Meta-Learning for Hyperparameter Optimization

Martin Wistuba Josif Grabocka

Amazon Web Services University of Freiburg

13 September 2023

Outline

"Textbook" ML: A Simplified Skeleton

Data:

- Training dataset with features x^{train} and target y^{train}

Model:

- Prediction model $f(x;\theta)$ with parameters $\theta \in \Theta$

Problem:

- Loss $\mathcal{L}\left(y^{\text{train}}, f\left(x^{\text{train}}\right)\right)$ abbreviated as $\mathcal{L}^{\text{train}}(\theta)$
- Objective: $\theta^* := \arg\min_{\theta \in \Theta} \mathcal{L}^{\text{train}}(\theta)$

"Real-world" ML: Lots of Hyperparameters

- The "textbook" simplified supervised learning definition is of little practical use.
- ML methods require several design choices before we can optimize the parameters.
  - Preprocessing
  - Data Augmentation
  - Model and Neural Architecture
  - Regularization
  - Optimization
- Entirety of design choices are called **hyperparameters**

Search Spaces of Hyperparameter Configurations

An example search space Λ:

Hyperparameter	**Range**	**Scale**
Architecture	{ConvNext, ViT, EfficientNet}	Discrete
Dropout	$[0.0, 1.0]$	Uniform
Optimizer	{SGD, Adam, RMSProp}	Discrete
Learning Rate	$\left[10^{-5}, 10^{0}\right]$	Log

A configuration $\lambda \in \Lambda$ is an element of the Cartesian product of hyperparameter ranges, e.g.:

$$\lambda = \left[\text{Arch.: ViT}, \text{Dropout} : 0.2, \text{Optim.: Adam}, \text{LR} : 10^{-4}\right]$$

Objective: **How** to find the optimal λ for a particular task?

Hyperparameter Optimization (HPO)

Hyperparameters $\lambda \in \Lambda$ where Λ is the design/search space.

- Effect: Parameters depend on hyperparameters θ_λ
- Goal: Find λ to minimize a validation loss $\mathcal{L}^{\text{val}}(\theta_\lambda)$

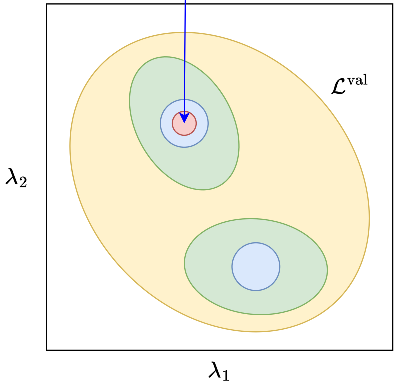
Hyperparameter optimization (HPO) problem:

$$\lambda^* := \arg\min_{\lambda \in \Lambda} \mathcal{L}^{\text{val}}(\theta^*_\lambda)$$
$$\text{s.t.} \quad \theta^*_\lambda := \arg\min_{\theta_\lambda \in \Theta} \mathcal{L}^{\text{train}}(\theta_\lambda)$$

For the sake of brevity, $\mathcal{L}^{\text{val}}(\theta^*_\lambda)$ can be alternatively denoted as $\ell(\lambda)$.

Difficulty of HPO

$$\lambda^* = \text{argmin}_{\lambda \in \Lambda} \mathcal{L}^{\text{val}}(\theta^*_\lambda)$$

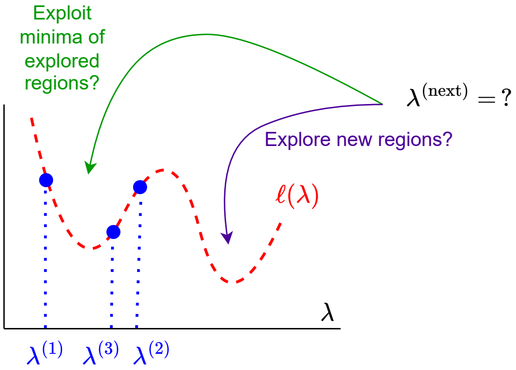


- $\mathcal{L}^{\text{val}}$ is non-convex
- $\mathcal{L}^{\text{val}}$ is expensive
- $\mathcal{L}^{\text{val}}$ is non-analytic

Outline

Outline

HPO as a sequential search



- HPO = sequential search
- **Given** evaluations $\left\{\lambda^{(t)}, \ell\left(\lambda^{(t)}\right)\right\}_{t=1}^{T}$
- **Which** $\lambda^{(\text{next})}$ to evaluate next?
- To explore, or to exploit, that is the question.

Flavors of HPO

- **Black-box**

  The only access to a function $\ell : \Lambda \rightarrow \mathbb{R}$ is by evaluating $\ell(\lambda)$ for any $\lambda \in \Lambda$.

- **Gray-box**

  Access to a function $\ell : \Lambda \rightarrow \mathbb{R}$ is by partial (low-cost) evaluations $\ell(\lambda)_b$ at a budget b.

