Hyperparameter Transfer Across Developer Adjustments

Anonymous Author(s)
Affiliation
Address
email

Abstract

After developer adjustments to a machine learning (ML) system, how can the results of an old hyperparameter optimization automatically be used to speed up a new hyperparameter optimization? This question poses a challenging problem, as developer adjustments can change which hyperparameter settings perform well, or even the hyperparameter space itself. While many approaches exist that leverage knowledge obtained on previous *tasks*, so far, knowledge from previous *development steps* remains entirely untapped. In this work, we remedy this situation and propose a new research framework: hyperparameter transfer across adjustments (HT-AA). To lay a solid foundation for this research framework, we provide four HT-AA baseline algorithms and eight benchmarks. The best baseline, on average , reaches a given performance 2x faster than a prominent HPO algorithm without transfer. As hyperparameter optimization is a crucial step in ML development but requires extensive computational resources, this speed up would lead to faster development cycles, lower costs, and reduced environmental impacts. To make these benefits available to ML developers off-the-shelf, we provide a python package that implements the proposed transfer algorithm.

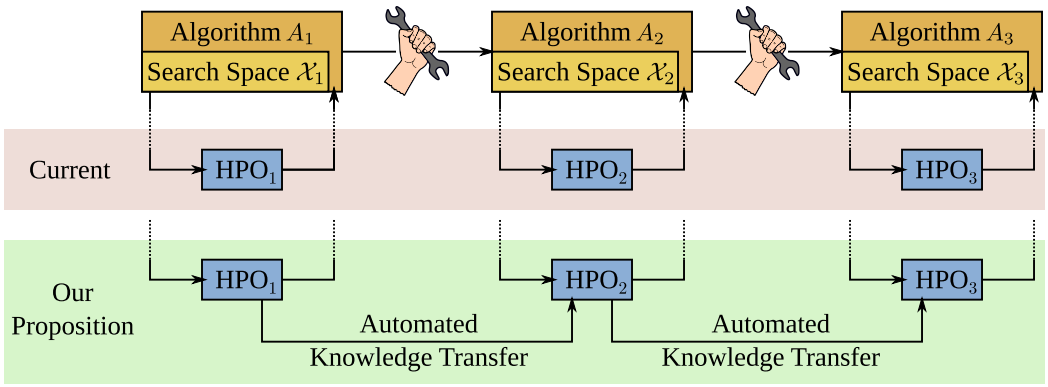

Figure 1: Hyperparameter Optimization (HPO) across adjustments to the algorithm or hyperparameter search space. Common practice is to perform HPO from scratch after each adjustment, or to somehow manually transfer knowledge. In contrast, we propose a new research framework about automatic hyperparameter knowledge transfers across adjustments.

Submitted to 34th Conference on Neural Information Processing Systems (NeurIPS 2020). Do not distribute.

1 Introduction: A New Hyperparameter Transfer Framework

The machine learning (ML) community arrived at the current generation of ML algorithms by performing many iterative adjustments. Likely, the way to artificial general intelligence requires many more adjustments. Each adjustment could change which hyperparameter settings perform well, or even the hyperparameter space itself (Chen et al., 2018; Li et al., 2020). For example, when deep learning developers change the optimizer, the learning rate's optimal value likely changes, and the new optimizer may also introduce new hyperparameters. Since machine learning systems are known to be very sensitive to their hyperparameters (Chen et al., 2018; Feurer and Hutter, 2018), developers are faced with the question of how to adjust their hyperparameters after changing their code. Assuming that the developers have results of one or several hyperparameter optimizations (HPOs) that were performed before the adjustments, they have two options:

1. Somehow manually transfer knowledge from old HPOs.

This is the option chosen by many researchers and developers, explicitly disclosed, e.g., in the seminal work on AlphaGo (Chen et al., 2018). However, this is not a satisfying option since manual decision making is time-consuming, often individually designed, and has already lead to reproducibility problems (Musgrave et al., 2020).

2. Start the new HPO from scratch.

Leaving previous knowledge unutilized can lead to higher computational demands and worse performance (Section 5). This is especially bad as the energy consumption of machine learning systems is already recognized as an environmental problem. Deep learning pipelines, for example, can have CO_2 emissions in the order of magnitude of multiple car lifetime's worth of emissions (Strubell et al., 2019) and their energy demands are growing furiously: Schwartz et al. (2019) cite a "300,000x increase from 2012 to 2018". Therefore, reducing the number of evaluated hyperparameter settings should be a general goal of the community.

The **main contribution** of this work is the introduction of a new research framework: *Hyperparameter transfer across adjustments (HT-AA)*, which empowers developers with a third option:

3. Automatically transfer knowledge from previous HPOs.

This option leads to advantages in two aspects: the automation of decision making and the utilization of previous knowledge. On the one hand, the automation allows to benchmark strategies, replaces expensive manual decision making, and enables reproducible and comparable experiments; on the other hand, the utilization of previous knowledge leads to faster development cycles, lower costs, and reduced environmental impacts.

To lay a solid foundation for the new transfer framework, our **individual contributions** are as follows: **-I-** A detailed introduction of the hyperparameter transfer across adjustments framework (Section 2). **-II-** A placement of our framework in existing research efforts and a discussion of the research opportunities that our framework opens up. **-III-** We provide four simple baseline algorithms for this problem (Section 4), and perform an empirical study across eight benchmarks, where one of the algorithms was twice as fast to reach a given performance than HPO from scratch (Section 5) **-IV-** Further, we provide open source code for our experiments, and benchmarks, and provide a python package with an out-of-the-box usable implementation of our HT-AA algorithms.

2 Hyperparameter Transfer Across Adjustments Framework

After having provided a broad introduction to the topic, we now provide a detailed description of hyperparameter transfer across developer adjustments (HT-AA). We first introduce hyperparameter optimization, then discuss the types of developer adjustments, and finally describe the transfer across these adjustments.

Hyperparameter optimization In this work, we focus on a basic version of hyperparameter optimization (HPO) and refer to Section 3 for a discussion on potential extensions of our framework to more advanced HPO scenarios. Specifically, the HPO formulation we utilize in this work is the following problem:

$$\underset{\boldsymbol{x}\in\mathcal{X}}{\text{minimize}}\, f_S(\boldsymbol{x}) \quad \text{with } b \text{ trials} \qquad , \tag{1}$$

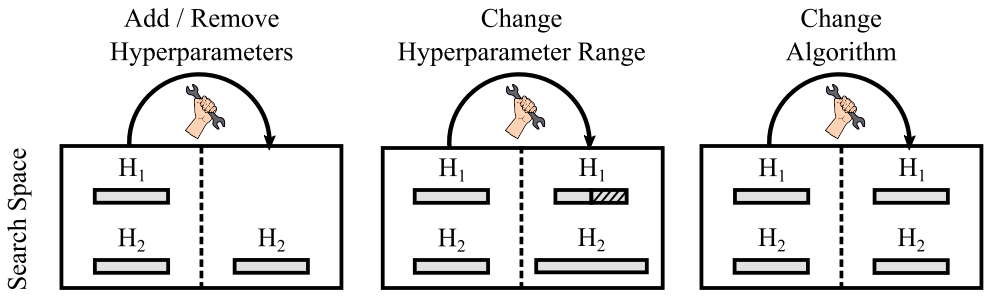

Figure 2: Developer adjustments from the perspective of hyperparameter optimization.

where S is a machine learning system, f_S is the objective function, b is the number of available evaluations, and $\mathcal{X}$ is the hyperparameter space. We refer to a specific HPO problem with the three tuple $(\mathcal{X},\ f_S,\ b)$.

