Experimental Procedures for Exploiting Structure in AutoML Loss Landscapes

Anonymous[1]

[1]Anonymous Institution

Abstract Recent observations regarding the structural simplicity of algorithm configuration landscapes have spurred the development of new configurators that obtain provably and empirically better performance. Inspired by these observations, we recently performed a similar analysis of AutoML Loss Landscapes – that is, the relationship between hyper-parameter configurations and machine learning model performance. In this study, we propose two new variations of an existing, state-of-the-art hyper-parameter configuration procedure. We designed each method to exploit a specific property that we observed common among most AutoML loss landscapes; however, we demonstrate that neither are competitive with existing baselines. In light of this result, we construct artificial algorithm configuration scenarios that allow us to show when the two new methods can be expected to outperform their baselines and when they cannot, thereby providing additional insights into AutoML loss landscapes.

1 Introduction

In recent years, a large variety of methods for automating machine learning have emerged to automatically select a machine learning training algorithm and model, and then configure its hyper-parameters (Bergstra et al., 2011; Bergstra and Bengio, 2012; Jamieson and Talwalkar, 2016; Klein et al., 2017; Kandasamy et al., 2017; Li et al., 2017, 2020; Fusi et al., 2018; Falkner et al., 2018; Yang et al., 2019), thereby making machine learning faster and easier to use for data scientists and non-experts alike. In parallel to this work, a growing amount of attention has been devoted to studying the landscapes – that is, the relationship between algorithm parameter settings and performance – of a related problem: The configuration of algorithms for solving $\mathcal{NP}$-hard problems (Pedersen, 2010; Yuan et al., 2010, 2012; Pushak and Hoos, 2018; Harrison et al., 2019, 2020, 2021; Asi and Duchi, 2019; Treimun-Costa et al., 2020; Cleghorn and Ochoa, 2021). These insights, for example, the observation that individual algorithm parameters tend to elicit uni-modal responses in performance (Pushak and Hoos, 2018), have already successfully translated into new, state-of-the-art configurators with provably (Hall et al., 2020) and empirically (Pushak and Hoos, 2020) better performance.

Comparatively little is known about AutoML loss landscapes (Garciarena et al., 2018; Pimenta et al., 2020), which relate machine learning training algorithm's hyper-parameters to the resulting model's validation loss. This gap inspired our own recent study of AutoML loss landscapes (Anonymous *et al.*, 2022), in which we observed two common trends in the hyper-parameters of individual machine learning models: First, they appear to induce landscapes that are uni-modal (or very close to it); and second, they appear to have relatively benign hyper-parameter interactions. In particular, we observed that a simplistic configuration procedure that optimizes each hyper-parameter a single time, in a random order, can find hyper-parameter configurations tied with optimal with very high probability. In light of these observations, existing hyper-parameter configuration procedures may be spending larger-than-necessary amounts of their budgets evaluating configurations in regions of the hyper-parameter configuration space that can be reasonably assumed to be of poor quality.

For this reason, using relatively simple surrogate models that incorporate our prior expectation regarding the structure of the landscapes may help to improve the efficiency of configuration procedures by cutting down on unnecessary configuration evaluations. However, both of our proposed surrogate models (see Section 2) proved to be too simple, resulting in similar or worse performance compared to BOHB (Falkner et al., 2018) for most of the real-world AutoML scenarios we studied (see Section 4). To investigate why, we evaluated the methods on four hand-crafted benchmark scenarios (see Section 3), which provides additional insights into a) the structure of AutoML loss landscapes, b) why these particular methods did not perform as well as expected, and c) under which conditions they can be expected to perform well.

2 Experimental AutoML Procedures

We propose two modifications to BOHB (Falkner et al., 2018) (which uses Bayesian optimization), which both involve replacing the tree-structured parzen estimator (TPE) (Bergstra et al., 2011) model with simpler, more constrained surrogate models, which were chosen to reflect one of the structural properties that we observed in AutoML loss landscapes (Anonymous *et al.*, 2022):

- **Convex quadratic approximation (CQA) surrogate models** (Rosen and Marcia, 2004). In fact, the results of our landscape analysis revealed that AutoML loss landscapes are not convex; however, it may still be helpful to use a convex quadratic function as a surrogate model for the landscape, as it is perhaps the simplest n-dimensional model that is constrained to be uni-modal.

- **B-Spline basis surrogate models**. Given that interactions between hyper-parameters in AutoML loss landscapes are relatively benign, fitting a linear model to a B-spline basis function (Eilers and Marx, 1996) is a natural choice. These models are known to provide high-quality and stable approximations, and they force the fitted model to contain no hyper-parameter interactions.

Similar to BOHB's TPE model, each of our surrogate models attempt to approximate the full AutoML loss landscapes. In the case of the convex quadratic surrogate model, this resulted in substantially increased overhead for fitting each of the models. Therefore, in order to avoid slowing down the configuration procedure unnecessarily, we fit the models asynchronously. However, this means that we are required to suggest multiple configurations using the same surrogate model. As a result, we employed a similar, but adapted version of BOHB's method for suggesting configurations, which does not completely optimize the surrogate model. For more details about each of these design decisions as well as how we fit each model, see Appendix A.

3 Experimental Setup

The primary goal in this study is to answer the question: "Can AutoML loss landscape structure be exploited to improve configurator performance?". Therefore, the strategy used to allocate budget between the evaluation of each candidate configuration is effectively an orthogonal design decision that acts as a confounding factor. To isolate the impact of the search method, our preliminary analysis contains only two baselines: BOHB (Falkner et al., 2018) and Hyperband (Li et al., 2017). To further eliminate confounding factors, we used the implementation of Hyperband that is available with the original implementation of BOHB, and used the same values for all of the shared parameters for all four of the configurators.