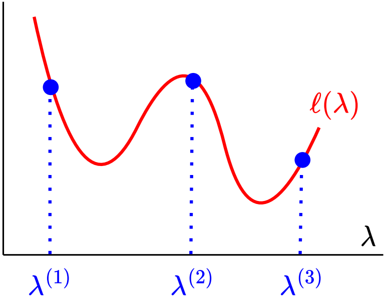
  In deep learning, additional access to model weights, layer activations, etc.

- **White-Box**:

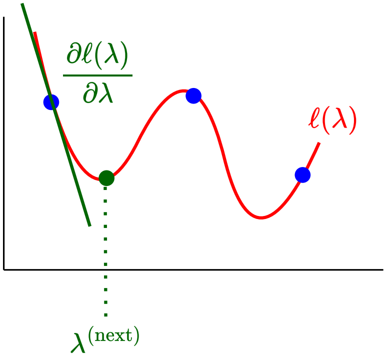
  In addition to the gray-box level of access, we can compute the gradients $\frac{\partial \ell(\lambda)}{\partial \lambda}$.

First-order optimization: Access to gradients

Function evaluations

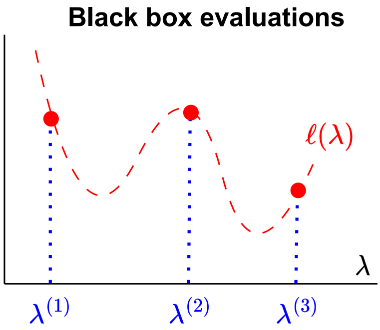


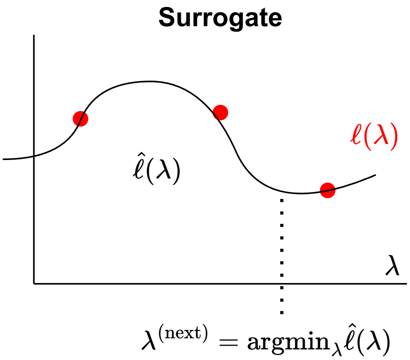
Next point to evaluate



Standard optimization of analytic functions with off-the-shelf first-/second-order techniques.

Black-box HPO: No access to gradients

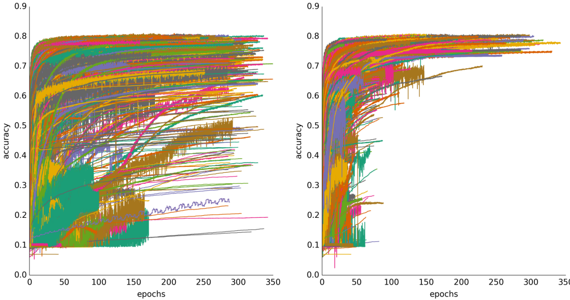




Black-box optimization of non-analytic functions through optimizable surrogates.

Gray-Box HPO

- Measure **approximately** $\ell(\lambda;\text{"budget"}) \approx \ell(\lambda)$:
  - Train on a **subset** of the dataset
  - Train for **fewer** epochs
  - Train for **less** ensemble models
- Rule out configurations after low-budget evaluations



White-Box HPO

Update parameters θ **and** hyperparameters λ jointly [1]:

$$\theta^{(t)} \leftarrow u\left(\theta^{(t-1)}, \lambda^{(t)}\right) \text{ and } \lambda^{(t+1)} \leftarrow \lambda^{(t)} - \eta \frac{\partial \mathcal{L}^{\text{val}}\left(\theta^{(t)}\right)}{\partial \lambda^{(t)}}$$

where:

$$\frac{\partial \mathcal{L}^{\text{val}}\left(\theta^{(t)}\right)}{\partial \lambda^{(t)}} = \frac{\partial \mathcal{L}^{\text{val}}\left(\theta^{(t)}\right)}{\partial \theta^{(t)}} \frac{\partial u\left(\theta^{(t-1)}, \lambda^{(t)}\right)}{\partial \lambda^{(t)}}$$

For instance, in the case where λ is the learning rate:

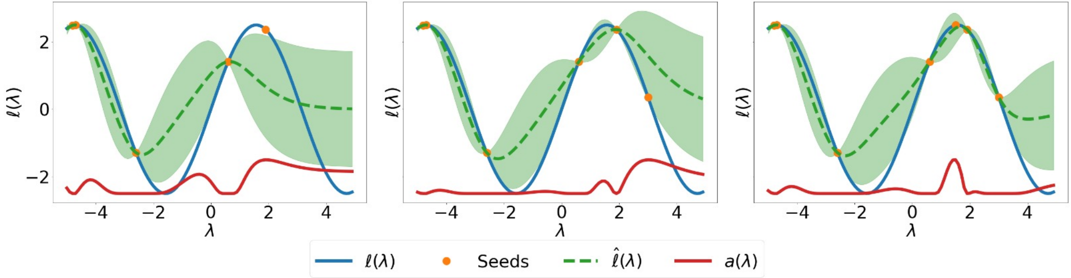
$$\frac{\partial u\left(\theta^{(t-1)}, \lambda^{(t)}\right)}{\partial \lambda^{(t)}} = \frac{\partial \left(\theta^{(t-1)} - \lambda^{(t)} \frac{\partial \mathcal{L}^{\text{train}}\left(\theta^{(t-1)}\right)}{\partial \theta^{(t-1)}}\right)}{\partial \lambda^{(t)}} = -\frac{\partial \mathcal{L}^{\text{train}}\left(\theta^{(t-1)}\right)}{\partial \theta^{(t-1)}}$$