Developer adjustments We now put developer adjustments on concrete terms. From the perspective of HPO, we consider two different categories of adjustments: ones that do not change the hyperparameter space $\mathcal{X}$ (homogeneous adjustments), and ones that do (heterogenous adjustments). For heterogeneous adjustments, we further differentiate between adjustments that add or remove a hyperparameter (hyperparameter adjustments), and adjustments that change the search space for a specific hyperparameter (range adjustments). Regarding homogeneous adjustments, these could either change the algorithm implementation or the hardware that the algorithm is run on. Figure 2 shows an illustration of the adjustment types.

Knowledge transfer across adjustments In general, a continuous stream of developer adjustments could be accompanied by many HPOs. However, in this work, we only consider the transfer between two HPO problems and refer to a discussion about a potential extension in Section 3. The two HPO problems arise from adjustments Ψ to a machine learning system S_{old} and its hyperparameter space $\mathcal{X}_{\text{old}}$, which lead to $S_{\text{new}}, \mathcal{X}_{\text{new}} := \Psi(S_{\text{old}},\ \mathcal{X}_{\text{old}})$. Specifically, the hyperparameter transfer across adjustments problem is to solve the HPO problem $(\mathcal{X}_{\text{new}},\ f_{S_{\text{new}}},\ b_{\text{new}})$, given the results for $(\mathcal{X}_{\text{old}},\ f_{S_{\text{old}}},\ b_{\text{old}})$. Compared to current HPO practices, developers can choose a lower budget b_{new}, given evidence for a transfer algorithm achieving the same performance faster.

3 Related Work and Research Opportunities

In this section, we discuss work related to hyperparameter transfer across adjustments (HT-AA) and present several research opportunities in combining existing ideas with HT-AA.

Transfer learning Transfer learning studies how to use observations from one or multiple source tasks to improve learning on one or multiple target tasks (Zhuang et al., 2019). If we view the HPO problems before and after specific developer adjustments as tasks, we can consider HT-AA as a specific transfer learning problem. As developer adjustments may change the hyperparameter space, HT-AA would then be categorized as a heterogeneous transfer learning problem (Day and Khoshgoftaar, 2017).

Transfer learning across adjustments Recently, Berner et al. (2019) transferred knowledge between deep reinforcement learning agents across developer adjustments. For each type of adjustment they encountered, they crafted techniques to preserve, or approximately preserve, the neural network policy. Their transfer strategies are inspired by Net2Net knowledge transfer (Chen et al., 2015), and they use the term surgery to refer to this practice. This work indicates that transfer learning across adjustments is not limited to knowledge about hyperparameters, but extends to a more general setting, leaving room for many research opportunities.

Continuous Knowledge Transfer In this paper, we focus on transferring knowledge from the last hyperparameter optimization (HPO) performed, but future work could investigate a continuous transfer of knowledge across many cycles of adjustments and HPOs. Transferring knowledge from HPO runs on multiple previous versions could lead to further performance gains, as information from

each version could be useful for the current HPO. Such continuous HT-AA would then be related to the field of continual learning. (Thrun and Mitchell, 1995; Lange et al., 2020).

Hyperparameter transfer across tasks (HT-AT) There exists an extensive research field that studies the transfer across *tasks* for HPOs (Vanschoren, 2018). The main difference to hyperparameter transfer across *adjustments* is that the former assumes an unchanging hyperparameter space, where as dealing with such changes is one of the main challenges in the later. In HT-AT, the hyperparameter space and the machine learning system remain unchanged, but the task that the system is applied to changes. Hyperparameter transfer across adjustments (HT-AA) problems, where all adjustments are homogeneous (do not change the hyperparameter space), are syntactically equivalent to an HT-AT problem. In such cases, approaches for HT-AT could be applied without modification, provided they work with one task. Further, adaptations of across task strategies to the across adjustments setting could lead to more powerful HT-AA approaches in the future, and the combination of across task and across adjustments hyperparameter transfer is an exciting research opportunity that could provide even larger speedups.

Advanced hyperparameter optimization HT-AA can be applied to one of the many extensions to the basic hyperparameter optimization (HPO) formulation. One such extension is multi-fidelity HPO, which allows the use of cheap-to-evaluate approximations to the actual objective (Li et al., 2017; Falkner et al., 2018). Similarly, cost-aware HPO adds a cost to each hyperparameter setting, so a cost model can prioritize the evaluation of cheap hyperparameter settings over expensive ones (Snoek et al., 2012). Yet another extension is to take different kinds of evaluation noise into account (Kersting et al., 2007) or to consider not one, but multiple objectives to optimize for (Khan et al., 2002). All these HPO formulations can be studied in conjunction with HT-AA, to either provide further speedups, or to deal with more general optimization problems.

4 Baseline Algorithms for HT-AA

In this section, we present several baselines for the specific instantiation of the hyperparameter transfer across adjustments (HT-AA) framework which considers basic hyperparameter optimization (HPO), a one-step transfer, and no user annotations or code analysis. We resist the temptation to introduce complex approaches alongside a new research framework and instead focus on a solid foundation. Specifically, we focus on approaches that do not consider any knowledge from the new HPO run for the transfer. As some of these strategies already lead to strong performance (Section 5), the design of more complex approaches is an exciting future direction. We first introduce the basic HPO algorithm that the transfer approaches build upon, and then the four approaches themselves.

Background For basic hyperparameter optimization and parts of the transfer algorithm, we employ the Tree-Structured Parzen Estimator (TPE) algorithm (Bergstra et al., 2011), which is the default algorithm in the popular HyperOpt package (Bergstra et al., 2013). TPE uses kernel density estimators to model the densities $l(\boldsymbol{x})$ and $g(\boldsymbol{x})$, for the probability of a given hyperparameter configuration $\boldsymbol{x}$ being worse (l), or better (g), than the best already evaluated configuration $\boldsymbol{x}^*$. To decide which configuration to evaluate, TPE then solves $\boldsymbol{x}^* \in \arg\max_{\boldsymbol{x}\in\mathcal{X}} {}^{g(\boldsymbol{x})}/_{b(\boldsymbol{x})}$ approximately. In our experiments, we use the TPE implementation and hyperparameter settings from Falkner et al. (2018).

Only Optimize New Hyperparameters A natural strategy for HT-AA is to set already optimized hyperparameters to their previous best setting and only tune new hyperparameters (Agostinelli et al., 2014; Huang et al., 2017; Wu and He, 2018). If the previous best setting is not a valid configuration anymore due to range adjustments, this strategy uses the best setting that still is a valid configuration. In the following, we refer to this strategy as *only-optimize-new*.

Drop Unimportant Hyperparameters Another strategy inspired by manual HT-AA efforts is to only optimize important hyperparameters. The utilization of importance measurements was, for example, explicitly disclosed in the seminal work on AlphaGo (Chen et al., 2018). Here, we determine the importance of each individual hyperparameter with functional analysis of variance (fANOVA) (Hutter et al., 2014) and do not tune hyperparameters with below mean importance. Therefore, this strategy only optimizes new hyperparameters and hyperparameters with above mean importance. In the following, we refer to this strategy as *drop-unimportant*.

First Evaluate Best The *best-first* strategy uses only-optimize-new for the first evaluation, and for the remaining evaluations uses standard TPE. This strategy has a large potential speed up and low risk as it falls back to standard TPE.

