Implementing Early Stopping for Tree Parzen Estimator Optimization

Nikhil Saggi [1]

Abstract

Hyperparameter optimization is an important step of the training process because it controls the peak achievable performance in most learning algorithms. But, the process can be expensive because contemporary methods often require input on when to trade off accuracy for cost efficiency, or vice versa. This paper focuses on marginal improvements in accuracy per epoch for Bayesian hyperparameter optimization using a sequential-based optimizer called Tree-structured Parzen Estimators. With this information, we construct and evaluate an early stopping scheme to terminate the algorithm when future improvements are likely minimal. With convolutional neural networks of varying complexity, our early stopping scheme decreases wall-clock runtime by 62.2% to 78.3%, while increasing validation error by 0% to 3.4%. This mechanism proves that a sensible number of epochs can be automatically chosen for large models and suggests that a more versatile mechanism could be developed for all models.

1. Introduction

When training a machine learning model, the selection of appropriate hyperparameters is necessary to control to model behavior. Hyperparameter optimization is the step in model creation where values over the hyperparameter space are sampled, and the values minimizing the objective of the model are selected to configure the model. The process of identifying an optimal set of hyperparameters, however, can add large overhead to the model training process. Additionally, many hyperparameters may exist in a combined space where their values co-vary with one another, which means finding an optimal configuration in a larger hyperparameter space can take greater time; for brute-force methods such as Grid search, the time increase will be exponential (LeCun et al., 1998). Random search is an alternate method that randomly selects points in the hyperparameter space, which prevents exponential runtime but no longer guarantees an optimal configuration is found (Bergstra & Bengio, 2012).

A more favorable approach may be Bayesian optimization, which involves building a probability model, called a surrogate function, for the objective of the original model (Močkus, 1975). The model is run incrementally to update the posterior of the surrogate function, and eventually an accurate estimate the optimal hyperparameters can be made. This process improves upon its predecessors because the hyperparameters that are tested with the model are informed by past choices. This minimizes the necessary number of runs of the model at the cost of greater time spent selecting hyperparameters to test (Snoek et al., 2012). In applications such as deep neural networks, this trade-off is favorable because updating the surrogate function takes less time than running the model. Using Bayesian Optimization requires a domain of hyperparameters to be defined, and an initial probability distribution of the ideal values. The estimator to build a surrogate function can vary, but a common one for such problems is the Tree-structured Parzen Estimator, or TPE (Bergstra et al., 2011). TPE estimates $P(X|y)$, or the probability of a set of hyperparameters X yielding a particular output of the loss function y. The estimator keeps a threshold output $y*$ and individually tracks probabilities of configurations yielding outputs above $y*$ and below $y*$, denoted $g(X)$ and $l(X)$, respectively. Then, the hyperparameter configuration that maximizes $l(X)$ and minimizes $g(X)$ is run with the model, as it is the most likely to minimize the objective. Finally, the computed objective is used to update $g(X)$ and $l(X)$. One iteration of finding hyperparameters and updating the probabilities is defined as an epoch.

TPE is a type of sequential-based model global optimizer (SMBO). Such algorithms operate in a looped structure, which requires a decision be made on the maximum number of iterations (Dai et al., 2019). After sufficient epochs, TPE is expected to converge to a minimal objective value; however, the number of iterations will vary with factors including the size of the hyperparameter space and initial probability distribution of hyperparameters (Bergstra et al., 2011). In that sense, the setup introduces a 'secondary' hyperparameter which is the number of epochs. Setting this parameter introduces a trade-off between runtime and accuracy because having too few epochs will prevent the discovery of an optimal configuration, and having too many epochs will run the model repeatedly for pessimal improvements in the objective.