Outline

Bayesian Optimization - Mechanism

Black-box policies for minimizing/maximizing functions $\ell(\lambda)$:

- **Given** $H := \{\lambda^{(i)}, \ell(\lambda^{(i)})\}_{i=1}^{n}$
- **Evaluate** $\lambda^{(\text{next})}$



The acquisition function a promotes regions where the surrogate $\hat{\ell}$ has both a high predicted mean and a high variance.

Bayesian Optimization - Algorithm

Algorithm 1: Bayesian Optimization

Initial design $H := \left\{\left(\lambda^{(i)}, \ell\left(\lambda^{(i)}\right)\right)\right\}_{i=1}^{n}$;

while *still budget remaining* **do**

Fit a probabilistic surrogate $\hat{\ell}$, e.g. surrogate $\hat{\ell} :=$ Gaussian-Process(H) ;

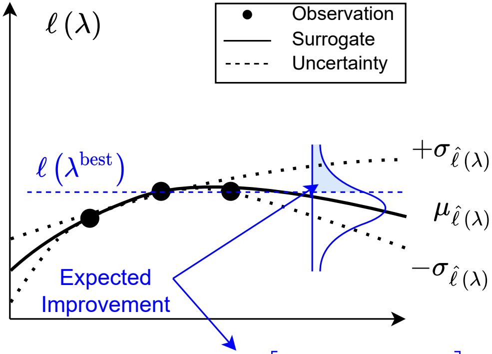
Recommend $\lambda^{\text{next}} := \arg\max_{\lambda}\ a\left(\hat{\ell}(\lambda)\right)$, e.g. acquisition a = Expected-Improvement ;

Evaluate $H \leftarrow H \cup \{(\lambda^{\text{next}}, \ell(\lambda^{\text{next}}))\}$

end

return $\lambda^* \leftarrow \arg\min_{(\lambda,\cdot)\in H} \ell(\lambda)$;

Typical Acquisition: Expected Improvement

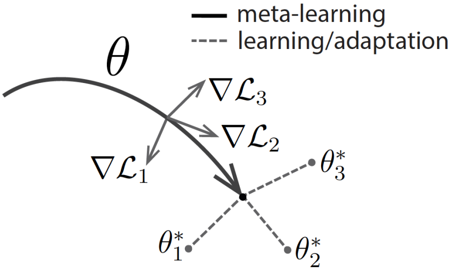


$$\lambda^{\text{next}} = \mathbf{argmax}_{\lambda}\ \mathbb{E}_{\hat{\ell}(\lambda)}\left[\hat{\ell}(\lambda) - \ell\left(\lambda^{best}\right)\right]$$

Outline

Model-Agnostic Meta-Learning

- Model learns from all the tasks
- Learn a representation that requires only few steps to the optimal representation for each task
- Performs well for few-shot learning problems



Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *ICML*. vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1126–1135

MAML Algorithm

Algorithm 2: Model-Agnostic Meta-Learning

Require: $p(\mathcal{T})$: distribution over tasks

Require: β, γ: step size hyperparameters

```
randomly initialize θ
while not done do
    Sample batch of tasks T_i ~ p(T)
    for all T_i do
        θ'_i = θ − β ∇_θ L_i(ℓ_θ)
    θ ← θ − γ ∇_θ Σ_{T_i~p(T)} L_i(ℓ_{θ'_i})
```