Transfer TPE (T2PE) We introduce T2PE in two parts: first, the strategy to deal with homogeneous adjustments or hyperparameter adjustments, and second, the strategy to deal with range adjustments. Please find the pseudocode for T2PE in Appendix B and an illustration in Appendix A.

For homogeneous adjustments and hyperparameter adjustments, the new hyperparameter space $\mathcal{X}_{\text{new}}$ and the old hyperparameter space $\mathcal{X}_{\text{old}}$ only differ in hyperparameters, not in hyperparameter ranges, so we can decompose them as $\mathcal{X}_{\text{new}} = \mathcal{X}_{\text{only-new}} \times \mathcal{X}_{\text{both}}$ and $\mathcal{X}_{\text{old}} = \mathcal{X}_{\text{both}} \times \mathcal{X}_{\text{only-old}}$, where $\mathcal{X}_{\text{both}}$ is the part of the hyperparameter space that remains unchanged across adjustments (see Figure 4 for reference). The core idea of our algorithm is to project the hyperparameter settings that were evaluated in the old HPO from $\mathcal{X}_{\text{old}}$ to $\mathcal{X}_{\text{both}}$. We sample over $\mathcal{X}_{\text{both}}$ from a TPE model on the projected results of the previous HPO, and for $\mathcal{X}_{\text{only-new}}$ we use a random sample (Figure 4). Once there are enough evaluations to fit a TPE model for the new HPO, we fit and use this new TPE model.

A range adjustment can remove values from the hyperparameter range, or add values. For an adjustment of hyperparameter range $\mathcal{X}_{old}^{H_i}$ to $\mathcal{X}_{new}^{H_i}$ this can be expressed as $\mathcal{X}_{new}^{H_i} = \mathcal{X}_{\text{both}}^{H_i} \cup \mathcal{X}_{\text{both,range-only-new}}^{H_i}$ with $\mathcal{X}_{\text{both}}^{H_i} = \mathcal{X}_{old}^{H_i} \setminus \mathcal{X}_{\text{both,range-only-old}}^{H_i}$. We handle range removals ($\mathcal{X}_{\text{both,range-only-old}}^{H_i} \neq \emptyset$) separately from range addition ($\mathcal{X}_{\text{both,range-only-new}}^{H_i} \neq \emptyset$). To handle range removals, T2PE ignores hyperparameter settings from the previous HPO that have hyperparameter values in $\mathcal{X}_{\text{both,range-only-old}}$ when forming the model M_{both}. The main idea in how we handle additions to ranges, is to guarantee that each added range $\mathcal{X}_{\text{both,range-only-new}}^{H_i}$ is sampled with probability proportional to its size with respect to $|\mathcal{X}_{\text{new}}^{H_i}|$, i.e., with probability $p_i = \frac{|\mathcal{X}_{\text{both,range-only-new}}^{H_i}|}{|\mathcal{X}_{\text{new}}^{H_i}|}$. To guarantee this property, T2PE first samples $\mathbf{x}_{\text{both}}$ from $\mathcal{X}_{\text{both}}$ according to M_{both}, then mutates $\mathbf{x}_{\text{both}}^i$ with probability p_i to a random sample from $\mathcal{X}_{\text{both,range-only-new}}^{H_i}$.

5 Experiments and Results

In this section, we empirically evaluate the four baseline algorithms as solutions for the hyperparameter transfer across adjustments problem. Our main experimental focus is on the speed up of the transfer strategies over TPE. Further, in Appendix F we show the results of a control study that compares TPE with different ranges of random seeds; and in Appendix G we compare random search to TPE. We first describe the benchmark scenarios and evaluation protocol used through all studies, and then present the results.

Benchmarks In our experiments we use eight benchmarks described in the following. As is common in hyperparameter optimization research, we employ tabular and surrogate benchmarks to allow computationally feasible benchmarking (Perrone et al., 2018; Falkner et al., 2018). Tabular benchmarks provide a lookup table for all possible hyperparameter settings, whereas surrogate benchmarks model the objective function (Eggensperger et al., 2014). We base our benchmarks on four existing hyperparameter optimization (HPO) benchmarks (Perrone et al., 2018; Klein and Hutter, 2019; Dong and Yang, 2019), which amount to four different algorithms: a fully connected neural network (FCN), neural architecture search for a convolutional neural network (NAS), a support vector machine (SVM), and XGBoost (XGB). For each of these base benchmarks, we consider two different types of adjustments (Table 1) to arrive at a total of eight benchmarks. Additionally, for each algorithm and adjustment we consider multiple tasks. We refer the reader to Appendix C for additional details on the benchmarks.

Evaluation Protocol We repeated all measurements across 100 different random seeds and report results for validation objectives, as not all benchmarks make test objectives available, and to reduce noise in our evaluation. We measured how much faster an approach A reaches a given objective value compared to approach B in terms of number of evaluations. We terminate runs after 400 evaluations and report ratio of means. To determine the target objective values, we measured TPE's average performance for 10, 20, and 40 evaluations. Further, for transfer approaches, we perform this experiment for different evaluation budgets for the HPO before the adjustments (also for 10, 20, and 40 evaluations).

Table 1: Developer adjustments in benchmarks

Benchmark	Adjustments
FCN-A	Increase #units-per-layer 16×; Double #epochs; Fix batch size hyperparameter
FCN-B	Introduce per-layer choice of activation function; Change learning rate schedule from constant to cosine decay
NAS-A	Add 3x3 average pooling as choice of operation to each edge
NAS-B	Add node to cell template (adds 3 hyperparameters)
XGB-A	Expose four booster hyperparameters
XGB-B	Change four unexposed booster hyperparameter values
SVM-A	Change kernel; Remove hyperparameter for old kernel; Introduce hyperparameter for new kernel
SVM-B	Increase range for cost hyperparameter

Results The transfer TPE (T2PE) and best-first strategy lead to large speedups, while drop-unimportant and only-optimize-new perform poorly. On average, T2PE reaches the given objective values about 1.5x faster than TPE, and best-first about 2x faster. (Figure 3). There are two main trends visible: (1) The more optimal the target objective, the smaller the speedup, and (2) the higher the budget for the previous HPO, the higher the speedup. For a more fine-grained visualization that shows violin plots over task means for each benchmark, we refer to Appendix D. Even while given 10x the budget compared to TPE, drop-unimportant and only-optimize-new do not reach the performance of TPE in a large percentage of cases (about one third). (Figure 10). These high failure rates make an evaluation for the speed up unfeasible. For the failure rates all approaches we refer the reader to Appendix E.

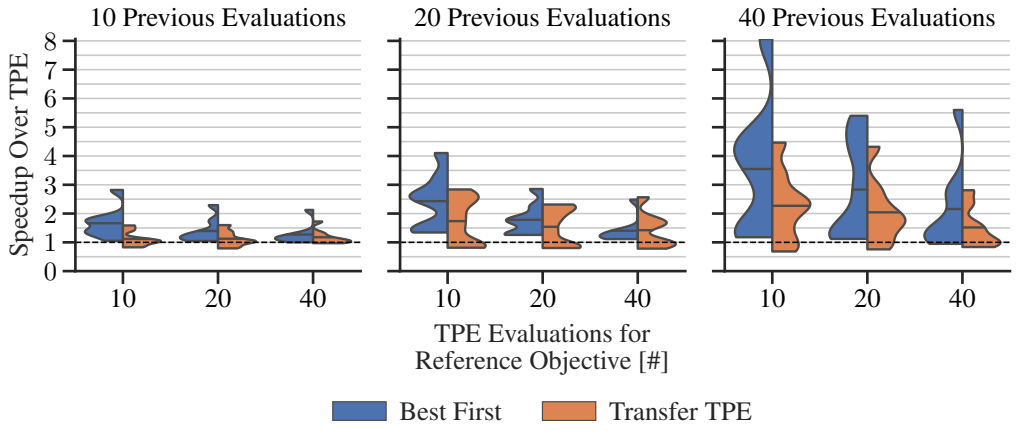

Figure 3: Speedup to reach a given reference objective value compared to TPE for best-first and transfer TPE across 8 benchmarks. The violins estimate densities of benchmark means. The horizontal line in each violin shows the mean across these benchmark means. Plots from left to right increase in budget for the pre-adjustment hyperparameter optimization.

