Towards Reproducible Neural Architecture and Hyperparameter Search

Aaron Klein
University of Freiburg
kleinaa@cs.uni-freiburg.de

Eric Christiansen
Google
ericmc@google.com

Kevin Murphy
Google
kpmurphy@google.com

Frank Hutter
University of Freiburg
fh@cs.uni-freiburg.de

Abstract

Recent advances in neural architecture and hyperparameter search demand tremendous computational resources which makes it almost impossible to reproduce experiments. We argue that this hinders the progress in this subfield since new methods can not be thoroughly compared to already existing methods. In this work, we generated a new benchmark for neural architecture search and hyperparameter optimization which is based on tabular data for a feed forward neural network. Each function evaluation is just a simple table look up and thus takes only milliseconds but mimics the true underlying optimization problem. Furthermore, we analyze the properties of this benchmark and compare a range of state-of-the-art neural architecture and hyperparameter search methods.

1 Introduction

Despite the tremendous success achieved by deep neural networks in the last few years (Krizhevsky et al., 2012; Sutskever et al., 2014) using them in practice remains challenging due to their sensitivity against a large amount of hyperparameters and architectural choices. Even experts often find the right setting to train the network successfully only by trial-and-error.

There has been a recent line of work in hyperparameter optimization (HPO) (Snoek et al., 2012; Hutter et al., 2011; Bergstra et al., 2011; Hazan et al., 2018; Li et al., 2017) and neural architecture search (NAS) (Baker et al., 2017; Zoph and Le, 2017; Real et al., 2017; Elsken et al., 2018) that tries to automate this process by casting it as an optimization problem. However, since each function evaluation consists of training and evaluating a deep neural network, running these methods can take several days even when using hundreds of GPUs.

Probably due to the high computational demands there is a certain lack of rigorous comparison between methods. The goal of this work is to facilitate a better empirical evaluation of HPO and NAS methods by providing a benchmark that is cheap to evaluate but still presents a realistic use case. We believe that this benchmark provides an easy and efficient way to conduct and reproduce experiments for neural architecture and hyperparameter search.

We collected a large grid of configurations of a feed forward neural network (see Section 2) for regression. Based on the gathered data, we give an in-depth analysis of the importance of hyperparameters and architectural choices, as well as the properties of the optimization problem (see Section 3). Finally, we compare a variety of well-known HPO and NAS methods from the literature, such as Bayesian optimization, evolutionary algorithms, compressed sensing techniques, a bandit based method and random-search (Section 4).

2nd Reproducibility in Machine Learning Workshop at ICML 2018, Stockholm, Sweden.

Table 1: The hyperparameter configuration space for the fully connected neural network architecture.

Hyperparameter	Choices	Best Configuration
Initial Learning Rate	$\{0.0005, 0.001, 0.005, 0.01, 0.05, 0.1\}$	0.0005
Batch Size	$\{8, 16, 32, 64\}$	32
Learning Rate Schedule	$\{\text{cosine}, \text{fix}\}$	cosine
Activation Function Layer 1	$\{\text{relu}, \text{tanh}\}$	relu
Activation Function Layer 2	$\{\text{relu}, \text{tanh}\}$	relu
Number of Units Layer 1	$\{16, 32, 64, 128, 256, 512\}$	256
Number of Units Layer 2	$\{16, 32, 64, 128, 256, 512\}$	256
Dropout Layer 1	$\{0.0, 0.3, 0.6\}$	0.3
Dropout Layer 2	$\{0.0, 0.3, 0.6\}$	0.0

2 Tabular Benchmark

In order to construct a benchmark, we used the UCI (Lichman, 2013) year prediction dataset, which is a subset of the Million Song Dataset (Bertin-Mahieux et al., 2011). In total, the dataset contains 515 345 data points with 90 features and we used 324 600 data points for training, 139 115 for validation and 51 630 for testing as described by Ruyu and Jiaying (2017). The data set allows us to train neural networks on CPU rather than GPUs and hence we can afford to run more configurations. Besides that, regression based on featurized data is an interesting task for neural networks since they scale much better with the number of data points than other methods such as for instance Gaussian processes.