We evaluated each configurator on five scenarios, spanning a range of different machine learning methods and datasets, as well as on four hand-crafted benchmark scenarios. We present a summary of the scenarios in Table 2 in Appendix B.[1] For each of the real-world scenarios, we

[1] We were unable to reproduce qualitatively similar results for BOHB (Falkner et al., 2018) and Hyperband (Li et al., 2017) on two of the benchmark scenarios we studied (see Appendix B). However, from personal communication with the authors, we confirmed that we correctly set up the experiments and reproduced their method of analysis. To the best of our knowledge, the only differences in our execution environments were the particular machines used and, perhaps, the versions of some of the software packages.

performed 25 independent runs of the configurators, and we report the median test loss with 95% bootstrap percentile confidence intervals. To obtain a stable estimate for the test loss, we performed 5 independent training runs of the machine learning methods using each configuration and recorded their mean test losses. For the hand-crafted benchmark scenarios, which require substantially less running time to use, we performed 101 independent runs of the configurators and we report the exact loss underlying the landscape.

We also evaluated the four configurators on four artificial scenarios that simulate a binary classifier's error rate, p, such that $0 < p < 1$, thereby allowing us to control the precise shape of the AutoML loss landscape optimized by the configuration procedures. This control allowed us to test hypotheses about the relative behaviour of the methods under various circumstances, thereby providing answers to questions that arose from our results on the real-world benchmark scenarios. We motivate and define the precise landscapes that we used for the simulated classifier scenarios in Chapter 4, based on the results of the real-world scenarios.

4 Results

We show the anytime configuration results for three configuration budgets for the five real-world scenarios in Table 3 in Appendix C. Overall, none of the methods consistently perform substantially better than any of the others; however, BOHB found the best or tied for the best on each scenario for all configuration budgets. Unfortunately, while both of the new methods we proposed can find high-quality configurations, neither of them appear to consistently work as well as BOHB.

Why does a CQA surrogate model not work better? Given that we observed that most AutoML loss landscapes are uni-modal (Anonymous *et al.*, 2022), it seems reasonable to assume that a uni-modal surrogate function should be able to improve the performance of BOHB. However, since the CQA surrogate model is a quadratic function, it implicitly assumes that the landscape is symmetric around the globally optimal configuration – which was frequently not the case in our previous study. Therefore, this could be at least partly responsible for the poor performance that we observed with the CQA surrogate model.

To investigate this hypothesis, we compared the performance of the four configuration procedures on two artificial scenarios, where the machine learning algorithm's loss is simulated as described in Appendix B. In both cases, the simulated classifier had a single real-valued hyperparameter, x, that was searched over the range of values $[-1, 1]$. For the first scenario, we set the error rate to be

$$p_{\text{symmetric}}(x) = |x|^3 + 0.01 \tag{1}$$

and for the second scenario we set it to

$$p_{\text{asymmetric}}(x) = \begin{cases} |x|^3 + 0.01 & \text{if } x < 0 \\ \frac{1}{5} \cdot |x|^3 + 0.01 & \text{otherwise.} \end{cases} \tag{2}$$

In the asymmetric scenario, it *should* be easier for the configuration procedures to find better-quality configurations, because 50% of the configurations obtain better error rates. However, we observed that when using a CQA surrogate model, the anytime configuration performance on the asymmetric scenario was actually worse than on the symmetric scenario (see Table 1). Furthermore, we see that on the symmetric scenario, using the CQA surrogate model yields significantly better performance than all of the other methods for most configuration budgets; however, on the asymmetric scenario it instead ties with all of the other methods for all but the 10% configuration budget, at which point it still has a small advantage.

This example clearly illustrates a rather large failure mode for using a CQA model as a surrogate function; even though the asymmetric artificial classifier scenario aligns very closely with the types of landscapes for which we had designed it to perform well, we see that it does not outperform any of the other methods at all.

Table 1: Results from applying the hyper-parameter configurators to the artificial, hand-crafted AutoML scenarios. At each configuration budget we show the median loss (percentage of errors) over the independent configurator runs with 95% bootstrap percentile confidence intervals. The median losses are shown in boldface if they are within 0.01% of the best loss for a given configuration budget. Smaller is better.

