Technical University of Denmark

Mini Project 1: Bayesian Optimization

02463 - Active Machine Learning & Agency

Group 15

Andreas Søndergaard - s214635

Lasse Vinther Rosshaug - s193562

Martin Mohr Balle - s214646

Mathias Correll Damsgaard - s214647

2nd march 2023

1 Abstract

This paper explores the use of Bayesian optimization for hyperparameter tuning in a convolutional neural network for the Fashion MNIST dataset. The study aims to investigate whether Bayesian optimization outperforms a random sampling approach in terms of finding the best values for the two hyperparameters: learning rate and dropout rate. Through our analysis, we found no significant difference in the model's performance between using the two approaches for tuning the hyperparameters.

2 Introduction

The choice of hyperparameters can significantly impact the performance of a machine learning model, and finding the optimal hyperparameters can be a time-consuming and computationally expensive task.
Bayesian optimization is a tool to reduce the computational cost of finding hyperparameters by predicting the best places to search for them as opposed to picking hyperparameters at random or testing all possible combinations.
To explore the effects of bayesian optimization we will be using the Fashion MNIST dataset which consists of grayscale 28 x 28 pixel images of clothing items with labels from 10 classes. A machine learning model will be built to classify the images, which will include hyperparameters that may be tuned to improve performance. We expect that tuning the hyperparameters with bayesian optimization will result in better performance than a random sampling approach.

3 Methods

We worked with the Fashion-MNIST dataset, which consists of 70,000 pictures [1]. We used a convolutional neural network to classify these pictures with hyperparameters optimized by bayesian optimization, and we chose to limit the amount of data we used. Therefore, our train data size ended with 3,000 pictures and the test set with 1,000. As we made these splits by random, we defined a random state for reproducibility.
The BO was done by scikit-optmimize's (also skopt) gp_minimuze function [2], which minimizes a problem, so our objective function was 1 - accuracy. The hyperparameters we optimized for in our CNN were the learning rate and dropout rate. These we respectively defined in real domain between $[10^{-4}, 10^{-2}]$ and $[0\%, 50\%]$. We chose to work with continuous parameters as bayesian optimization searches through a continous landscape, and we wanted to match that. We also trained our CNN for 5 epochs each time, as to not give the model too much time to convert towards optimal weights each time, and hence hopefully make initial optimization more relevant. After we had generated our random sampling, we used the first data point as the initial evaluation point for BO, so that the two method would have a common starting point. The kernel our BO used is a standard Matern 5/2 with a 1 noise level white noise kernel added to it. The acquisition function was gp-hedge, and it used either probability of improvement, expected improvement and upper confidence bound based on a probability of each function. This probability was updated each time with a softmax weighted by how well they performed on the previous run. Since skopt minimized the function, the acqusistion functions were actually defined as the negative PI and EI and the lower confidence bound. The exploration parameters was set at a default value of $\xi = 0.01 \wedge \kappa = 1.96$.
We let both approaches run for 30 iterations and so collected as many different accuracies, which we could use to compare their performance. The optimal hyperparameters from BO was then used again to train a model that predicted labels for new unseen data points to get an unbiased generalisation accuracy. 1000 pictures from the data set, separated from the training/test data, was used used for this test. On this data set we also tried with random sampling again as wanted something to compare this accuracy to.

4 Results

By inspecting Figure 1 we see that the results appear very similar despite the approach used. Bayesian optimization seems to fluctuate a bit more, after it quickly found a minimum. This could very well be

because it only had the one common starting point to make the distribution on, and so it needs to search a lot to look for new global minimum. Random sampling also seems to fluctuate a bit, but not as much.

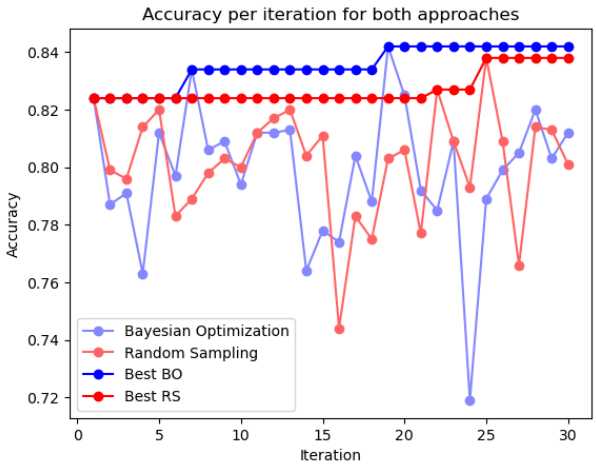

Figure 1: Plot of accuracy for each iteration of each approach and their respective best during the iterations.

In the end random search finds hyperparameters that can produce a higher accuracy than bayesian optimization. Random sampling results in $LR = 4.9 \cdot 10^{-3} \wedge DR = 4.8\% \Rightarrow 83.8\%$ versus Bayesian optimization giving $LR = 2.3 \cdot 10^{-3} \wedge DR = 11.0\% \Rightarrow 84.2\%$
We have calculated a 95% confidence interval based in a t distribution for the accuracies of random sampling hyperparameters on the extra generalization test set. This is summarised in Table 1 with the accuracy from using the optimal hyperparameters from bayesian optimization.

Approach	**Accuracy**
95% CI - random sample	[81.3% ; 82.7%]
Generalization accuracy - bayesian optimization	82.6%

Table 1: Table over calculated accuracies and confidence intervals for each approach.

We can see that the accuracy from bayesian optimization just barely lie inside the confidence interval for random sampling the hyperparameters.

5 Discussion

Even though the results are only based on a single run of the two approaches, and more tangible results could have been obtained with cross-validation, where the effect of random fluctuations would be smaller, the results still indicate that there is no significant difference between using Bayesian optimization and a simple random sample for this specific task of tuning the learning rate and dropout rate of the CNN. We didn't go all in and do cross-validation because of limited time, and the computation time would have been too long.
Multiple different factors might be the reason for why this is the case. First of all, these two hyperparameters are probably not the most influential in determining the CNN's accuracy. Other hyperparameters like the architecture of the CNN, including the number and size of layers, and the type of activation functions, might have a more significant impact on the overall performance of the model. These are however discrete number and so we chose not to optimize them as descirbed earlier.
We chose to limit the amount of data we used on a lot going from a total of 70.000 pictures to 5.000 randomly

chosen pictures to try and make the learning rate and dropout rate more relevant hyperparameters. In hindsight it might still have been a relatively too large data set to train the model on, which was 3.000. Therefore, it may not be so surprising our two hyperparameters don't significantly affect the performance of the model. In general, when the data set is large the learning rate is less likely to cause issues such as overfitting or slow convergence. Similarly, when we have a large data set, the regularization effect of the dropout rate may also be less critical. Hence in the future one might want to limit the data set more to see a significant effect of the hyperparameters.

6 Conclusion

In conclusion, our experiment indicates that there is no significant difference between using Bayesian optimization and a simple random search for tuning the learning rate and dropout rate of a CNN.

7 References

[1] Zalando. Fashion mnist. https://www.kaggle.com/datasets/zalando-research/fashionmnist.

[2] Scikit-Optimize. skopt.gp_minimize. https://scikit-optimize.github.io/stable/modules/generated/skopt.gp_minimize.html.

8 Learning Outcome

During the work with this mini project on bayesian optimization we have learned:

- How some hyperparameters may not always be suited to Bayesian optimization
- How to use and apply Bayesian optimization in a real life setting
- How to implement Bayesian optimization in python
- Gotten a deeper understanding of the GP-hedge acquisition function