In this paper, we balance concerns for accuracy and

runtime by measuring the efficiency of TPE per epoch and observing the marginal improvements to accuracy that occur. In expectation, this will decrease with successive epochs while the runtime will increase at a constant rate. The dual consideration of hardware and statistical efficiency may indicate a hyperparameter configuration that minimizes the objective will not justify the computation required for its identification. So, we implement and evaluate a system akin to early stopping in neural networks, where TPE will terminate if the algorithm cannot further minimize the objective after a certain number of epochs.

2. Related work

Other studies have considered approaches to determining epoch settings for SMBO algorithms. (Snoek et al., 2015) explores the use of neural networks as an SMBO algorithm to combat runtime scaling with the number of observations. (Klein et al., 2017) proposes an algorithm called FABOLAS that adds training set size to the optimization problem, which permits SMBO algorithms to discover an optimal configuration quicker. The setup in these papers could function congruently to the methods we develop and could further decrease hyperparameter optimization runtime.

Since the goal of reducing the number of epochs is to limit how often the model is run, an alternate approach to the problem is to develop a cheap-to-evaluate surrogate that maps the hyperparameter space to the objective, thus eliminating the necessity to run the model to find optimal hyperparameters. (Eggensperger et al., 2015) develops such a method, where regressions are run on evaluations of the model for given hyperparameter settings. This requires more model runs at the start and, depending on the data required by the model, could be slower than traditional SMBO methods including our approach. However, if data already exists of past model runs with various hyperparameters, then this approach can compute a surrogate without running the model since, unlike Bayesian approaches, it does not require hyperparameters in runs to be selected based on likelihood functions.

3. Experiments and Results

The methodology of the paper is divided into two parts. The first section will focus on the improvements per epoch of TPE, and how the marginal improvements to accuracy vary across execution. The second section will design and test a model for early stopping based on the observations from the first section. All experiments were run using a 12GB NVIDIA Tesla K80 single GPU.

3.1. Measuring returns per epoch

First, we would like to identify the characteristics of a model that benefits the most from using Bayesian hyperparameter optimization. To do so, we compare the minimum attained objective value per epoch with TPE versus Random search, which will function as a baseline. This is because we assume dimensionality of hyperparameters is a primary concern, so Random Search outcomes stochastically dominate Grid search outcomes.

To start, we consider an OLS model which has a small parameter space and low runtime. Twenty models, with random β_1 and β_0 parameters, were generated, along with 500 data points perfectly fitting the created curve. Both Random search and TPE were trained with the model for 2000 epochs to estimate β_1 and β_0. The lowest obtained loss per epoch for a representative iteration is shown in Figure 1. In all examples, Random search and TPE converged to β_0 and β_1 parameters within 2% of the actual values, and each approach reached its respective minimum loss consistently around 1350 epochs. However, in terms of utility it is important to consider when an 'acceptable' loss was attained by each approach, which we define as a loss within 5% of its eventual optimal. On average, Random search discovered an acceptable configuration by epoch 229, whereas TPE reached this point by epoch 327. This indicates that Random search converges to an optimal quicker for models with smaller hyperparameter space. Furthermore, the wall-clock time for Random search was 0.11 ms per epoch and 23 ms per epoch for TPE, meaning Random search performed about 200x faster than TPE due to the low speed of computing OLS relative to the time needed to train a Parzen estimator.

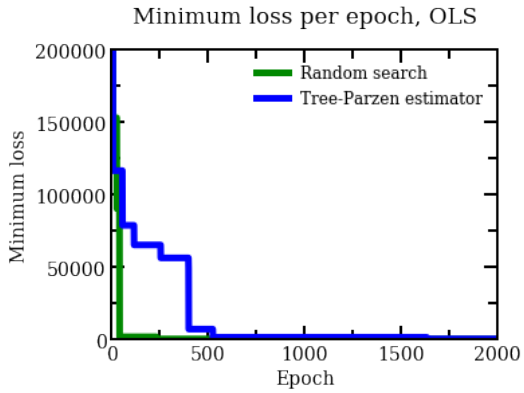

Figure 1. Lowest obtained sum-of-squares loss per epoch using Random search and TPE optimizers to find OLS parameters.