	Symmetric	**Asymmetric**	**No Interactions**	**Interactions**
Budget (10%)	13k (Examples)	13k (Examples)	13k (Examples)	13k (Examples)
CQA	**1.01** [1.01, 1.02]	**1.04** [1.03, 1.09]	**3.56** [2.89, 4.88]	**3.08** [2.73, 3.70]
Spline	1.06 [1.03, 1.10]	1.08 [1.04, 1.13]	5.19 [3.93, 6.24]	3.50 [2.99, 4.19]
Random (HB)	1.11 [1.05, 1.20]	1.08 [1.05, 1.12]	5.26 [3.75, 6.39]	4.12 [3.40, 4.43]
TPE (BOHB)	1.12 [1.07, 1.25]	1.08 [1.04, 1.14]	4.32 [3.36, 5.48]	3.68 [3.03, 4.32]
Budget (50%)	67k (Examples)	67k (Examples)	67k (Examples)	67k (Examples)
CQA	**1.01** [1.00, 1.01]	**1.02** [1.01, 1.03]	2.12 [1.86, 2.29]	**1.27** [1.19, 1.37]
Spline	1.02 [1.01, 1.04]	**1.02** [1.01, 1.03]	**1.27** [1.20, 1.44]	1.60 [1.47, 1.74]
Random (HB)	1.04 [1.02, 1.05]	**1.02** [1.01, 1.03]	2.06 [1.84, 2.40]	1.91 [1.77, 2.25]
TPE (BOHB)	1.04 [1.03, 1.10]	**1.02** [1.01, 1.04]	2.40 [1.90, 2.83]	1.64 [1.38, 2.04]
Budget (100%)	135k (Examples)	135k (Examples)	135k (Examples)	135k (Examples)
CQA	**1.00** [1.00, 1.00]	**1.01** [1.00, 1.01]	1.32 [1.21, 1.47]	**1.15** [1.12, 1.18]
Spline	**1.01** [1.00, 1.02]	**1.01** [1.00, 1.02]	**1.11** [1.09, 1.13]	1.26 [1.19, 1.31]
Random (HB)	1.02 [1.01, 1.04]	**1.01** [1.01, 1.02]	1.65 [1.54, 1.85]	1.59 [1.48, 1.73]
TPE (BOHB)	1.03 [1.01, 1.04]	**1.01** [1.01, 1.02]	1.38 [1.23, 1.59]	1.27 [1.16, 1.32]

Why does the spline surrogate model not work better? Similar to the CQA model, we had expected the spline surrogate model to yield better results, given that we observed that most hyper-parameters could be safely configured independently in a random sequence (Anonymous *et al.*, 2022). However, note that this definition of simplicity among hyper-parameter interactions may not actually indicate that a *surrogate model* can assume that the hyper-parameters are independent of each other. Indeed, the function $f(x, y) = |x - y|$ is trivial to minimize when considering either x or y independently – for any fixed value of one, the other can be adjusted to yield an optimal solution. Despite this "simplicity", a the spline surrogate model would be unable to accurately model the function due to the strong interaction between the hyper-parameters x and y.

To investigate, we again compared the performance of the four configuration procedures on two artificial classifier scenarios. Each scenario contained two hyper-parameters, x and y in $[-1, 1]$. In one case there were no interactions and the error rate was defined as

$$p_{\text{no interactions}}(x, y) = \frac{1}{2} \cdot |x| + 0.01 \tag{3}$$

(y was ignored). The second case used the same objective function after applying a 45 degree rotation, that is, the error rate was set to

$$p_{\text{interactions}}(x, y) = \frac{1}{2 \cdot \sqrt{2}} \cdot |x - y| + 0.01. \tag{4}$$

In this experiment, the scenario with hyper-parameter interactions *should* be easier for the configurators, because the basin of optimal solutions lies along the diagonal $x = -y$ (which has length $2 \cdot \sqrt{2}$ for $x, y \in [-1, 1]$) instead of along the line $x = 0$ (which has length 2 over the same domain). However, we see that this simple rotation of the landscape causes the scenario to become substantially more challenging for the spline model. For both of the larger two configuration budgets, the spline model finds configurations with smaller error rates when there are no hyper-parameter interactions (see Table 1). Similarly, for all three configuration budgets, we see that the spline model finds the best solutions when there are no interactions, whereas when there are interactions, the CQA model performs best. (Note that both of the landscapes in these scenarios are perfectly symmetric along two axes around the global optima, hence we would expect CQA to do well in both cases – which it does.)

This example clearly illustrates an important failure mode for the spline model; even though it may often be sufficient to configure each hyper-parameter independently in a random sequence, it is not necessarily safe to build surrogate models that assume that they are independent.

Furthermore, similar to the parameters of the meta-heuristics studied by Yuan et al. (2012), we expect there to be some hyper-parameters with this type of compensatory interaction in AutoML scenarios. For example, consider the learning rate (`l_rate`) and the momentum decay rate (`mdecay`) hyper-parameters for the Bayesian neural network scenarios. The learning rate corresponds to the step size taken in each iteration of the optimization process, and the momentum decay rate corresponds to how quickly old gradient information is lost. Specifically, a large value for momentum corresponds to making heavy use of the previous gradients, whereas a small value corresponds to heavily trusting the most recently observed gradient.

In cases where gradient measurements are noisy, it is helpful to use large values of momentum to take an average over a larger number of previous gradient observations, thereby smoothing the path taken by the optimizer. This should be particularly important for large step sizes, since otherwise the optimizer will be taking large steps in nearly-random directions, effectively placing too much trust in each gradient observation. Indeed, we empirically observed precisely this relationship between these two hyper-parameters. For example, the best 5% of the 1 234 anytime configurations suggested by any of the configuration procedures have a correlation coefficient of 0.83 for the `l_rate` and `mdecay` hyper-parameters on the protein structure dataset. These observations likely explain why the spline model did not produce better results on the real-world scenarios.

5 Conclusions and Future Work

We introduced two experimental modifications to a state-of-the-art AutoML hyper-parameter configuration procedure, BOHB (Falkner et al., 2018). Both of the modifications we made were inspired by the simplicity of the structure we observed in most AutoML loss landscapes (Anonymous *et al.*, 2022). However, in both cases, the models were too simple to be effective on real-world problems, which has lead to a deeper understanding of the structure of AutoML loss landscapes.