6 Conclusion

In this work, we introduced hyperparameter transfer across developer adjustments to improve the efficiency during the development of machine learning systems. In light of rising energy demands of machine learning (ML) systems and rising global temperatures, more efficient ML development practices are an important issue now and will become more important in the future. As already one of the simple baseline algorithm considered in this work leads to large empirical speed ups, our new framework represents a promising step towards efficient ML development.

References

Agostinelli, F., Hoffman, M., Sadowski, P., and Baldi, P. (2014). Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*.

Bergstra, J., Yamins, D., and Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International conference on machine learning*, pages 115–123.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Chen, T., Goodfellow, I., and Shlens, J. (2015). Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*.

Chen, Y., Huang, A., Wang, Z., Antonoglou, I., Schrittwieser, J., Silver, D., and de Freitas, N. (2018). Bayesian optimization in alphago. *arXiv preprint arXiv:1812.06855*.

Day, O. and Khoshgoftaar, T. M. (2017). A survey on heterogeneous transfer learning. *Journal of Big Data*, 4(1):29.

Dong, X. and Yang, Y. (2019). Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations*.

Eggensperger, K., Hutter, F., Hoos, H., and Leyton-Brown, K. (2014). Surrogate benchmarks for hyperparameter optimization. In *ECAI workshop on Metalearning and Algorithm Selection (MetaSel'14)*.

Falkner, S., Klein, A., and Hutter, F. (2018). BOHB: Robust and efficient hyperparameter optimization at scale. In *Proceedings of the 35th International Conference on Machine Learning (ICML 2018)*, pages 1436–1445.

Feurer, M. and Hutter, F. (2018). Hyperparameter optimization. In Hutter, F., Kotthoff, L., and Vanschoren, J., editors, *Automatic Machine Learning: Methods, Systems, Challenges*, pages 3–38. Springer. In press, available at http://automl.org/book.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

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.

Kersting, K., Plagemann, C., Pfaff, P., and Burgard, W. (2007). Most likely heteroscedastic gaussian process regression. In *Proceedings of the 24th international conference on Machine learning*, pages 393–400.

Khan, N., Goldberg, D. E., and Pelikan, M. (2002). Multi-objective bayesian optimization algorithm. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pages 684–684. Citeseer.

Klein, A. and Hutter, F. (2019). Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*.

Lange, M. D., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. (2020). A continual learning survey: Defying forgetting in classification tasks.

Li, H., Chaudhari, P., Yang, H., Lam, M., Ravichandran, A., Bhotika, R., and Soatto, S. (2020). Rethinking the hyperparameters for fine-tuning. In *International Conference on Learning Representations*.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017). Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In OpenReview.net (2017).

Musgrave, K., Belongie, S., and Lim, S.-N. (2020). A metric learning reality check. *arXiv preprint arXiv:2003.08505*.

OpenReview.net, editor (2017). *Proceedings of the International Conference on Learning Representations (ICLR'17)*.

Perrone, V., Jenatton, R., Seeger, M. W., and Archambeau, C. (2018). Scalable hyperparameter transfer learning. In *Advances in Neural Information Processing Systems*, pages 6845–6855.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. (2019). Green ai. *arXiv preprint arXiv:1907.10597*.

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In Bartlett, P., Pereira, F., Burges, C., Bottou, L., and Weinberger, K., editors, *Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NIPS'12)*, pages 2960–2968.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3645–3650, Florence, Italy. Association for Computational Linguistics.

Thrun, S. and Mitchell, T. M. (1995). Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1):25 – 46. The Biology and Technology of Intelligent Autonomous Agents.

Vanschoren, J. (2018). Meta-learning: A survey. *arXiv preprint arXiv:1810.03548*.

Wu, Y. and He, K. (2018). Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2019). A comprehensive survey on transfer learning. *arXiv preprint arXiv:1911.02685*.

A Transfer TPE Sampling Illustration

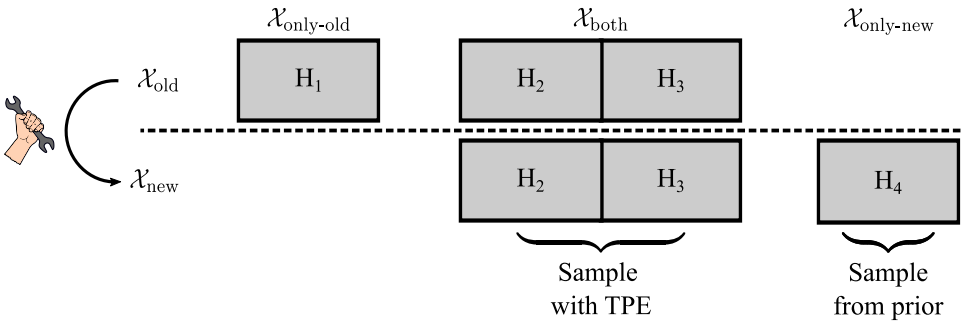

Figure 4: Sampling in T2PE for hyperparameter additions / removals.

B Pseudocode

Algorithm 1 Sampling strategy in transfer TPE

Input: Current hyperparameter space $\mathcal{X}_{\text{new}}$, previous hyperparameter space $\mathcal{X}_{\text{old}}$, config ranking of previous optimization $\mathcal{C}$

Decompose $\mathcal{X}_{\text{new}} = (\mathcal{X}_{\text{both}} \cup \mathcal{X}_{\text{both,range-only-new}}) \times \mathcal{X}_{\text{only-new}}$
Discard configs in $\mathcal{C}$ that have hyperparameter values in $\mathcal{X}_{\text{both,range-only-new}}$
Project configs in $\mathcal{C}$ to space $\mathcal{X}_{\text{both}}$, to yield config ranking $\mathcal{C}_{\text{both}}$
Fit TPE model M_{both} for $\mathcal{X}_{\text{both}}$ on $\mathcal{C}_{\text{both}}$
for t **in** $1, \ldots, N$ **do**
    if is random fraction **then** ▷ From TPE implementation, e.g., 1/3 of cases
        Sample $\mathbf{x}_{\text{new}}$ from prior on $\mathcal{X}_{\text{new}}$
    else if no model for $\mathcal{X}_{\text{new}}$ **then**
        Sample $\mathbf{x}_{\text{both}}$ from $\mathcal{X}_{\text{both}}$ according to M_{both}
        for hyperparameter range $\mathcal{X}^{H_i}_{\text{both,range-only-new}} \neq \emptyset$ **in** $\mathcal{X}_{\text{both,range-only-new}}$ **do**
            Set $p := \frac{|\mathcal{X}^{H_i}_{\text{both,range-only-new}}|}{|\mathcal{X}^{H_i}_{\text{new}}|}$
            Sample x^i from prior on $\mathcal{X}^{H_i}_{\text{both,range-only-new}}$
            Set $\mathbf{x}^i_{\text{both}} := \mathrm{x}^i$ with probability p
        Sample $\mathbf{x}_{\text{only-new}}$ from prior on $\mathcal{X}_{\text{only-new}}$
        Combine $\mathbf{x}_{\text{both}}$ with $\mathbf{x}_{\text{only-new}}$ to yield sample $\mathbf{x}_{\text{new}}$
    else
        Fit TPE model M_{new} for $\mathcal{X}_{\text{new}}$ on current observations
        Sample $\mathbf{x}_{\text{new}}$ from $\mathcal{X}_{\text{new}}$ according to M_{new}
return