randomly initialize θ

while not done **do**

  Sample batch of tasks $\mathcal{T}_i \sim p(\mathcal{T})$

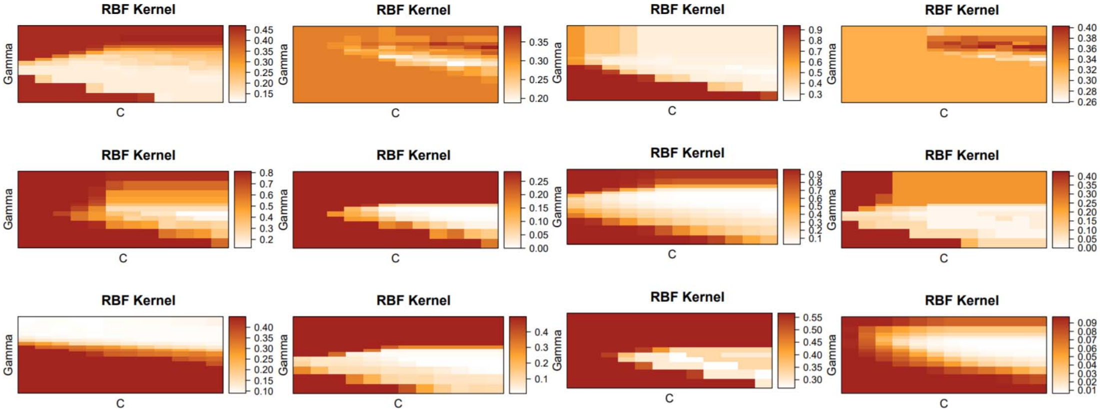
  for all $\mathcal{T}_i$ **do**

    $\theta'_i = \theta - \beta \nabla_\theta \mathcal{L}_i(\ell_\theta)$

  $\theta \leftarrow \theta - \gamma \nabla_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_i(\ell_{\theta'_i})$

Outline

Transfer Learning helps HPO



Response function of different datasets can look very similar.

Objective of Transfer Learning in HPO

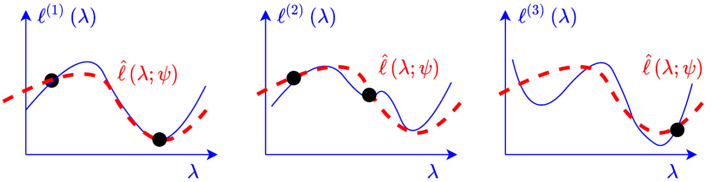
In transfer learning for HPO, we have access to a history of HPO runs (a meta-dataset),

$$\mathcal{H} = \Big\{ \left(\lambda^{(1)}, \ell^{(1)}\left(\lambda^{(1)}\right)\right), \ldots, \left(\lambda^{(n_1)}, \ell^{(1)}\left(\lambda^{(n_1)}\right)\right), \\ \ldots, \\ \left(\lambda^{(1)}, \ell^{(M)}\left(\lambda^{(1)}\right)\right), \ldots, \left(\lambda^{(n_M)}, \ell^{(M)}\left(\lambda^{(n_M)}\right)\right) \Big\},$$

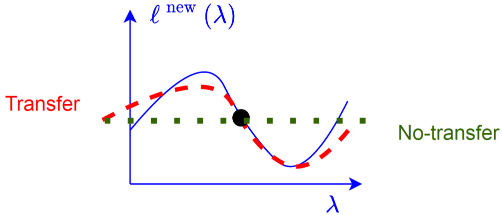
for a set of datasets $\{D_1, \ldots D_M\}$ with respective response functions $\left\{\ell^{(1)}, \ldots \ell^{(M)}\right\}$.

Objective: Use $\mathcal{H}$ to find good λ for a new $\ell^{(M+1)}$ faster.

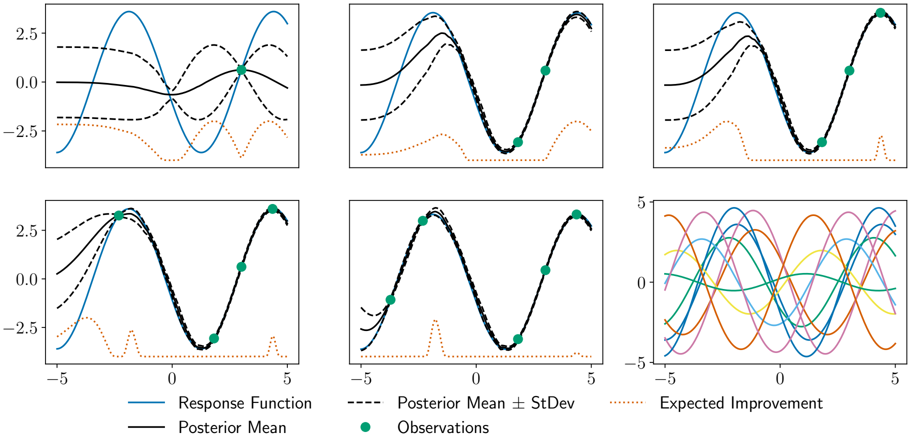
Conceptual Illustration: Meta-Learned Surrogates



Meta-learning a surrogate on source tasks (above) helps HPO on a target task (below):



Transfer Learning: Fewer HPO Trials



Martin Wistuba and Josif Grabocka. "Few-Shot Bayesian Optimization with Deep Kernel Surrogates". In: *ICLR*. OpenReview.net, 2021

Outline

Outline

Evaluation Metrics (1/3)

- Evaluation is done on multiple benchmarks, each having many different tasks.
- Presenting results per task is infeasible, aggregation of results is required.
- Aggregation is non-trivial.
  - Do you account only for performance after a fixed budget or do you also consider the speed of progress?
  - How do you define a fixed budget for datasets of with different sizes?
  - How do you account for different scales in different datasets?