As base architecture, we used a two layer feed forward neural network followed by a linear output layer on top. The configuration space (denoted in Table 1) only includes a modest number of 4 architectural choice (number of units and activation functions for both layers) and 5 hyperparameters (dropout rates per layer, batch size, initial learning rate and learning rate schedule) in order to allow for an exhaustive evaluation of all the 62 208 configurations resulting from discretizing the hyperparameters as in the table. We encode numerical hyperparameters as ordinals and all other hyperparameters are coded as categorical. Each network was trained with Adam (Kingma and Ba, 2014) for 100 epochs, optimizing the mean absolute error as was done by Ruyu and Jiaying (2017). We repeated the training of each configuration 4 independent times with a different seed for the random number generator and recorded the mean performance for each run. Generating this benchmark in total took 12014 CPU days. The dataset, as well as the code to reproduce the experiments, are publicly available at `https://github.com/automl/nas_benchmarks`.

3 Analysis

We now analyze the generated dataset through the lens of HPO and NAS to obtain a deeper understanding of the properties of the benchmark. If not stated otherwise, we always consider for each configuration the average metric (such as test error or training time) over the 4 repetitions.

3.1 Level of Difficulty

To obtain a better understanding of how difficult it is to optimize this benchmark, we computed the empirical cumulative distribution of the final test error and training runtime of all configurations (see Figure 1). One can see that only roughly 3% of all configurations achieve a final test error that is lower than 7. Furthermore, it seems that the dataset contains many outliers, i. e. configurations that achieve an error that is multiple orders of magnitudes higher than the average. Maybe less surprisingly, also the training runtime varies dramatically ranging from a few minutes to almost 9 hours (see Figure 1 middle).

Due to the stochastic nature of the training process, evaluating the same configuration multiple times will lead to slightly different results. We estimated this observation noise for each configuration by computing the standard deviation between the repetitions of the final test performance. As can be seen in Figure 1 (right) the observation noise is heteroscedastic, which means that different configurations come with a different noise level.

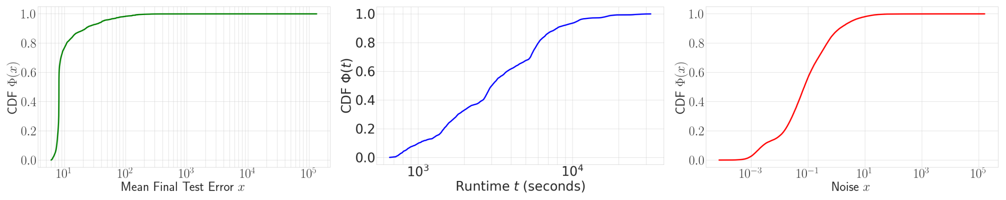

Figure 1: The empirical cumulative distribution of all configurations for the final test performance (left), the training runtime (middle) and the observation noise (right).

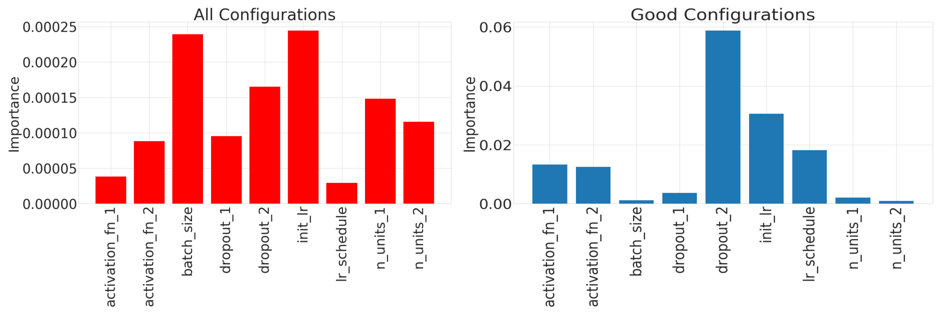

Figure 2: Hyperparameter importance based on fANOVA package for all configurations (left) and of the subspace of configurations that achieve a better performance than 7 (right).