C Benchmark Suite Details

C.1 Overview

Table 2: Benchmarks overview

Benchmark	#Hyperparameters Old	#Hyperparameters New	#Tasks
FCN-A	6	5	4
FCN-B	6	8	4
NAS-A	6	6	3
NAS-B	3	6	3
XGB-A	5	9	10
XGB-B	6	6	10
SVM-A	2	2	10
SVM-B	2	2	10

C.2 FCN-A & FCN-B

Budget For FCN-A the budget is set to 100. For FCN-B, additional to the changes in the search space (Table 5), the budget is increased from 50 to 100.

Table 3: Values for integer coded hyperparameters in FCN benchmarks

Hyperparameter	Values
# Units Layer $\{1, 2\}$	$(16, 32, 64, 128, 256, 512)$
Dropout Layer $\{1, 2\}$	$(0.0, 0.3, 0.6)$
Initial Learning Rate	$(0.0005, 0.001, 0.005, 0.01, 0.05, 0.1)$
Batch Size	$(8, 16, 32, 64)$

Table 4: Search spaces in FCN-A. Numerical hyperparameters are encoded as integers, see Table 3 for specific values for these hyperparameters.

Steps	Hyperparameter	Range/Value	Prior
1	# Units Layer 1	1	-
1	# Units Layer 2	1	-
1	Batch Size	$\{0, \ldots, 3\}$	Uniform
1, 2	Dropout Layer 1	$\{0, \ldots, 2\}$	Uniform
1, 2	Dropout Layer 2	$\{0, \ldots, 2\}$	Uniform
1, 2	Activation Layer 1	{ReLu, tanh}	Uniform
1, 2	Activation Layer 2	{ReLu, tanh}	Uniform
1, 2	Initial Learning Rate	$\{0, \ldots, 5\}$	Uniform
1, 2	Learning Rate Schedule	Constant	Uniform
2	# Units Layer 1	5	-
2	# Units Layer 2	5	-
2	Batch Size	1	-

Table 5: Search spaces in FCN-B. Numerical hyperparameters are encoded as integers, see Table 3 for specific values for these hyperparameters.

Steps	Hyperparameter	Range/Value	Prior
1	Activation Layer 1	tanh	-
1	Activation Layer 2	tanh	-
1	Learning Rate Schedule	Constant	-
1, 2	# Units Layer 1	$\{0, \ldots, 5\}$	Uniform
1, 2	# Units Layer 2	$\{0, \ldots, 5\}$	Uniform
1, 2	Dropout Layer 1	$\{0, \ldots, 2\}$	Uniform
1, 2	Dropout Layer 2	$\{0, \ldots, 2\}$	Uniform
1, 2	Initial Learning Rate	$\{0, \ldots, 5\}$	Uniform
1, 2	Batch Size	$\{0, \ldots, 3\}$	Uniform
2	Activation Layer 1	{ReLu, tanh}	Uniform
2	Activation Layer 2	{ReLu, tanh}	Uniform
2	Learning Rate Schedule	Cosine	-

C.3 NAS-A & NAS-B

Table 6: Search spaces in NAS-A.

Steps	Hyperparameter	Range/Value	Prior
1, 2	$0 \to 2$	{ none, skip-connect, conv1x1, conv3x3, avg-pool3x3 }	Uniform
1, 2	$0 \to 3$	{ none, skip-connect, conv1x1, conv3x3, avg-pool3x3 }	Uniform
1, 2	$2 \to 3$	{ none, skip-connect, conv1x1, conv3x3, avg-pool3x3 }	Uniform
2	$0 \to 1$	{ none, skip-connect, conv1x1, conv3x3, avg-pool3x3 }	Uniform
2	$1 \to 2$	{ none, skip-connect, conv1x1, conv3x3, avg-pool3x3 }	Uniform
2	$1 \to 3$	{ none, skip-connect, conv1x1, conv3x3, avg-pool3x3 }	Uniform

Table 7: Search spaces in NAS-B.

Steps	Hyperparameter	Range/Value	Prior
1	$0 \to 1$	{ none, skip-connect, conv1x1, conv3x3 }	Uniform
1	$0 \to 2$	{ none, skip-connect, conv1x1, conv3x3 }	Uniform
1	$0 \to 3$	{ none, skip-connect, conv1x1, conv3x3 }	Uniform
1	$1 \to 2$	{ none, skip-connect, conv1x1, conv3x3 }	Uniform
1	$1 \to 3$	{ none, skip-connect, conv1x1, conv3x3 }	Uniform
1	$2 \to 3$	{ none, skip-connect, conv1x1, conv3x3 }	Uniform
2	$0 \to 1$	{ none, skip-connect, conv1x1, conv3x3, avg-pool3x3 }	Uniform
2	$0 \to 2$	{ none, skip-connect, conv1x1, conv3x3, avg-pool3x3 }	Uniform
2	$0 \to 3$	{ none, skip-connect, conv1x1, conv3x3, avg-pool3x3 }	Uniform
2	$1 \to 2$	{ none, skip-connect, conv1x1, conv3x3, avg-pool3x3 }	Uniform
2	$1 \to 3$	{ none, skip-connect, conv1x1, conv3x3, avg-pool3x3 }	Uniform
2	$2 \to 3$	{ none, skip-connect, conv1x1, conv3x3, avg-pool3x3 }	Uniform

C.4 SVM-A & SVM-B

Table 8: Search spaces in SVM-A.

Steps	Hyperparameter	Range/Value	Prior
1	Kernel	Radial	-
1	Degree	$\{2, \ldots, 5\}$	Uniform
1, 2	Cost	$[2^{-10}, 2^{10}]$	Log-uniform
2	Kernel	Polynomial	-
2	γ	$[2^{-5}, 2^{5}]$	Log-uniform

Table 9: Search spaces in SVM-B.