Evaluation Metrics (2/3)

- Average Regret

$$\frac{1}{M}\sum_{i=1}^{M} \ell^{(i)}(\lambda_{\text{best}}^{(i)}) - \ell_{\text{min}}^{(i)}$$

Evaluation Metrics (2/3)

- Average Regret

$$\frac{1}{M}\sum_{i=1}^{M}\ell^{(i)}(\lambda_{\text{best}}^{(i)})-\ell_{\text{min}}^{(i)}$$

- Normalized Average Regret

$$\frac{1}{M}\sum_{i=1}^{M}\frac{\ell^{(i)}\left(\lambda_{\text{best}}^{(i)}\right)-\ell_{\text{min}}^{(i)}}{\ell_{\text{max}}^{(i)}-\ell_{\text{min}}^{(i)}}$$

Evaluation Metrics (3/3)

- Average Rank

$$\frac{1}{M}\sum_{i=1}^{M}\text{rank}^{(i)}\left(\lambda_{\text{best}}^{(i)}\right)$$

where $\text{rank}^{(i)}\left(\lambda_{\text{best}}^{(i)}\right)$ is the rank obtained by the method compared to other methods.

Evaluation Metrics (3/3)

- Average Rank

$$\frac{1}{M}\sum_{i=1}^{M}\text{rank}^{(i)}\left(\lambda^{(i)}_{\text{best}}\right)$$

where $\text{rank}^{(i)}\left(\lambda^{(i)}_{\text{best}}\right)$ is the rank obtained by the method compared to other methods.

- Area under any of the previous metric curves.
  - All previous metrics can only be reported for a given budget.
  - The sum of a metric at every given budget will yield a single value.

Evaluation Metrics - Example

Dataset	**Opt1**	**Opt2**	**Opt3**	$\ell_{\text{min}}^{(t)}$	$\ell_{\text{max}}^{(t)}$
Dataset 1	70%	**65%**	80%	60%	80%
Dataset 2	60%	59%	**58%**	20%	65%
Dataset 3	98%	99%	**97.9%**	97%	99%

Evaluation Metrics - Example

Dataset	**Opt1**	**Opt2**	**Opt3**	$\ell_{\text{min}}^{(t)}$	$\ell_{\text{max}}^{(t)}$
Dataset 1	70%	**65%**	80%	60%	80%
Dataset 2	60%	59%	**58%**	20%	65%
Dataset 3	98%	99%	**97.9%**	97%	99%
Average Regret	17%	**15.3%**	19.6%		

Evaluation Metrics - Example

Dataset	Opt1	Opt2	Opt3	$\ell_{\text{min}}^{(t)}$	$\ell_{\text{max}}^{(t)}$
Dataset 1	70%	**65%**	80%	60%	80%
Dataset 2	60%	59%	**58%**	20%	65%
Dataset 3	98%	99%	**97.9%**	97%	99%
Average Regret	17%	**15.3%**	19.6%		
Normalized Average Regret	**62.9%**	70.6%	76.5%		

Evaluation Metrics - Example

Dataset	**Opt1**	**Opt2**	**Opt3**	$\ell_{\text{min}}^{(t)}$	$\ell_{\text{max}}^{(t)}$
Dataset 1	70%	**65%**	80%	60%	80%
Dataset 2	60%	59%	**58%**	20%	65%
Dataset 3	98%	99%	**97.9%**	97%	99%
Average Regret	17%	**15.3%**	19.6%		
Normalized Average Regret	**62.9%**	70.6%	76.5%		
Average Rank	2.3	2.0	**1.7**		

Evaluation Metrics - Summary

- What is the right evaluation metric?
  - Every single one has their own problems.
  - Evaluate with respect to all.
    - If they all agree on a best method, that's probably the best one.
    - If they disagree, results are inconclusive.
  - Unclear: Performance differences at different optimization budgets.

Benchmarks Overview

Benchmark	#Evals	#Datasets	#HPs	#Fidelities	Comments
LCBench [19]	70K	35	7	50	MLP - architecture and HPs
WEKA [14]	1.3M	59	1-7	1	several search spaces
HPO-B v1 [11]	6.4M	196	1-53	1	OpenML benchmark
HPO-B v2 [11]	6.3M	101	2-18	1	for cross-search space HPO
HPOBench [3]	50K	1-20	2-26	varies	trees or epochs as fidelity
TaskSet [10]	29M	1162	1-10	varies	optimizer HPs for different NNs

Outline

Meta-Features

- Meta-features are describing properties of the dataset.
  - Desired property: iff the meta-features between two datasets are similar, the best hyperparameter configurations are similar.

