mathbf{x}) = 2A + \sum_{i=1}^{2} \left[x_i^2 - A\cos(2\pi x_i)\right] \tag{2}$$

The Rastrigin function [2] is shown in Figure 1b and defined in (2), with $x_i \in [-5.12, 5.12]$ and usually $A = 10$. The function exhibits many local minima, with a global optimum in $\mathbf{x} = (0, 0)$ that is difficult to find, making it a good example for our **RQ2**. To test our hypothesis, we performed weight initialization by running $R = 2$ random steps and $B = 8$ bayesian steps, followed by 50 iterations of SGD with learning rate 0.001. As a comparison, the same framework is used with random initialization, both performing $R = 1$ or $R = 10$ steps. This whole procedure is repeated 10 000 times, keeping track of the final points where the algorithm converged and their respective losses.

C. Titanic + Neural network

The Titanic dataset [4] comes from a Kaggle competition with the goal of predicting the likelihood of survival of each passenger, using 8 main features. In our study, we keep only 3 features, corresponding to age, fare and gender (encoded with a dummy variable), while the outcome is represented by a binary variable corresponding to survival or death. Since Bayesian optimization works best when used in small dimensional problems, we implemented for this dataset a small fully-connected neural network with 3 input nodes, one hidden layer of size 3 and a single output. This small network already creates 16 parameters to optimize: 12 weights and 4 biases.

To investigate our **RQ2**, we initialize the weights of the network with $R = 5$ random steps and $B = 10$ bayesian steps, searching in the range $[-1/\sqrt{3}\ ;\ +1/\sqrt{3}]$. The network is then optimized with SGD for 1000 iterations with a learning rate of 0.001, minimizing the Binary Cross Entropy loss criterion. As a baseline, the same is done but with the standard weight initialisation from PyTorch [11], and both the techniques are carried out for 100 repetitions.

IV. Results and discussion

A. Rosenbrock

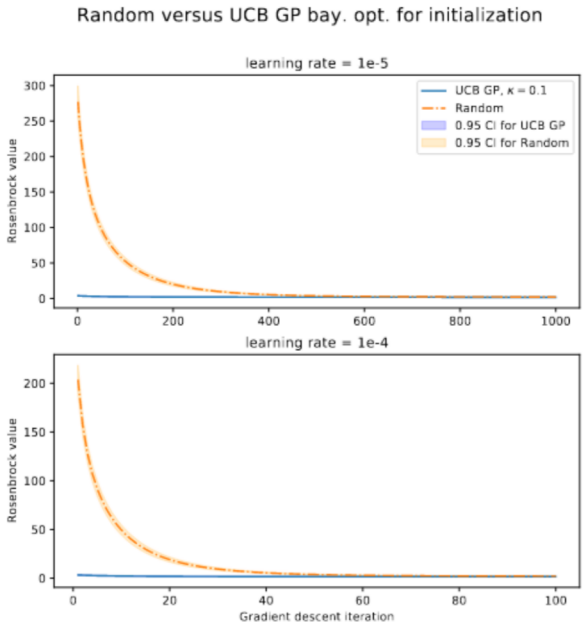
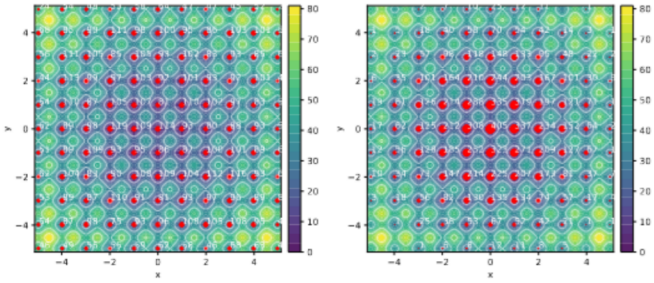


Figure 2: Optimizing the Rosenbrock function using gradient descent with initialization randomly versus with Bayesian optimization. Note specifically the difference in learning rate and the number of iterations needed for convergence. A 0.95 confidence interval is included but is difficult to see because of low variance over 1000 trials.

For optimizing the Rosenbrock function, two primary results



(a) $R = 1$, $B = 0$ (b) $R = 2$, $B = 8$

Figure 3: Results for the Rastrigin function. The size of the circles is proportional to the number of times the algorithm converged to each point, specified also by the labeling numbers.