Steps	Hyperparameter	Range/Value	Prior
1	Cost	$[2^{-5}, 2^{5}]$	Log-uniform
1, 2	γ	1	-
1, 2	Degree	5	-
1, 2	Kernel	{Polynomial, Linear, Radial}	Uniform
2	Cost	$[2^{-10}, 2^{10}]$	Log-uniform

C.5 XGB-A & XGB-B

Table 10: Search spaces in XGB-A

Steps	Hyperparameter	Range/Value	Prior
1	Colsample-by-tree	1	-
1	Colsample-by-level	1	-
1	Minimum child weight	1	-
1	Maximum depth	6	-
1, 2	Booster	Tree	-
1, 2	# Rounds	$\{1, \ldots, 5,000\}$	Uniform
1, 2	Subsample	$[0, 1]$	Uniform
1, 2	Eta	$[2^{-10}, 2^{0}]$	Log-uniform
1, 2	Lambda	$[2^{-10}, 2^{10}]$	Log-uniform
1, 2	Alpha	$[2^{-10}, 2^{10}]$	Log-uniform
2	Colsample-by-tree	$[0, 1]$	Uniform
2	Colsample-by-level	$[0, 1]$	Uniform
2	Minimum child weight	$[2^{0}, 2^{7}]$	Log-uniform
2	Maximum depth	$\{1, \ldots, 15\}$	Uniform

Table 11: Search spaces in XGB-B

Steps	Hyperparameter	Range/Value	Prior
1	Colsample-by-tree	1	-
1	Colsample-by-level	1	-
1	Minimum child weight	1	-
1	Maximum depth	6	-
1, 2	Booster	{ Linear, Tree }	-
1, 2	# Rounds	$\{1, \ldots, 5,000\}$	Uniform
1, 2	Subsample	$[0, 1]$	Uniform
1, 2	Eta	$[2^{-10}, 2^{0}]$	Log-uniform
1, 2	Lambda	$[2^{-10}, 2^{10}]$	Log-uniform
1, 2	Alpha	$[2^{-10}, 2^{10}]$	Log-uniform
2	Colsample-by-tree	1	-
2	Colsample-by-level	0.5	-
2	Minimum child weight	10	-
2	Maximum depth	10	-

D Speed Ups

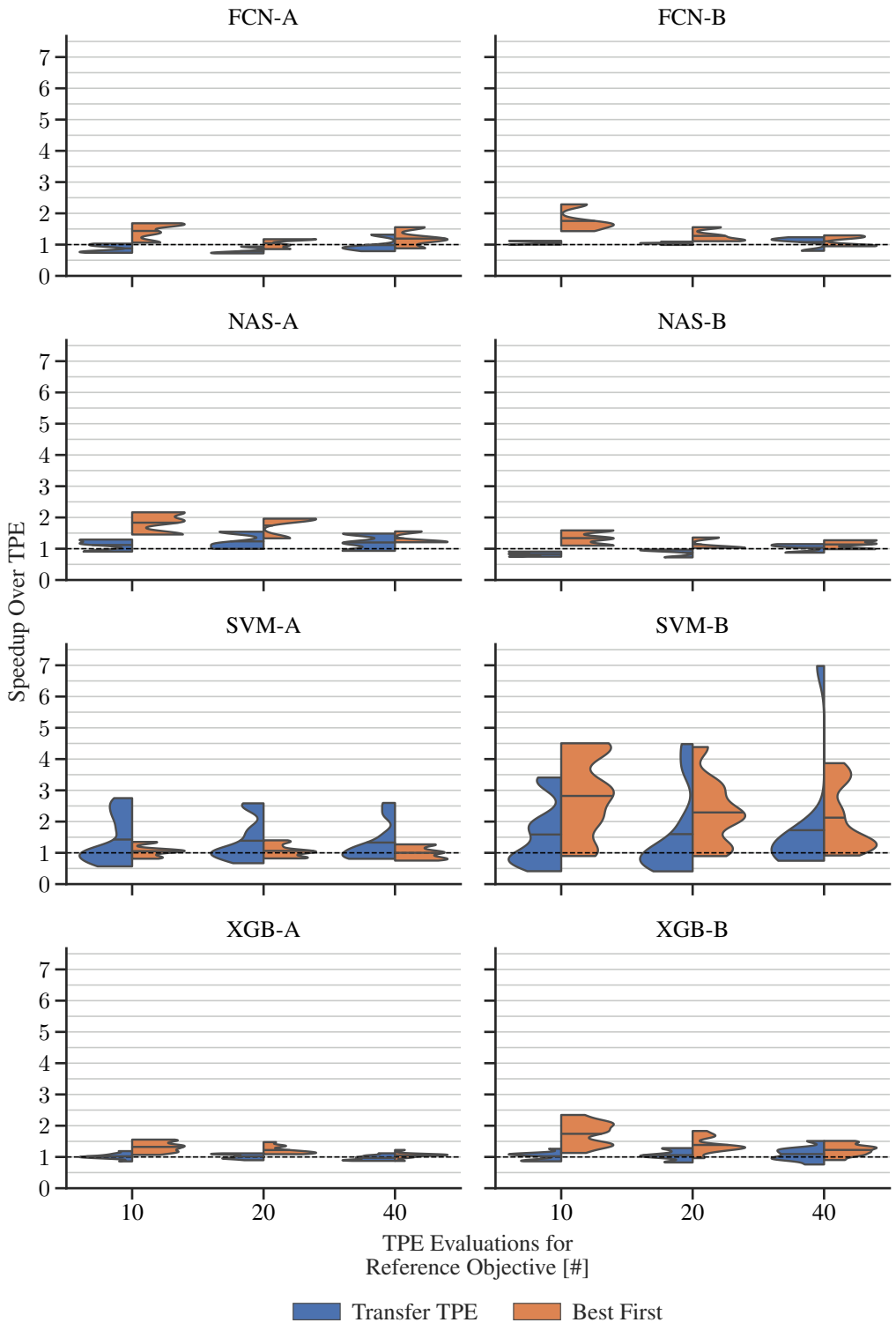

Figure 5: Speedup of transfer TPE and best-first over TPE across tasks for each of 8 benchmarks. The previous HPO has a budget of 10 evaluations. The violins estimate densities of the task means. The horizontal line in each violin shows the mean across these task means. In each plot, the budget for the TPE reference increases.