We categorize them according to how they are generated:

- **Feature Engineering:** All meta-features that are created based on human-defined operations.
  - Classical approach
  - Big variety of meta-features proposed in the literature
- **Feature Learning:** Meta-features are learned directly from the data.

Meta-Features - Engineering

- **Simple:** Such as dataset size, number of features, etc.
- **Statistical:** Such as kurtosis, skewness, etc.
- **Information Theoretic:** Such as normalized class entropy, mutual information, etc.
- **Model-Based:** Features are extracted from a simple model trained on the data
  - Examples: number of leaves in a decision tree trained without pruning
- **Landmarking:** Performance metrics of simple learners. (e.g. Naive Bayes accuracy)

Matthias Reif et al. "Automatic classifier selection for non-experts". In: *Pattern Anal. Appl.* 17.1 (2014), pp. 83–96

Meta-Features - Learning

In meta-feature learning, a function $\phi : \mathcal{D} \to \mathbb{R}^k$ is learned, which extracts k-dimensional meta-features from a given dataset $D \in \mathcal{D}$.

The function φ is parameterized and its parameters are learned from datasets and their similarity scores.

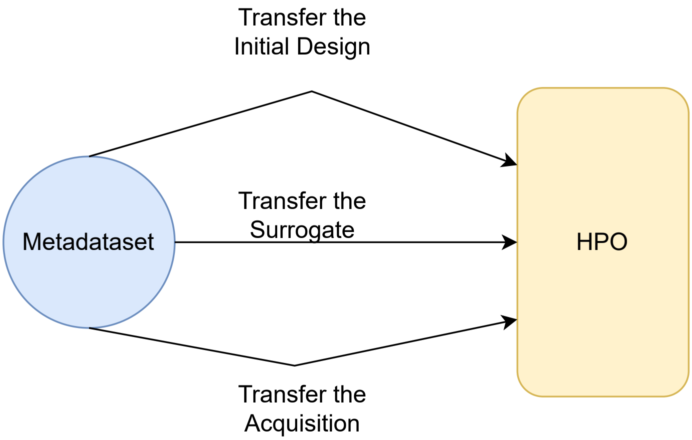
Noteworthy meta-feature learning strategies:

- Tabular Data: Dataset2Vec [7].
- Image Data: Set transformer [9].

.

Outline

Transfer modalities for HPO with Bayesian Optimization



Outline

Transfer by Initialization

Basic idea: Start with configurations that did well in the past. Then, continue with an arbitrary HPO technique.

Advantages

- Works very well in practice.
- Transparent method, easy to understand.
- Works with most HPO methods.
- Typically easy to implement.
- No additional overhead introduced.
- Can be shared and updated easily.

Disadvantages

- Initialization length might be a critical hyperparameter.
- Does not adapt, might struggle with negative transfer.

Formal Problem Definition

A hyperparameter optimization initialization is a sequence of hyperparameter configurations $\mathcal{I} = (\lambda_1 \ldots \lambda_n)$ which minimizes

$$\mathcal{L}(\mathcal{D}, \mathcal{I}) = \sum_{D \in \mathcal{D}} \min\{\ell_D(\lambda_i) \mid i \in \{1 \ldots n\}\} \; .^1$$

In words: At least one configuration is a good configuration for any dataset we have seen so far.

Desired Properties

- **No redundancies:** There should be no two configurations that are very similar.
- **Coverage:** The entire search space should be covered.
- **Good performance:** The initialization should already yield good results.

[1] For simplicity, we ignore that ℓ_D may require normalization.

Nearest-Neighbor Initialization (1/2)

1. Given is a set of datasets $\{D_1, \ldots, D_M\}$ and corresponding best hyperparameter configurations $\{\lambda_1, \ldots, \lambda_M\}$.

2. Measure the similarity between each dataset and the new dataset D_{new} with some distance function d.

3. Select the n configurations from the best configurations corresponding to the datasets with the highest similarity.

Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. "Initializing Bayesian Hyperparameter Optimization via Meta-Learning". In: *AAAI*. AAAI Press, 2015, pp. 1128–1135

Nearest-Neighbor Initialization (2/2)

- The distance between two datasets is non-trivial to compute.
- Common choice: Euclidean distance between meta-features.

Problems

- **Possible redundancies:** Similar datasets may have similar best configurations.
- **Dataset similarity:** With wrong similarities, we may use a bad initialization.

Greedy Initialization (1/2)

Greedy initialization uses a greedy selection algorithm to minimize

$$\mathcal{L}(\mathcal{D},\mathcal{I}) = \sum_{D\in\mathcal{D}} \min\{\ell_D(\lambda_i) \mid i \in \{1\dots n\}\} \ .$$

1. Create an empty list $\mathcal{I} = \emptyset$.
2. Add the element $\lambda^\star \in \Lambda$ to $\mathcal{I}$, where

$$\lambda^\star = \arg\min_{\lambda\in\Lambda} \mathcal{L}(\mathcal{D},\mathcal{I}\cup\{\lambda\}) \ ,$$

until $|\mathcal{I}| = I$

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. "Sequential Model-Free Hyperparameter Tuning". In: *ICDM*. IEEE Computer Society, 2015, pp. 1033–1038

Greedy Initialization (2/2)

- In the optimal case, a set of configurations is evaluated on all datasets.
- If this is not possible, use surrogate models $\hat{\ell}_D$ to impute the missing values.