3.2 Hyperparameter Importance

To analyze the importance of hyperparameters, assessing the change of the final error with respect to changing a single hyperparameter at a time we used the fANOVA tool developed by Hutter et al. (2014). It quantifies the importance of a hyperparameter by marginalizing the error obtained by setting it to a specific value over all possible values of all other hyperparameters. The importance of a hyperparameter is then the variation in error that is explained by this hyperparameter. In the default setting the fANOVA fits a random forest model on the observed function values in order to compute the marginal predictions. However, since we already evaluated the full configuration space, we do not even need to use a model and can compute the integrals directly.

As it can be seen in Figure 2 (left), on average across the entire configuration space, the initial learning rate and the batch size obtained the highest importance value. However, the importance of individual hyperparameters is very small due to a few outliers with very high errors, which only happen for a few combinations of several hyperparameter values.

A better estimate of hyperparameter importance in a region of the configuration space with reasonable performance can be obtained by capping validation errors before applying fANOVA; Figure 2 (right) shows the results of this procedure with errors capped at a maximum of 7. This shows that in this more interesting part of the configuration space, the dropout rate in the second layer is the most important parameter, followed by the learning rate and the learning rate schedule. Interestingly, the dropout rate in the first layer is much less important.

4 Comparison

In this section we use the generated benchmark to evaluate different HPO and NAS methods. To mimic the randomness that comes with evaluating a configuration, in each function evaluation we randomly sample one of the four performance values. To obtain a realistic estimate of the wall-clock time required for each optimizer, we accumulated the stored runtime of each configuration the optimizer evaluated. We do not take the additional overhead of the optimizer into account since it is negligible compared to the training time of the neural network. After each function evaluation we estimate the incumbent as the configuration with the lowest observed error and compute the regret between the incumbent and the globally best configuration. We performed 500 independent runs of each method and report the mean and the standard error of the mean.

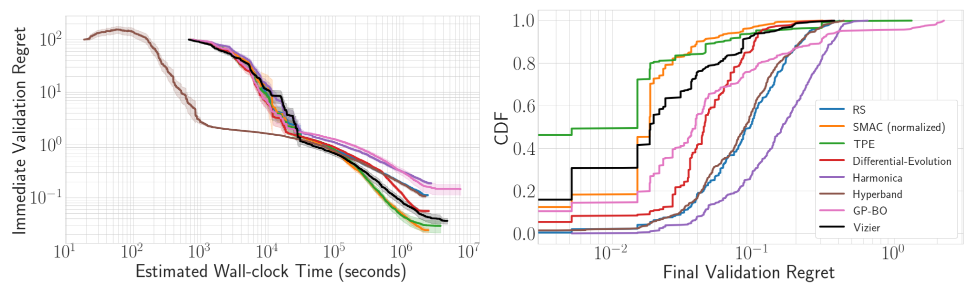

Figure 3: On the left we plot the validation performance of the incumbent of each method. On the right we show the empirical cumulative distribution of final validation regret of all methods.

4.1 Performance over time

We compared the following HPO and NAS methods from the literature (see Figure 3): random search, SMAC (Hutter et al., 2011)[1], Tree Parzen Estimator (TPE) (Bergstra et al., 2011), differential evolution (Storn and Price, 1997), Vizier (Golovin et al., 2017), the compressed sensing technique Harmonica (Hazan et al., 2018), Gaussian process based Bayesian optimization (GP-BO) (Snoek et al., 2012)[2] and Hyperband (Li et al., 2017).