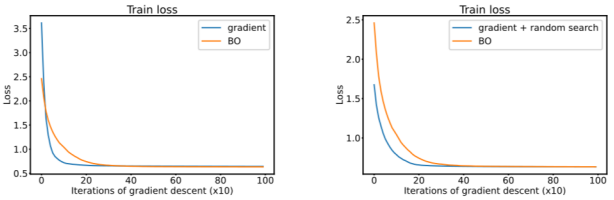
are presented in Figure 2. The main point to note is that for a lower learning rate, more gradient descent steps are required to reach convergence, but when initialization is done with Bayesian optimization a lot of this descent has already been performed. Thus, in a fashion the method becomes almost immune to the learning rate using Bayesian optimization. With a lower learning rate, the cost trade-off for using Bayesian optimization also becomes bigger, as more gradient descent steps are needed otherwise.

B. Rastrigin

The results for the Rastrigin function are shown in Figure 3. When standard random weight initialization is performed, with $R = 1$, the algorithm converges to final minima distributed pretty much uniformly on the whole domain of the function, except for the borders, reaching an average loss of 17.3 (std 11.3). Instead, when Bayesian optimization is used, the algorithm tends to converge more often to points towards the center of the space, where the global minimum lies and where local minima are overall better, reaching an average loss of 7.9 (std 6.7). While this seems to clearly favour Bayesian optimization, the results with an analogous number of random steps $R = 10$, here omitted for space reasons, are very similar or even a bit better, with an average loss of 6.8 (std 5.6). It seems, therefore, that the advantage in converging to a better optimum is overall an achievement of the greater number of steps rather than be specifically due to the Bayesian component.

C. Neural Network

The results on the optimization of the neural network are given in figure 4a. As expected, the loss for bayesian optimization starts out better, and it also ends on a better final loss. In our case we found that the loss was around 2 percent better. The p-value is 0.01 for our 100 instances. At first look, it seems that bayesian optimization is able to find a better loss.



(a) No random search (b) Random search

Figure 4: Results for the neural network. The blue curve is the train loss using only gradient descent. The orange curve is the train loss using bayesian optimization prior to gradient descent.

However, we should really compare bayesian optimization to random search. In figure 4b, we find that the loss for bayesian optimization starts out worse than random search, and the final loss is almost the same. There is a very slight but statistically insignificant difference, as the p-value is 0.85.

V. Conclusion

- **RQ1**: Comparing BO to using just one iteration of random initialization, BO + SGD does reach convergence in fewer iterations. The lower learner rate is used, the bigger is the gain of saved iterations using BO + SGD. However, this might not be a fair comparison since BO in this case runs ten iterations to find a point of initialization, while random only runs one.
- **RQ2**: This comparison is especially unfair when trying to avoid local optima, as the random search prior to the bayesian optimization could be a big advantage. When comparing our results for BO against SGD with some prior random searches, one finds that the results are very close, and the difference has no statistical significance. This confirms that for this research question and the data we used, BO does not help significantly in avoiding local optima.

References

[1] H. H. Rosenbrock, "An automatic method for finding the greatest or least value of a function," *The Computer Journal*, vol. 3, no. 3, pp. 175–184, Mar. 1960. [Online]. Available: https://doi.org/10.1093/comjnl/3.3.175

[2] L. A. Rastrigin, "Extremal control systems," *Theoretical Foundations of Engineering Cybernetics Series*, 1974.

[3] Wikipedia, "Test functions for optimization — Wikipedia, the free encyclopedia," http://en.wikipedia.org/w/index.php?title=Test%20functions%20for%20optimization&oldid=1081181076, 2022, [Online; accessed 03-June-2022].

[4] Kaggle.com, "Kaggle titanic dataset," accessed: 2022-05-01. [Online]. Available: https://www.kaggle.com/c/titanic

[5] J. Močkus, "On bayesian methods for seeking the extremum," in *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*. Springer Berlin Heidelberg, 1975, pp. 400–404.

[6] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012.

[7] P. I. Frazier, "A tutorial on bayesian optimization," 2021.

[8] N. Murru and R. Rossini, "A bayesian approach for initialization of weights in backpropagation neural net with application to character recognition," *Neurocomputing*, vol. 193, pp. 92–105, Jun. 2016.

[9] M. Denil, B. Shakibi, L. Dinh, M. A. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., vol. 26. Curran Associates, Inc., 2013.

[10] B. Knyazev, M. Drozdzal, G. W. Taylor, and A. Romero, "Parameter prediction for unseen deep architectures," in *Advances in Neural Information Processing Systems*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035.