Since the benefits of Bayesian optimization are greatest with larger models, we next observe the performance of each optimizer on convolutional neural networks. Our

evaluation occurs over four hyperparameters: batch size, learning rate, momentum multiplier, and network size. The hyperparameters were estimated five times each by Random search and TPE estimators, with a random initial seed for each of the five runs. Each time the model was run by the optimzer, it ran for five iterations (meaning, the weights were updated five times). Identical seeds were used for each Random search and TPE run pair to allow for comparison. A factor to consider is that the network architecture as a whole is a multi-dimensional hyperparameter. Rather than parameterize every component of the network, we ran two separate experiments with a 'well-designed' network and a 'poorly-designed' network, as defined by the attainable validation error with a predetermined ideal set of hyperparameters and limited number of iterations. We believe this is a sensible decision because, in practice, control of the model structure is typically given to the user, and we would like to observe an optimizer that is robust in finding hyperparameter configurations for strong and weak structures.

We first test a well-designed network for the MNIST dataset, which can obtain 99.47% accuracy with optimal hyperparameters (Tabik et al., 2017). Random search and TPE were trained with the model for 100 epochs, and the lowest obtained validation error per epoch for a representative iteration is shown in Figure 2. On average, Random search discovered a configuration within 5% of its eventual optimal after 35 epochs, and TPE found such a configuration after 24 epochs. So, unlike with OLS, TPE converges to an optimal quicker than Random search when the parameter space is greater. Additionally, the wall-clock time for Random search was 40.37 sec per epoch and 38.39 sec per epoch for TPE, which are approximately similar now that model runtime comprises most of the execution time.

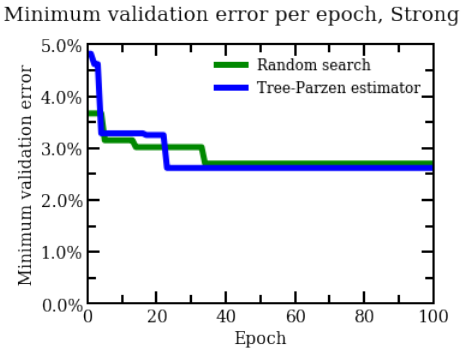

Figure 2. Lowest obtained validation error per epoch using Random search and TPE optimizers to find CNN hyperparameters on MNIST dataset.

We then test a poorly-designed network over the CIFAR-10 dataset, which will converge to 27.88% validation error after approximately 30,000 neural network iterations (He et al., 2015). Again, Random search and TPE were trained with the model for 100 epochs, and we show the lowest validation error for each epoch in Figure 3. Random search found a configuration within 5% of its eventual optimal after 24 epochs, whereas TPE took 4 epochs. The minimum error curves for this test, however, were closer than with the MNIST model, likely because this model converges slower, so the speed in convergence may start to differ more with more iterations. The wall-clock times were 14.24 sec per epoch for Random search and 14.53 sec per epoch for TPE, which is a negligible difference.

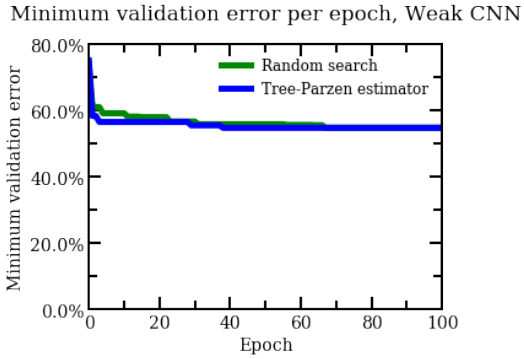

Figure 3. Lowest obtained validation error per epoch using Random search and TPE optimizers to find CNN hyperparameters on CIFAR-10 dataset.