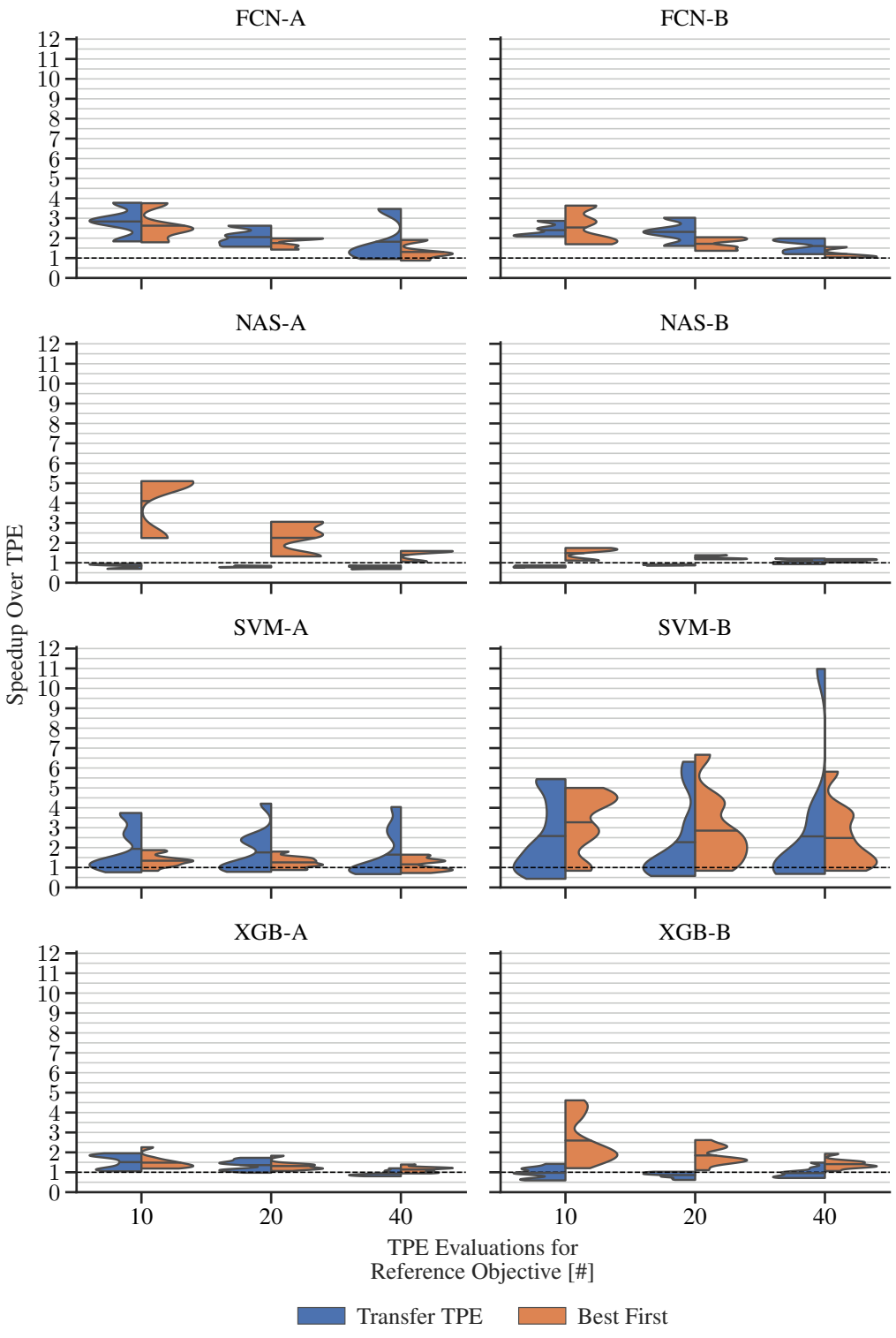

Figure 6: Speedup of transfer TPE and best-first over TPE across tasks for each of 8 benchmarks. The previous HPO has a budget of 20 evaluations. The violins estimate densities of the task means. The horizontal line in each violin shows the mean across these task means. In each plot, the budget for the TPE reference increases.

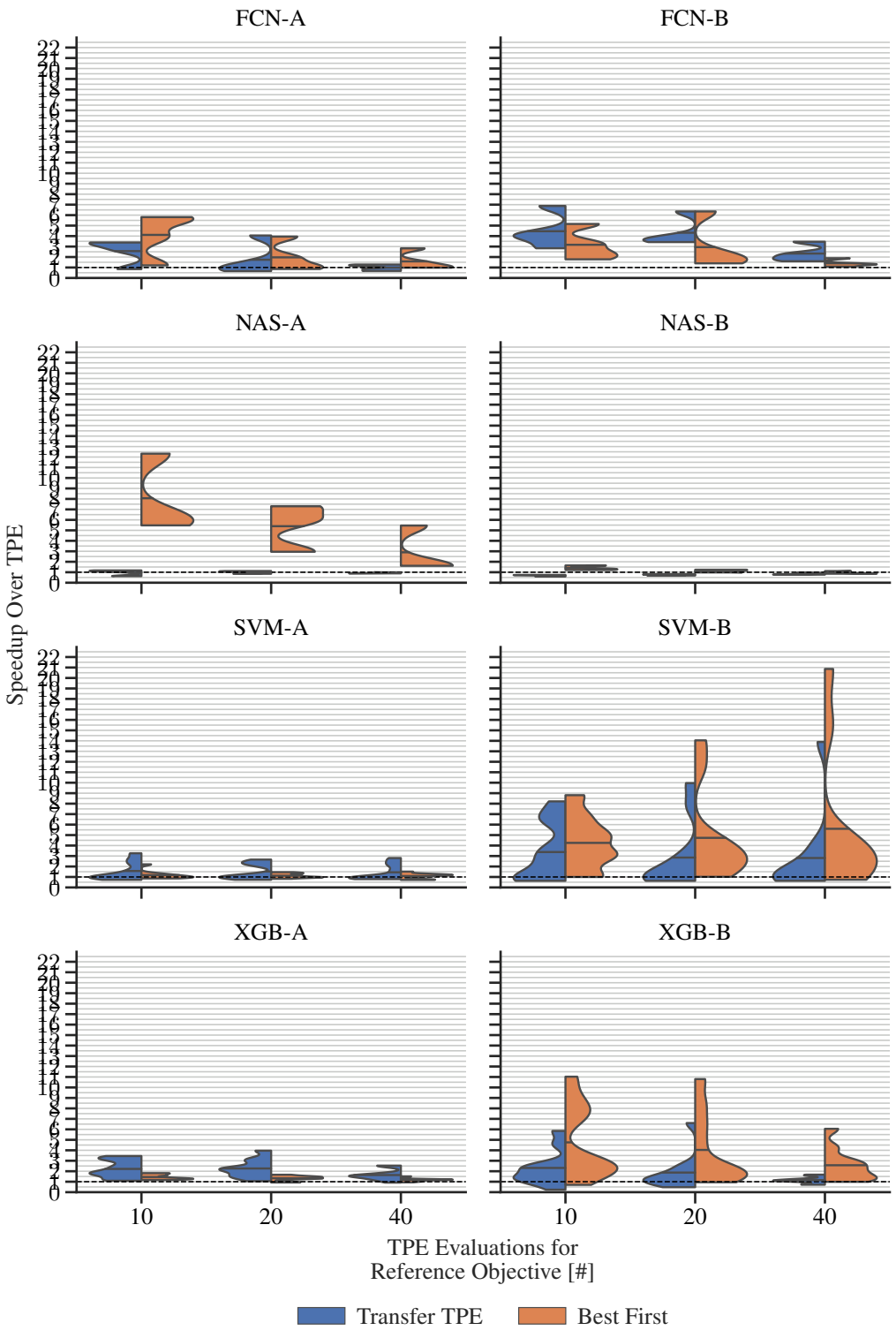

Figure 7: Speedup of transfer TPE and best-first over TPE across tasks for each of 8 benchmarks. The previous HPO has a budget of 40 evaluations. The violins estimate densities of the task means. The horizontal line in each violin shows the mean across these task means. In each plot, the budget for the TPE reference increases.

E Failure Rates

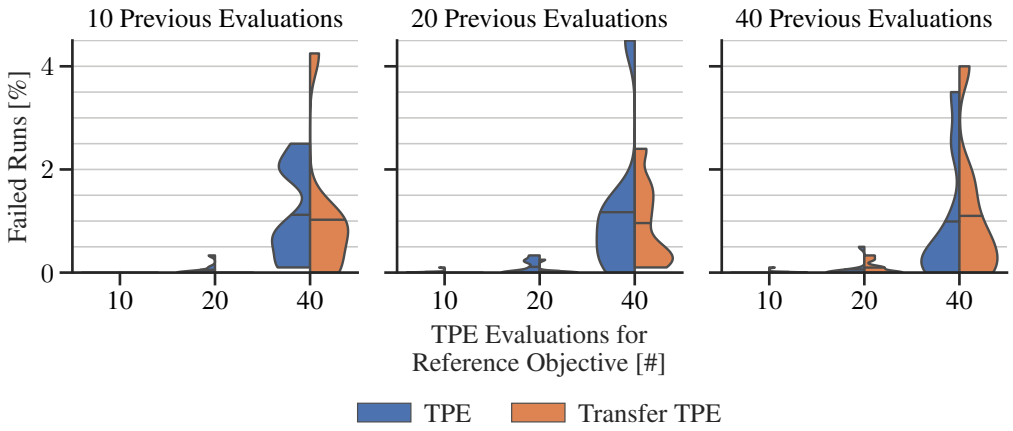

Figure 8: Failure rates for transfer TPE and TPE across 8 benchmarks. The violins estimate densities of the task means. The horizontal line in each violin shows the mean across these task means. The plots from left to right utilize increasing budget for the pre-adjustment hyperparameter. In each plot, the budget for the TPE reference increases.

