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. A combination of the best two baselines, on average and depending on the budgets involved, reaches a given performance 1.2–2.9x faster than a prominent HPO algorithm without transfer. As hyperparameter optimization is a crucial step in ML development but requires extensive computational resources, this speedup 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 algorithms.

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.
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, 2019), 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 CO₂ 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 (Section 5). - IV- Further, we provide open source code for our experiments benchmark and HT-AA baseline 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:

$$\min_{x \in \mathcal{X}} f_S(x) \text{ with } b \text{ trials},$$

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 (heterogeneous 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 $\Psi$ to a machine learning system $S_{\text{old}}$ 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 $b_{\text{new}}$, 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, whereas 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 \( l(x) \) and \( g(x) \), for the probability of a given hyperparameter configuration \( x \) being worse \((l)\), or better \((g)\), than the best already evaluated configuration \( x^* \). To decide which configuration to evaluate, TPE then solves

\[
arg \max_{x \in \mathcal{X}} \frac{g(x)}{b(x)}
\]

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 speedup 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 \( X_{\text{new}} \) and the old hyperparameter space \( X_{\text{old}} \) only differ in hyperparameters, not in hyperparameter ranges. so we can decompose them as \( X_{\text{new}} = X_{\text{only-new}} \times \lambda_{\text{both}} \) and \( X_{\text{old}} = \lambda_{\text{both}} \times X_{\text{only-old}} \), where \( \lambda_{\text{both}} \) is the part of the hyperparameter space that remains unchanged across adjustments (see figure in Appendix B for reference). The core idea of our algorithm is to project the hyperparameter settings that were evaluated in the old HPO from \( X_{\text{old}} \) to \( \lambda_{\text{both}} \). We sample over \( \lambda_{\text{both}} \) from a TPE model on the projected results of the previous HPO, and for \( X_{\text{only-new}} \) we use a random sample (Appendix B). 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 \( X_{\text{old}} \) to \( X_{\text{new}} \) this can be expressed as \( \lambda_{i}^{H_{\text{new}}} = \lambda_{i}^{H_{\text{old}}} \cup \lambda_{i}^{H_{\text{both,range-only-new}}} \) with \( \lambda_{i}^{H_{\text{both,range-only-new}}} = \lambda_{i}^{H_{\text{old}}} \backslash \lambda_{i}^{H_{\text{both,range-only-old}}} \). We handle range removals \( \lambda_{i}^{H_{\text{both,range-only-old}}} \neq \emptyset \) separately from range addition \( \lambda_{i}^{H_{\text{both,range-only-new}}} \neq \emptyset \). To handle range removals, T2PE ignores hyperparameter settings from the previous HPO that have hyperparameter values in \( \lambda_{i}^{H_{\text{both,range-only-old}}} \) when forming the model \( M_{\text{both}} \). The main idea in how we handle additions to ranges, is to guarantee that each added range \( \lambda_{i}^{H_{\text{both,range-only-new}}} \) is sampled with probability proportional to its size with respect to \( |X_{\text{new}}| \), i.e.,

\[
\lambda_{i}^{H_{\text{both,range-only-new}}} \text{ is sampled with probability } \frac{|X_{\text{both,range-only-new}}|}{|X_{\text{new}}|}.
\]

To guarantee this property, T2PE first samples \( \lambda_{i}^{H_{\text{both}}} \) from \( \lambda_{i}^{H_{\text{both}}} \) according to \( M_{\text{both}} \), then mutates \( \lambda_{i}^{H_{\text{both}}} \) with probability \( p_{i} \) to a random sample from \( \lambda_{i}^{H_{\text{both,range-only-new}}} \).

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 speedup 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 aggregate these ratios across tasks and benchmarks, we use the geometric mean. 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 our benchmarks.

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

**Results** Transfer TPE (T2PE), best-first, and their combination, lead to large speedups, while drop-unimportant and only-optimize-new perform poorly. On average and depending on the budgets for the old and new HPO, T2PE reaches the given objective values 1.0–1.7x faster than TPE, best-first 1.2–2.6x faster, and their combination 1.2–2.9x 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 (20–70%; Appendix E); this makes an evaluation for the speedup unfeasible.

![Figure 3: Speedup to reach a given reference objective value compared to TPE for best-first, transfer TPE, and their combination, across 8 benchmarks.](image)

**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 some of the simple baseline algorithm considered in this work leads to large empirical speedups, our new framework represents a promising step towards efficient ML development.
Acknowledgments and Disclosure of Funding

The authors acknowledge support by the state of Baden-Württemberg through bwHPC and the German Research Foundation (DFG) through grant no INST 39/963-1 FUGG. Robert Bosch GmbH is acknowledged for financial support. This work has partly been supported by the European Research Council (ERC) under the European Union’s Horizon 2020 research and innovation programme under grant no. 716721.

References