Advantages

- **Low redundancies:** Configurations are chosen to complement each other.
- **Robust:** If a new set is similar to any previously dataset, the initialization sequence contains at least one good configuration.

Disadvantages

- Greedy selection is an approximation.
- Can depend on quality of surrogates.

Initialization with Evolutionary Algorithms

- Alternative to the greedy selection that will find solutions closer to the optimum.
- Same advantages and disadvantages.

1. Initialize $\mathcal{I}$ with configurations that performed best on some random datasets.
2. Update $\mathcal{I}$ with an evolutionary algorithm.

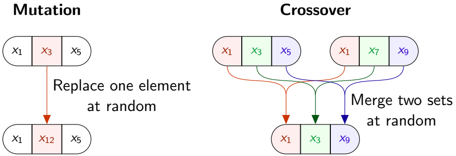


Figure 1: Examples for the mutation and crossover operation with $I = 3$.

Martin Wistuba and Josif Grabocka. "Few-Shot Bayesian Optimization with Deep Kernel Surrogates". In: *ICLR*. OpenReview.net, 2021

Initialization Learning (1/3)

In initialization learning our problem is solved via gradient-based methods.

$$\mathcal{L}\left(\mathcal{D},\mathcal{I}\right)=\sum_{D\in\mathcal{D}}\min\{\ell_D(\lambda_i)\mid i\in\{1\ldots n\}\}\;.$$

Problem: This loss is not differentiable because

1. the minimum function is not differentiable and
2. ℓ_D is only partially observed and the computation for arbitrary λ is expensive.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. "Learning hyperparameter optimization initializations". In: *DSAA*. IEEE, 2015, pp. 1–10

Differentiable Meta-Loss

Problem 1: Minimum function is not differentiable.

- Replace it with the soft-minimum function.

$$\min\{\lambda_1, \ldots, \lambda_n\} \approx \sum_{i=1}^{n} \lambda_i \sigma(\lambda)_i$$

where

$$\sigma\left((\lambda_1, \ldots, \lambda_n)^T\right)_i = \frac{e^{\beta\lambda_i}}{\sum_{j=1}^{n} e^{\beta\lambda_j}}$$

Differentiable Meta-Loss

Problem 2: ℓ_D is only partially observed and the computation for arbitrary λ is expensive.

- Replace ℓ_D with a differentiable surrogate models $\hat{\ell}_D$ that is trained on all available observations on D.

Thus, the final, differentiable meta-loss is

$$\mathcal{L}\left(\mathcal{D},\mathcal{I}\right)=\frac{1}{|\mathcal{D}|}\sum_{D\in\mathcal{D}}\sum_{i=1}^{n}\sigma_{D,i}\hat{\ell}_D\left(\lambda_i\right)$$

Initialization Learning Algorithm

1. Initialize $\mathcal{I}$ with configurations that performed best on some random datasets.

2. Update $\mathcal{I}$ with gradient descent

$$\frac{\partial}{\partial \lambda_{I,j}} \mathcal{L}\left(\mathcal{D}, \mathcal{I}\right) = \frac{1}{|\mathcal{D}|} \sum_{D \in \mathcal{D}} \sigma_{D,I} \cdot \left(\frac{\partial}{\partial \lambda_{I,j}} \hat{\ell}_D\left(\lambda_I\right) \right) \cdot \left(\beta\left(1 - \sigma_{D,I}\right) \hat{\ell}_D\left(\lambda_I\right) + 1 \right)$$

Outline

Transfer by Surrogates

Basic idea: Meta-learn a probabilistic surrogate from the evaluations of a meta-dataset

Advantages

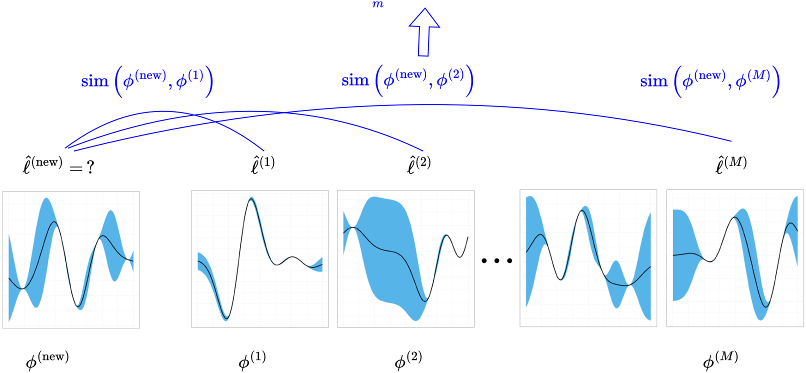
- Leads to state-of-the-art results
- Easily accommodates meta-features as auxiliary surrogate features
- Can be extended to zero-shot HPO without an initial design