In our first attempt, we replaced the TPE model in BOHB with a convex quadratic approximation (CQA) surrogate model. We assumed that because the CQA model is uni-modal (like most AutoML loss landcapes), it should help guide the search process away from regions of the configuration space that we can reasonably assume contain poor configurations. However, the CQA model implicitly assumes that the landscape is symmetric around the global optimum, which proved to be too strong of an assumption.

Our second attempt used a B-spline model, which are known to provide high-quality approximations of functions with no interactions between the variables. This choice was motivated by our observation that configuring each hyper-parameter independently, a single time and in a random order, yields configurations tied with optimal with very high probability (Anonymous *et al.*, 2022). However, this can still occur in the presence of strong *compensatory hyper-parameter interactions* – that is, when the basin of optimal solutions is diagonally oriented within the landscape. In practice, we expect this type of interaction to occur between some hyper-parameters. For example, we showed that the high-quality configurations for the Bayesian neural network scenarios have a correlation coefficient of 0.83 between the values of their momentum decay rate and learning rate hyper-parameters.

Our results suggest a natural next question: What surrogate model can be used to encode our priors, without being too simple, thereby making assumptions that are too strong?

6 Reproducibility Checklist

1. For all authors…

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 4.

   (b) Did you describe the limitations of your work? [Yes] Yes, in fact that was, arguably, the main contribution of our study. See Section 4.

   (c) Did you discuss any potential negative societal impacts of your work? [No] We're not aware of any. Our new methods yielded negative results, and so should have no immediate societal impact. The main impact of our work is a scientific one, which will hopefully contribute to the development of better AutoML methods.

   (d) Have you read the ethics author's and review guidelines and ensured that your paper conforms to them? `https://automl.cc/ethics-accessibility/` [Yes]

2. If you are including theoretical results…

   (a) Did you state the full set of assumptions of all theoretical results? [N/A]

   (b) Did you include complete proofs of all theoretical results? [N/A]

3. If you ran experiments…

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit version), an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [No] But the code will be made available prior to publication of the paper.

   (b) Did you include the raw results of running the given instructions on the given code and data? [No] But they will be made available prior to publication of the paper.

   (c) Did you include scripts and commands that can be used to generate the figures and tables in your paper based on the raw results of the code, data, and instructions given? [No] But they will be made available prior to publication of the paper.

   (d) Did you ensure sufficient code quality such that your code can be safely executed and the code is properly documented? [Yes]

   (e) Did you specify all the training details (e.g., data splits, pre-processing, search spaces, fixed hyperparameter settings, and how they were chosen)? [Yes] Yes, see Appendices A and B.

   (f) Did you ensure that you compared different methods (including your own) exactly on the same benchmarks, including the same datasets, search space, code for training and hyperparameters for that code? [Yes]

   (g) Did you run ablation studies to assess the impact of different components of your approach? [No]

   (h) Did you use the same evaluation protocol for the methods being compared? [Yes] See Section 3.

   (i) Did you compare performance over time? [Yes]

   (j) Did you perform multiple runs of your experiments and report random seeds? [Yes]

(k) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(l) Did you use tabular or surrogate benchmarks for in-depth evaluations? [Yes]

(m) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No]

(n) Did you report how you tuned hyperparameters, and what time and resources this required (if they were not automatically tuned by your AutoML method, e.g. in a NAS approach; and also hyperparameters of your own method)? [Yes] See Appendix A.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets…

(a) If your work uses existing assets, did you cite the creators? [Yes]

(b) Did you mention the license of the assets? [Yes] See Appendix A.

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] All datasets used are open-sourced.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] As far as we are aware, it does not.

5. If you used crowdsourcing or conducted research with human subjects…

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

References

Anonymous *et al.*, A. (to appear 2022). AutoML loss landscapes. *ACM Transactions on Evolutionary Learning and Optimization (TELO)*, page 26 pages.

Asi, H. and Duchi, J. C. (2019). The importance of better models in stochastic optimization. *Proceedings of the National Academy of Sciences*, 116(46):22924–22930.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Proceedings of the Twenty-fifth Conference on Advances in Neural Information Processing Systems (NeurIPS 2011)*, pages 2546–2554.

Bergstra, J. S. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(10):281–305.

Blackard, J. A. and Dean, D. J. (1999). Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and Electronics in Agriculture*, 24(3):131–151.

Chen, T., Fox, E., and Guestrin, C. (2014). Stochastic gradient hamiltonian monte carlo. In *Proceedings of the Thirty-First International Conference on Machine Learning (ICML 2014)*, pages 1683–1691. PMLR.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the Twenty-Second ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD 2016)*, pages 785–794.

Cleghorn, C. W. and Ochoa, G. (2021). Understanding parameter spaces using local optima networks: A case study on particle swarm optimization. In *Proceedings of the Twenty-Third International Genetic and Evolutionary Computation Conference Companion (GECCO 2021 Companion)*, pages 1657–1664.

Dua, D. and Graff, C. (2017). UCI machine learning repository.

Eilers, P. H. and Marx, B. D. (1996). Flexible smoothing with B-splines and penalties. *Statistical Science*, 11(2):89–121.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the Thirty-Fifth International Conference on Machine Learning (ICML 2018)*, pages 1437–1446.

Falkner, S., Lindauer, M., and Hutter, F. (2015). SpySMAC: Automated configuration and performance analysis of SAT solvers. In *Proceedings of the Eighteenth International Conference on Theory and Applications of Satisfiability Testing (SAT 2015)*, pages 215–222.

Fusi, N., Sheth, R., and Elibol, M. (2018). Probabilistic matrix factorization for automated machine learning. *Proceedings of the Thirty-Second Conference on Neural Information Processing Systems (NeurIPS 2010)*, 31.