3.2. Early stopping model

An observation from the previous experiment is that an 'acceptable' objective value ($\leq$ 5% away from eventual optimal) was not always obtained from the optimizer after a consistent number of epochs. In addition to varying based on the shape of the objective function and hyperparameter space size, the necessary number of epochs also varied when repeating each trial with a different random seed. To control this secondary hyperparameter, we tested an early-stopping method for TPE that will terminate if a new minimum objective value has not been found in the last 10% of epochs. This termination may only occur once 20% of the total number of epochs have completed. The former condition serves to detect when TPE has discovered a hyperparameter configuration close enough to the optimal that it is unable to easily reach a higher maximized expectation with a new configuration. The second condition is to prevent a premature termination in objective functions that are less convex. To test the effectiveness of this method, we perform the same TPE tests as before but with the early stopping mechanism.

A sample result of early stopping with the OLS model is shown in Figure 4. On average, the OLS algorithm terminated after 505 epochs, but the loss at this point was approximately 676% higher than the eventually attainable

minimum loss if the algorithm had kept running. By skipping 1495 epochs, the wall-clock time decreased by about 36.2%. For smaller models, the decreased runtime does not seem to justify the loss in accuracy obtained from early stopping.

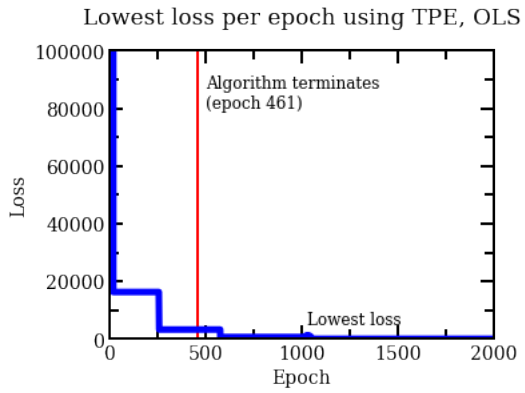

Figure 4. A sample run of TPE optimizer for OLS. The blue dot shows where the lowest loss would be obtained if the optimizer ran for all epochs, and the red line shows where the algorithm stopped.

A sample result of early stopping with the well-designed network is shown in Figure 5 (top). On average, the algorithm terminates after 34 epochs, and, in every run, this was the minimal obtained validation error after all 100 epochs. By stopping early, the wall-clock time decreased by 62.2%. A similar result is seem with the poorly-designed network, shown in Figure 5 (bottom). On average, the algorithm terminates after 28 epochs, and the validation error at that point was 3.4% higher than the eventual minimum. By stopping early, the wall-clock time decreased by 78.3%. With both examples, early stopping functions as intended, since the optimal hyperparameter configuration yields a validation error within 5% of the possible minimum and decreases wall-clock time by a noticeable amount.