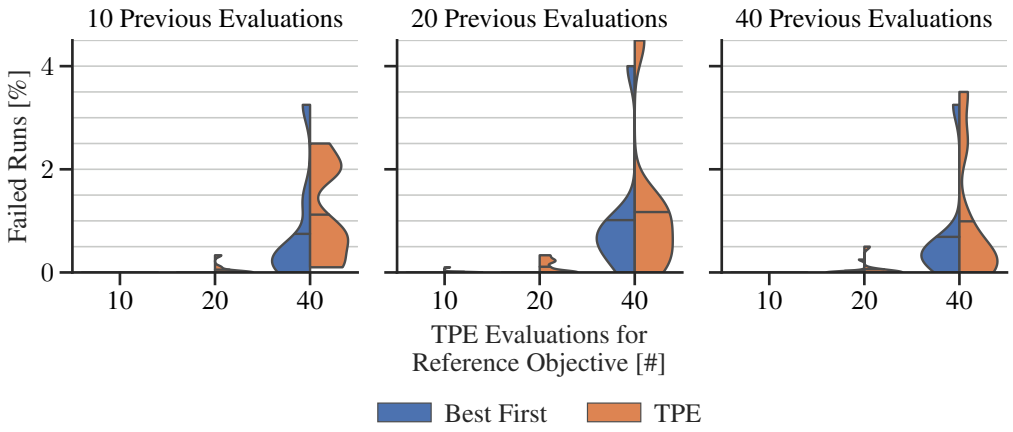

Figure 9: Failure rates for best-first and TPE across 8 benchmarks. The violins estimate densities of task means. The horizontal line in each violin shows the mean across these task means. The plots from left to right utilize increasing budget for the pre-adjustment hyperparameter optimization. In each plot, the budget for the TPE reference increases.

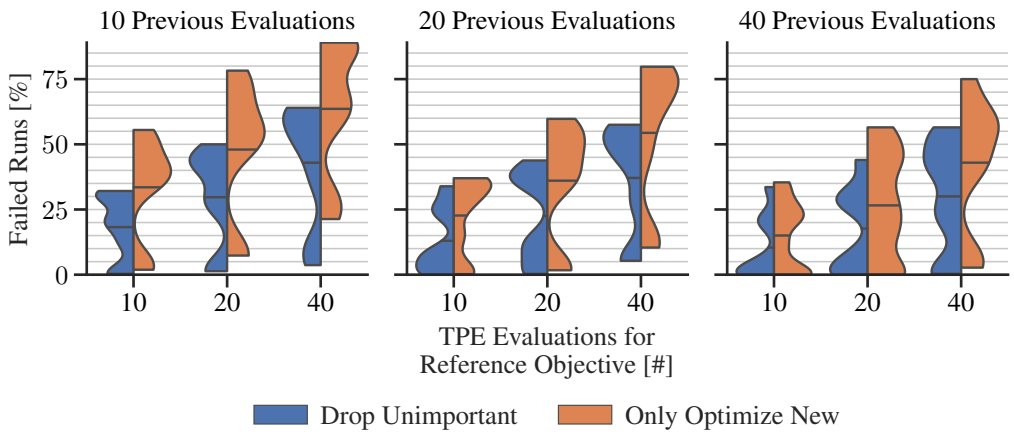

Figure 10: Percent of runs that never reach the reference objective for the drop-unimporant and only-optimize-new approach. Each data point for the violins represents the mean percentage of failures for a benchmark. The line in each violin shows the mean across these benchmark means. Plots from left to right increase in budget for the pre-adjustment hyperparameter optimization. In each plot, the budget of the TPE reference increases.

F Control Study: TPE for Different Random Seed Ranges

As a sanity check, and to gauge the influence of random seeds, we compare TPE to itself with different seed ranges. In general we observe little differences in TPE and TPE2, with the exception of one outlier task (Figure 11).

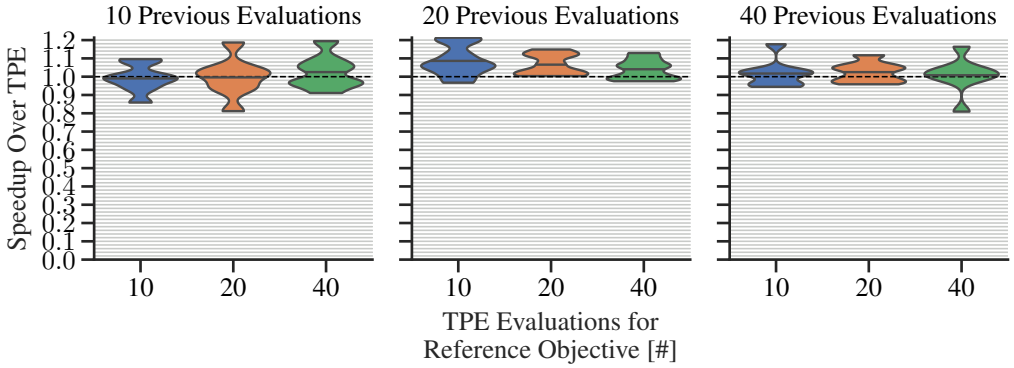

Figure 11: Speedup of TPE over TPE2 across 8 benchmarks. The violins estimate densities of the benchmark means. The horizontal line in each violin shows the mean across these benchmark means. The plots from left to right utilize increasing budget for the pre-adjustment hyperparameter optimization. In each plot, the budget for the TPE reference increases.

G Control Study: Random Search vs TPE

As a sanity check, and for context, we compare TPE to random search (Figure 12).

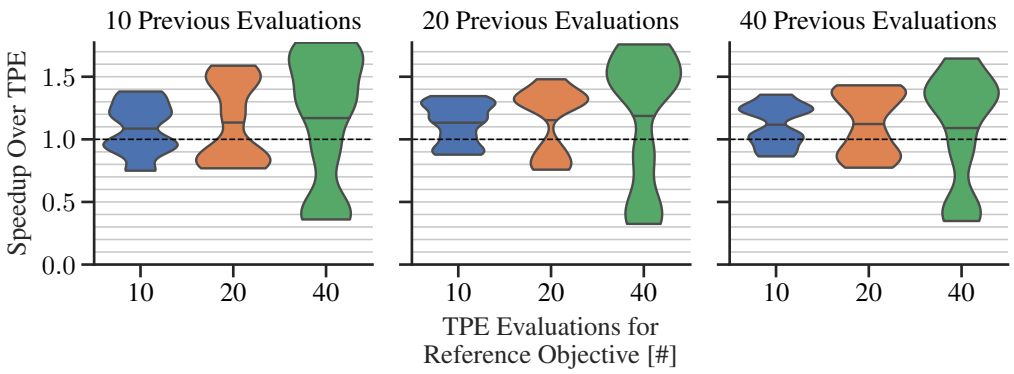

Figure 12: Speedup of random search over TPE across 8 benchmarks. The violins estimate densities of the benchmark means. The horizontal line in each violin shows the mean across these benchmark means. The plots from left to right utilize increasing budget for the pre-adjustment hyperparameter optimization. In each plot, the budget for the TPE reference increases.