Garciarena, U., Santana, R., and Mendiburu, A. (2018). Analysis of the complexity of the automatic pipeline generation problem. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2018)*, pages 1–8. IEEE.

Hall, G. T., Oliveto, P. S., and Sudholt, D. (2020). Fast perturbative algorithm configurators. In *Proceedings of the Sixteenth International Conference on Parallel Problem Solving from Nature (PPSN 2020)*, pages 19–32. Springer International Publishing.

Harrison, K. R., Ombuki-Berman, B. M., and Engelbrecht, A. P. (2019). The parameter configuration landscape: A case study on particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2019)*, pages 808–814.

Harrison, K. R., Ombuki-Berman, B. M., and Engelbrecht, A. P. (2020). Visualizing and characterizing the parameter configuration landscape of differential evolution using physical landform classification. In *Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI 2020)*, pages 2437–2444.

Harrison, K. R., Ombuki-Berman, B. M., and Engelbrecht, A. P. (2021). Visualizing and characterizing the parameter configuration landscape of particle swarm optimization using physical landform classification. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2021)*, pages 2299–2306. IEEE.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2014). An efficient approach for assessing hyperparameter importance. In *Proceedings of the Thirty-First International Conference on Machine Learning (ICML 2014)*, pages 754–762.

Jamieson, K. and Talwalkar, A. (2016). Non-stochastic best arm identification and hyperparameter optimization. In *Artificial Intelligence and Statistics*, pages 240–248. PMLR.

Kandasamy, K., Dasarathy, G., Schneider, J., and Póczos, B. (2017). Multi-fidelity bayesian optimisation with continuous approximations. In *Proceedings of the Thirty-Fourth International Conference on Machine Learning (ICML 2017)*, pages 1799–1808.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Proceedings of the Thirty-First Conference on Advances in Neural Information Processing Systems (NeurIPS 2017)*, 30:3146–3154.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017). Fast Bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial Intelligence and Statistics*, pages 528–536. PMLR.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, E., Ben-tzur, J., Hardt, M., Recht, B., and Talwalkar, A. (2020). A system for massively parallel hyperparameter tuning. In *Proceedings of Machine Learning and Systems (MLSys 2020)*, volume 2, pages 230–246.

Pedersen, M. E. H. (2010). *Tuning & simplifying heuristical optimization*. PhD thesis, University of Southampton.

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

Pimenta, C. G., de Sá, A. G., Ochoa, G., and Pappa, G. L. (2020). Fitness landscape analysis of automated machine learning search spaces. In *Proceedings of the Twentieth European Conference on Evolutionary Computation in Combinatorial Optimization (EVOCOP 2020 – Part of EvoStar)*, pages 114–130. Springer International Publishing.

Pushak, Y. and Hoos, H. H. (2018). Algorithm configuration landscapes: More benign than expected? In *Proceedings of the Fifteenth International Conference on Parallel Problem Solving from Nature (PPSN 2018)*, pages 271–283.

Pushak, Y. and Hoos, H. H. (2020). Golden parameter search: Exploiting structure to quickly configure parameters in parallel. In *Proceedings of the Twenty-Second International Genetic and Evolutionary Computation Conference (GECCO 2020)*, pages 245–253.

Rosen, J. B. and Marcia, R. F. (2004). Convex quadratic approximation. *Computational Optimization and Applications*, 28(2):173–184.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shapiro, A. D. (1987). *Structured induction in expert systems*. Addison-Wesley Longman Publishing Co., Inc.

Springenberg, J. T., Klein, A., Falkner, S., and Hutter, F. (2016). Bayesian optimization with robust Bayesian neural networks. *The Thirtieth Conference on Advances in Neural Information Processing Systems (NeurIPS 2016)*, 29:4134–4142.

Treimun-Costa, G., Montero, E., Ochoa, G., and Rojas-Morales, N. (2020). Modelling parameter configuration spaces with local optima networks. In *Proceedings of the Twenty-Second International Genetic and Evolutionary Computation Conference (GECCO 2020)*, pages 751–759.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Yang, C., Akimoto, Y., Kim, D. W., and Udell, M. (2019). OBOE: Collaborative filtering for automl model selection. In *Proceedings of the Twenty-Fifth ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (SIGKDD 2019)*, pages 1173–1183.

Yuan, Z., De Oca, M. A. M., Birattari, M., and Stützle, T. (2012). Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms. *Swarm Intelligence*, 6(1):49–75.

Yuan, Z., Stützle, T., and Birattari, M. (2010). MADS/F-Race: Mesh adaptive direct search meets F-race. In *Proceedings of the Twenty-Third International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE 2010)*, pages 41–50.

A Extended Methods

In this section, we include additional details about our two experimental hyper-parameter configuration methods. Each method was implemented as a modification to the open-source version of BOHB (Falkner et al., 2018), which was released with a BSD 3-clause license at `https://github.com/automl/HpBandSter`.

A.1 Convex Quadratic Surrogate Models

To fit convex quadratic models, we use the methods proposed by Rosen and Marcia (2004). One advantage of their methods compared to others is that the models are fitted using the L_1 norm, which is more robust than the L_2 norm in the presence of noise and outliers. They further impose the constraint that the fitted models must under-estimate all of the available training data. While it is unclear whether or not this will be beneficial for our application, we do not expect it to harm the performance of the method. Adding such a constraint will increase the residuals of an optimally-fitted model and thus will decrease the accuracy of the model's predictions. However, a model need not have high predictive accuracy in order to be an effective surrogate model; indeed, an ideal surrogate model only needs two properties: it should be easy to optimize and its global optimum should be in a similar location to the global optimum of the original landscape.