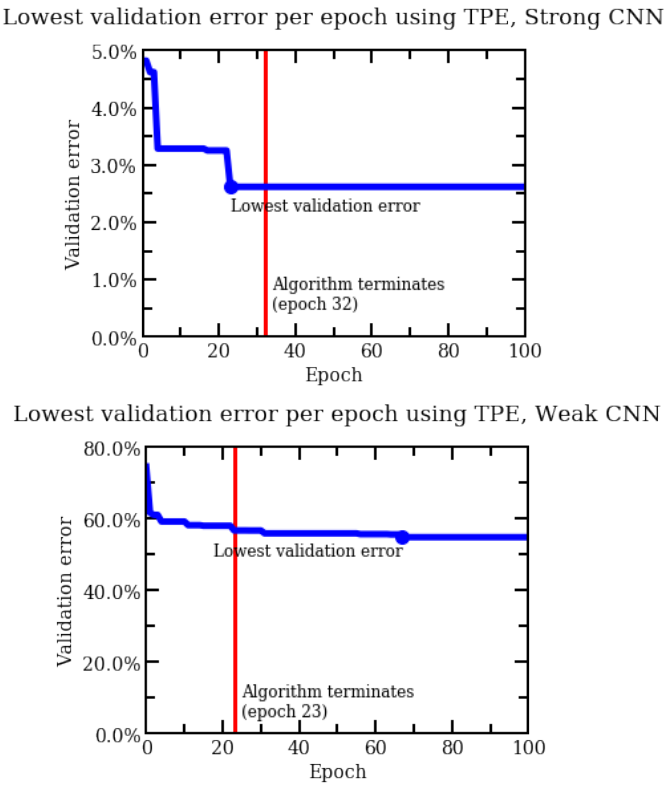

Figure 5. A sample run of TPE optimizer for CNNs with MNIST (top) and CIFAR-10 (bottom) datasets. The blue dot shows where the lowest validation error would be obtained if the optimizer ran for all epochs, and the red line shows where the algorithm stopped.

4. Impact and Future work

This paper highlights the value of observing marginal improvements per epoch for TPE, since doing so provides information of proximity to the optimal hyperparameter configuration. The early stopping mechanism demonstrates this relation with larger models such as convolutional neural nets, where early stopping substantially reduced runtime with minimal impact on accuracy. The method did not work for smaller models such as OLS, possibly because the accuracy of the parameters is has more importance. And, considering that the model runtime is lower, it does not make sense to run the model less frequently and sacrifice accuracy. A shortcoming with this early stopping mechanism is that the specified thresholds for stopping were fixed. We suspect a more robust model could be developed if the thresholds are conditioned on some parameters of the model to indicate its complexity – one such field that could suffice is the training time.

Early stopping could also be improved by valuing the returns for each epoch. The current implementation will treat any decrease in objective function as an indicator to not terminate. But, if TPE continues to identify new minima of negligible significance, the algorithm will never terminate. At a certain point, the small decreases in the objective may be outweighed by the energy cost and runtime. A stopping protocol could consider whether decreases occur at a small enough percentage for the algorithm to still terminate.

This study focused on improving the number of necessary epochs for TPE through early stopping, which is a means to evaluate the posterior. An analogous strategy would be to consider initial factors that control the necessary number of epochs, namely the initial probability distributions of the hyperparameters. By adjusting the prior for the optimization, fewer epochs may be required to reach a similar optimum.

References

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null): 281–305, February 2012. ISSN 1532-4435.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, pp. 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc. ISBN 9781618395993.

Dai, Z., Yu, H., Low, B. K. H., and Jaillet, P. Bayesian optimization meets Bayesian optimal stopping. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1496–1506, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/dai19a.html.

Eggensperger, K., Hutter, F., Hoos, H. H., and Leyton-Brown, K. Efficient benchmarking of hyperparameter optimizers via surrogates. In *AAAI*, pp. 1114–1120, 2015. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9993.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition, 2015.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets. In Singh, A. and Zhu, J. (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 528–536, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR. URL http://proceedings.mlr.press/v54/klein17a.html.

LeCun, Y., Bottou, L., Orr, G. B., and Müller, K.-R. Efficient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pp. 9–50, Berlin, Heidelberg, 1998. Springer-Verlag. ISBN 3540653112.

Močkus, J. On bayesian methods for seeking the extremum. In Marchuk, G. I. (ed.), *Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974*, pp. 400–404, Berlin, Heidelberg, 1975. Springer Berlin Heidelberg. ISBN 978-3-540-37497-8.

Snoek, J., Larochelle, H., and Adams, R. P. Practical bayesian optimization of machine learning algorithms, 2012.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M. M. A., Prabhat, and Adams, R. P. Scalable bayesian optimization using deep neural networks, 2015.

Tabik, S., Peralta, D., Herrera-Poyatos, A., and Herrera, F. A snapshot of image pre-processing for convolutional neural networks: case study of mnist. *International Journal of Computational Intelligence Systems*, 10:555–568, 2017. ISSN 1875-6883. doi: https://doi.org/10.2991/ijcis.2017.10.1.38. URL https://doi.org/10.2991/ijcis.2017.10.1.38.