Hyperband achieved a reasonable performance relatively quickly but it converged to simple random search. The two Bayesian optimization methods, SMAC and TPE worked as well as random search in the beginning but started to perform superior to all other methods once they obtained a meaningful model. Differential evolution and Vizier needed more time than TPE and SMAC to outperform random search; however, they were not able to achieve the same final performance as the other two methods. Harmonica did not work better than random search on this benchmark; we assume that due to the higher order correlation between hyperparameters (see Section 3), the benchmark does not fit Harmonica's assumptions and cannot be model with sparse and low degree polynomials. Also GP-BO has problems on this benchmark, which is to be expected since Gaussian processes tend to work best for continuous input spaces whereas our configuration space is discretized. We note that running the same evaluation for all these optimization techniques on the real benchmark would have taken 169.6 CPU years.

4.2 Robustness

Besides achieving good performance, we argue that robustness plays an important role in practice for HPO and NAS methods. Figure 3 (right) shows the empirical cumulative distribution of the regret for the final incumbent across all 500 runs of each method. Interestingly, even though SMAC achieves the lowest average performance only less than 20% of the runs achieve a final regret below 10^{-2}, whereas for TPE roughly half of all runs achieved a regret that is lower than 10^{-2}. This is even more dramatic for GP-BO, for which the top 20% of all runs are roughly on par with SMAC but the mean performance is worse than random search (see Figure 3 left).

5 Conclusions

We presented a new tabular benchmark for neural architecture and hyperparameter search that is cheap to evaluate but still recovers the original optimization problem, enabling us to rigorously compare various methods from the literature. Based on the data that we generated for this benchmarks, we had a closer look at the difficulty of the optimization problem and the importance of different hyperparameters.

In future work we will generate more of these benchmarks for other architectures and datasets. Ultimately, we hope that such benchmarks will help the community to easily reproduce experiments and evaluate new developed methods without spending enormous GPU resources.

[1]We used SMAC3 from `https://github.com/automl/SMAC3`

[2]We used the implementation from Klein et al. (2017)

6 Acknowledgements

This work has partly been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant no. 716721. Frank Hutter also gratefully acknowledges a Google Faculty Research Award. The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG.

References

Baker, B., Otkrist, G., Nikhil, N., and Ramesh, R. (2017). Designing neural network architectures using reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

Bartlett, P., Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors (2012). *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NIPS'12)*.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NIPS'11)*, pages 2546–2554.

Bertin-Mahieux, T., Ellis, D., Whitman, B., and Lamere, P. (2011). The million song dataset. In *Ismir*, volume 2, page 10.

Elsken, T., Metzen, J. H., and Hutter, F. (2018). Multi-objective architecture search for cnns. arXiv:1804.09081.

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. (2017). Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1487–1495. ACM.

Hazan, E., A, K., and Yuan, Y. (2018). Hyperparameter optimization: A spectral approach. In *International Conference on Learning Representations (ICLR) 2018 Conference Track*.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In Coello, C., editor, *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*, volume 6683 of *Lecture Notes in Computer Science*, pages 507–523. Springer-Verlag.

Hutter, F., Hoos, H., and Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance. In Xing, E. and Jebara, T., editors, *Proceedings of the 31th International Conference on Machine Learning, (ICML'14)*, pages 754–762. Omnipress.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. arXiv:1412.6980.

Klein, A., Falkner, S., Mansur, N., and Hutter, F. (2017). Robo: A flexible and robust bayesian optimization framework in python. In *NIPS 2017 Bayesian Optimization Workshop*.

Krizhevsky, A., Sutskever, I., and Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In Bartlett et al. (2012), pages 1097–1105.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

Lichman, M. (2013). UCI machine learning repository.

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., and Kurakin, A. (2017). Large-scale evolution of image classifiers. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*.

Ruyu, T. and Jiaying, L. (2017). Release year prediction for songs. Technical report.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In Bartlett et al. (2012), pages 2960–2968.

Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. arXiv:1409.3215.

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR'17)*.