To fit the convex quadratic under-estimator models, Rosen and Marcia (2004) proposed two methods, both of which formulate the problem as a two-step procedure. In the first step of each, the problem is simplified by imposing additional constraints on the quadratic model, thereby allowing the problem to be efficiently solved using linear programming. They then proposed to use the incumbent solution from the linear program to initialize the parameters of the non-linear program that is used to fit the general convex quadratic approximation model. This initial guess for the solution to the non-linear program can then be iteratively refined using an interior-point method.

In the first method, they force the Hessian to be a diagonal matrix, and in the second, they force it to be a diagonally-dominant matrix. Rosen and Marcia (2004) found that the second method yielded higher-quality final solutions, therefore, we chose to use it when possible. We fit a general convex quadratic surrogate model using their second method whenever there was enough data to do so (when the number of points evaluated m, was at least $(n+1)\cdot(n+2)/2$, where n is the number of parameters) and we fit a separable convex quadratic surrogate model otherwise (when $2\cdot n+1 < m < (n+1)\cdot(n+2)/2$).

The methods proposed by Rosen and Marcia (2004) also require a parameter ϵ, which is used to constrain the minimum eigenvalue of the Hessian, thereby improving the conditioning of the surrogate-fitting optimization problems and ensuring that the optimum of the surrogate is unique. In our experiments, we set $\epsilon = 10^{-3}$, since it is well-known that some hyper-parameters may have relatively little impact on the solution quality (see for example, Hutter et al. (2014) or Chapter **??**). Furthermore, we optimized the linear programs using the default linear-programming solver available in python's `SciPy` package (Virtanen et al., 2020). To optimize the non-linear programs, we used `SciPy`'s sequential least squares quadratic programming solver (SLSQP).

A.2 B-Spline Basis Surrogate Models

Gaussian processes are a more commonly used model in AutoML methods that would likely provide similar-quality results to spline models. Guassian processes are typically chosen, because they can provide a confidence interval around their predictions, which is necessary for most Bayesian optimization methods. However, in our application, where we wish to focus on exploitation rather than trading off between exploration and exploitation, we do not require this confidence interval. Instead, we prefer splines for their lower computational complexity.

Another advantage of using splines is that they support various methods for extrapolation. For our application, we chose to extrapolate using a constant function, as this will ensure that unpromising directions for exploration remain unexplored, whereas promising directions for exploration will continue to be explored.

To compute the spline basis function, we used the implementation available in `scikit-learn`'s latest development branch,[2] with the default settings of its parameters: five knots per feature with a third-degree basis. For hyper-parameters that are searched on a linear scale, we used a uniform spacing for the knots, whereas we spaced the knots geometrically for hyper-parameters that are searched on a logarithmic scale. Because we anticipated that the models would frequently be under-constrained, particularly early in the configuration process when relatively little configuration performance data is available, we used LASSO regression with five-fold cross-validation to choose the value of the regularization parameter.

A.3 Asynchronous Model Fitting and Selection

Since fitting these surrogate models typically requires substantially more time than fitting a TPE model, we fit the models asynchronously. In BOHB, new models are fitted when worker processes report new configuration results to the main process. In our methods, we initiate and check for completed model-fitting processes in this function as well (see Algorithm 1). We limit the number of models being fitted at any given time to one, to ensure that it does not overburden the machine and slow down the evaluation of the configurations.

Similarly, BOHB also evaluates each candidate configuration asynchronously. Each time a worker process for BOHB is done evlauating a configuration, the worker process reports its results to the main process. When the main process receives this data, it checks to see if there is sufficient data available to fit a model and if there are no models that are currently being fitted. If both of these conditions are met, the main process will initiate an asynchronous model-fitting process. Once the model-fitting process is complete, it reports its results back to the main process.

Similar to BOHB, we fit separate CQA or spline models for each level of fidelity of the performance estimates. In some early prototypes, we experimented with using a weighted combination

[2] As of 2021-06-24.

Algorithm 1 The report function for our experimental AutoML configurators.

```
input
    $m_I(c)$, the performance of a newly evaluated configuration
    $f$, the level of fidelity at which the configuration was evaluated
    $R$, a dictionary mapping fidelities to configuration evaluation results
    $model$, a dictionary mapping fidelities to previously-fit surrogate models (or $None$)
output
    None
procedure report
    # Save the configuration results and check for model-fitting results
    $R[f][c] := m_I(c)$
    Check if a model-fitting process has completed
    # Save the new model, if applicable
    if a model-fitting process reported $model_{new}$ at fidelity level $f'$, then
        if $model[f']$ is $None$ or $model_{new}$ better approximates $R[f']$ than $model[f']$, then
            $model[f'] := model_{new}$
    # Start a new model-fitting process, if applicable
    if no model-fitting processes are running, then
        for each fidelity $f \in R$ in descending order, do
            if $length(R[f]) \geq 2 \cdot |P| + 1$, then
                Initiate a model-fitting process with $R[f]$
                break
    return
```

of some or all of the available performance estimates, regardless of their level of fidelity. However, we observed that because some measures of fidelity not only decrease the variability in the performance estimates, but also substantially change the performance values for a given configuration, doing so typically caused the model-fitting procedure to diverge, thereby producing unreliable surrogate models. Therefore, as soon as sufficient data to fit a model is available for a higher-fidelity performance estimate, the methods switch to fitting models using the higher level of fidelity instead.

In some rare cases the addition of new configuration performance data, or the switch from a large training set with low-fidelity performance estimates to a small training set with higher-fidelity performance estimates, can de-stabilize the model-fitting procedure, thereby causing it to diverge. To protect against this, we only accept a new surrogate model if the mean absolute error of a newly-fitted model is not worse than the current accepted model, if one exists.

A.4 Suggesting a Configuration

The simplest way to use a surrogate model to guide an optimization process is to find the globally optimal solution to the surrogate model and then to evaluate that solution using the original objective function. When the surrogate model is a convex quadratic model, this would be both easy and inexpensive to do. However, surrogate models that are fitted to previous performance data are typically only good approximations of the original landscape near to the previously-evaluated solutions. Therefore, it is common practice to employ, for example, a trust region, which limits how far away from the current incumbent the surrogate model should be trusted. The next configuration to evaluate is then the configuration predicted to have the optimal performance by the surrogate model that is contained within the trust region.

However, in our application, the surrogate models are fitted asynchronously, so it is also sometimes necessary for multiple new configurations to be suggested using the same surrogate model. Therefore, rather than using a trust region, we employ a somewhat similar trick to BOHB, whereby we randomly sample new configurations from a mixture of three Gaussian models, where each of the three Gaussian models are parameterized to be centered around the three best known configurations. To determine which configurations are best, we simply used the largest available fidelity budget evaluation for each of the configurations (see Algorithm 2).

Algorithm 2 The suggest function for our experimental AutoML configurators.

```
input
    $R$, A dictionary mapping fidelities to configuration evaluation results
    $model$, A dictionary mapping fidelities to previously-fit surrogate models (or $None$)
    $C$, the parameter configuration space
output
    $c_{next}$, a configuration to evaluate
procedure suggest
    # Find the largest-fidelity model, if any
    Initialize $model_{current} := None$
    for each fidelity $f \in R$ in descending order, do
        if $model[f]$ is not $None$, then
            $model_{current} := model[f]$
            break
    # Sample a new configuration using the model, if possible
    if $model_{current}$ is $None$, then
        Pick $c_{next}$ by sampling a random configuration from $C$
    else
        # Find the best configuration at any fidelity level
        Initialize an empty dictionary, $performance := \{\}$
        for each fidelity $f \in R$, in ascending order, do
            for each configuration $c \in R[f]$, do
                $performance[c] := R[f][c]$
        # Sample from a mixture of 3 Gaussian models (see text for details)
        Sample 9 configurations from around the best 3 configurations in $performance$
        Pick $c_{next}$ as the best of the 9 configurations according to $model_{current}$
    return $c_{next}$
```

We chose to use a mixture of three Gaussian models instead of a single Gaussian model, in case any of the low-fidelity budgets yield high-variance performance estimates that could cause the procedure to select the wrong configuration as the incumbent. We heuristically set the covariance of each of the Gaussian models to be a diagonal matrix parameterized such that the standard deviation of each hyper-parameter is equal to 5% of the total range of values for the hyper-parameter. Finally, each time a configuration needs to be suggested we sample a total of nine candidate configurations from the mixture of Gaussian models and suggest the one predicted to have the best performance by the surrogate model. The number of Gaussians, their standard deviations, and the number of samples drawn were each chosen based on some preliminary experiments on hand-crafted benchmarks.

For spline surrogate models, this procedure has the added benefit that we do not need to be able to locate the optimal configuration according the spline model, which may be non-trivial to do, as the model is not guaranteed to be uni-modal.

B Benchmark Scenarios

The first three scenarios we studied were also used in the original evaluation of BOHB (Falkner et al., 2018). The first two are for a Bayesian neural network trained using stochastic gradient Hamiltonian Monte-Carlo sampling (Chen et al., 2014) with scale adaption (Springenberg et al., 2016) on two UCI datsets (Dua and Graff, 2017), Boston housing and protein structure. We were unable to reproduce qualitatively similar results compared to those reported in Falkner et al. (2018) for either of these two scenarios. In each case, we observed that the gap between the performance for BOHB and Hyperband was much smaller than reported in their original study. From personal

Table 2: The five real-world and four hand-crafted scenarios used to compare the four different hyper-parameter configurators.

Model/Algorithm	**Dataset/Task**	**Fidelity Type**	**Fidelity Range**	**# HP**
Bayesian Neural Network	Boston Housing	MCMC Steps	[500, 10 000]	7
	Protein Structure	MCMC Steps	[500, 10 000]	7
Proximal Policy Optimization	Cart Pole Swing-Up	# Training Runs	[1, 9]	5
Xgboost	King-Rook *vs* King-Pawn	# Estimators	[2, 128]	6
Histogram Gradient Boosting	Covertype	Dataset Fraction	[1%, 100%]	7
Simulated Binary Classifier	Symmetric	Dataset Size	[500, 5 000]	1
	Asymmetric	Dataset Size	[500, 5 000]	1
	No Interactions	Dataset Size	[500, 5 000]	2
	Interactions	Dataset Size	[500, 5 000]	2

communication with the authors, we confirmed that we correctly set up the experiments and reproduced their method of analysis. To the best of our knowledge, the only differences in our execution environments were the particular machines used and, perhaps, the versions of some of the software packages.

The third scenario we used from their study configures the hyper-parameters of proximal policy optimization (Schulman et al., 2017) on the cart pole swing-up reinforcement learning task. In this case, even though the final performance gap between BOHB and Hyperband was also slightly smaller than originally reported, the results were still qualitatively similar. The results that we present for these scenarios in Figure **??** further differ from the original results on the scenarios because we present the median test losses (obtained via a held-out test set) as opposed to the mean validation losses (estimated by the configuration procedures themselves) as was done by Falkner et al. (2015).

For the next two scenarios, we configured the hyper-parameters of Xgboost (Chen and Guestrin, 2016) on the UCI (Dua and Graff, 2017) king-rook *vs* king-pawn (Shapiro, 1987) dataset and histogram gradient boosting (`scikit-learn`'s (Pedregosa et al., 2011) implementation of a machine learning method inspired by LightGBM (Ke et al., 2017)) on the covertype dataset (Blackard and Dean, 1999).

For the simulated classifier scenarios, the classifier was constructed such that the variability in the measurements of the landscape mimic the behaviour of a binary classifier with a given error rate on a validation set of a given size. In particular, for each scenario, we defined functions that describe the landscape by mapping configurations to error rates, p, such that $0 < p < 1$. Then, given the error rate, p, of a particular configuration and a validation dataset size of n (the level of fidelity), we simulated an evaluation of a configuration by randomly drawing n trials from a binomial distribution with a success rate of p, and we recorded the loss as the number of successful trials divided by the total number of trials, n. We simulated short training times for the procedure by sleeping for one second per 1 000 instances in the dataset. This simulation ignores the variability in performance due to the random seed used when training a machine learning model; however, it still provides a cheap-to-evaluate and approximately realistic method for generating landscapes with exactly known structure that can be used to evaluate the configurators.

C Extended Results

In Table 3, we show the results from applying the four hyper-parameter configuration procedures to the real-world AutoML scenarios. As can be seen, neither of the two new experimental methods provide results competitive with the two baselines: BOHB (Falkner et al., 2018) and Hyperband (Li et al., 2017). A similar analysis of other configuration budgets (not shown) yields qualitatively similar results.

Table 3: Results from applying the hyper-parameter configurators to the real-world AutoML scenarios. At each configuration budget we show the median test loss over the independent configurator runs with 95% bootstrap percentile confidence intervals. The median losses are shown in boldface if they are not worse than the best loss for a given configuration budget, according to a one-sided Welch t-test with a 5% significance level. All losses are scaled by dividing by the loss of the single best known configuration for each scenario. Smaller is better.

	BNN-BH	**BNN-PS**	**PPO-CP**	**XGB-KR**	**HGB-CT**
Budget (10%)	1 580 (Seconds)	2 420 (Seconds)	4 667 (Seconds)	148 (Seconds)	888 (Seconds)
CQA	**1.91** [1.46, 2.42]	**2.14** [1.63, 4.30]	**10.49** [7.67, 13.74]	**1.72** [1.44, 2.28]	1.28 [1.18, 1.35]
Spline	**1.52** [1.40, 1.79]	**1.77** [1.63, 1.87]	13.74 [9.42, 13.74]	**1.72** [1.41, 2.41]	1.30 [1.21, 1.42]
Random (HB)	3.10 [1.60, 4.00]	**2.13** [1.77, 4.74]	**7.73** [6.21, 10.13]	**1.91** [1.50, 2.00]	**1.22** [1.13, 1.42]
TPE (BOHB)	**1.76** [1.45, 2.05]	**1.87** [1.56, 3.09]	**7.70** [4.57, 9.84]	**1.69** [1.47, 2.00]	**1.23** [1.14, 1.48]
Budget (50%)	7 899 (Seconds)	12 102 (Seconds)	23 337 (Seconds)	742 (Seconds)	4 442 (Seconds)
CQA	**1.44** [1.31, 1.67]	**1.97** [1.72, 2.60]	5.73 [3.06, 6.47]	1.91 [1.44, 2.12]	1.17 [1.12, 1.24]
Spline	**1.50** [1.42, 1.70]	1.85 [1.55, 2.24]	**5.31** [4.10, 8.43]	**1.56** [1.41, 1.94]	1.11 [1.10, 1.14]
Random (HB)	**1.39** [1.36, 1.57]	**1.50** [1.31, 1.83]	**4.97** [3.25, 6.86]	**1.78** [1.53, 1.97]	**1.08** [1.07, 1.10]
TPE (BOHB)	**1.38** [1.27, 1.68]	**1.24** [1.10, 1.40]	**3.59** [2.31, 5.37]	**1.37** [1.28, 1.53]	**1.09** [1.07, 1.14]
Budget (100%)	15 798 (Seconds)	24 204 (Seconds)	46 674 (Seconds)	1 484 (Seconds)	8 884 (Seconds)
CQA	**1.44** [1.31, 1.67]	**1.97** [1.62, 2.32]	**4.45** [2.92, 6.11]	1.72 [1.41, 2.00]	1.12 [1.12, 1.17]
Spline	**1.50** [1.43, 1.77]	1.85 [1.35, 2.24]	4.44 [3.53, 6.51]	**1.56** [1.41, 1.94]	1.09 [1.06, 1.13]
Random (HB)	**1.39** [1.36, 1.57]	**1.50** [1.31, 1.83]	**4.67** [2.69, 6.66]	1.72 [1.53, 1.97]	1.08 [1.06, 1.09]
TPE (BOHB)	**1.38** [1.27, 1.68]	**1.25** [1.10, 1.43]	**2.22** [1.76, 3.21]	**1.38** [1.31, 1.59]	**1.05** [1.03, 1.08]