Disadvantages

- Requires implementing and running a meta-learning procedure for the surrogate parameters
- Requires a careful selection of hyper-hyper-parameters for the meta-learning procedure

Two-stage Surrogate Transfer

$$\hat{\ell}^{(\text{new})} \propto \sum_m \text{sim}\left(\phi^{(\text{new})}, \phi^{(m)}\right) \hat{\ell}^{(m)}$$



Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. "Two-Stage Transfer Surrogate Model For Automatic Hyperparameter Optimization". In: *European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 9851*. ECML PKDD 2016. Riva del Garda, Italy: Springer-Verlag, 2016,

Few-Shot Bayesian Optimization (FSBO)

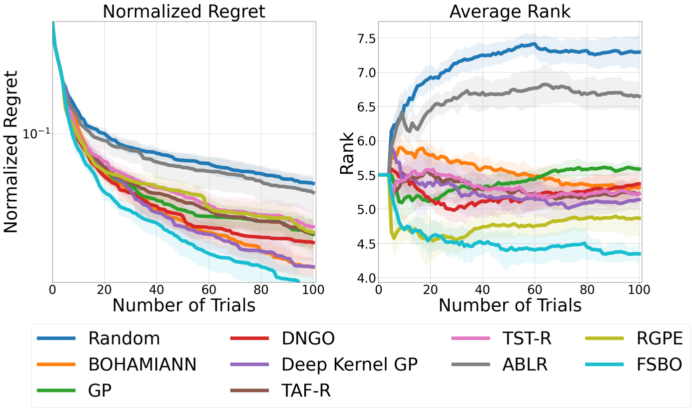
- Given n hyperparameter configs evaluated on K datasets
- Meta-dataset of evaluations $M := \bigcup_{m=1}^{M} \left\{\lambda_i, \ell^{(m)}(\lambda_i)\right\}_{i=1}^{n}$
- Meta-learn a parametric probabilistic surrogate $\ell(\lambda) \approx \hat{\ell}(\lambda; \psi) + \epsilon$ to approximate:

$$\psi^* := \arg\max_{\psi} \sum_{m=1}^{M} \sum_{i=1}^{n} \log p\left(\ell^{(m)}(\lambda_i) \mid \lambda_i, \psi\right)$$

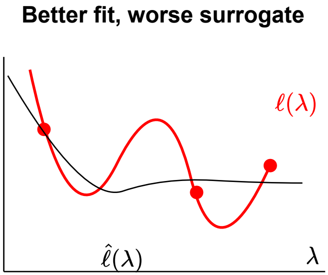
- On a new task: Initialize Bayesian optimization with the meta-learned surrogate $\hat{\ell}(\lambda; \psi^*)$

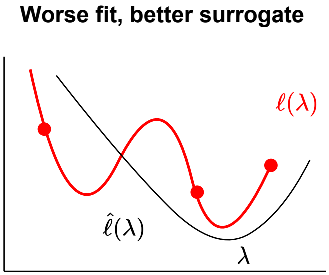
Martin Wistuba and Josif Grabocka. "Few-Shot Bayesian Optimization with Deep Kernel Surrogates". In: *ICLR*. OpenReview.net, 2021

FSBO - Performance



Novel Paradigm: Surrogate fitting as Learning-to-rank





- A good surrogate when $\left[\arg\min_\lambda \hat{\ell}(\lambda) \approx \arg\min_\lambda \ell(\lambda)\right]$
- Rank preservation is more important than fitness

Deep Ranking Ensembles (DRE)

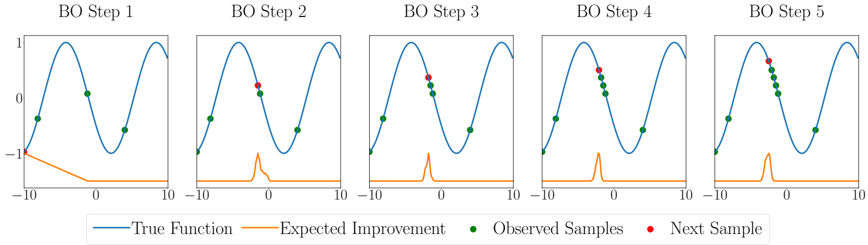
- In HPO only the best hyperparameter configuration matters.
- Ground-truth rank $\pi(i) = \sum_{k=1}^{n} \mathbb{1}_{\ell(\lambda_k) \leq \ell(\lambda_i)}$
- We learn a ranker $r : \Lambda \to \mathbb{R}$ to approximate the true ranks.
- Optimize the ranker via a list-wise learning-to-rank loss:

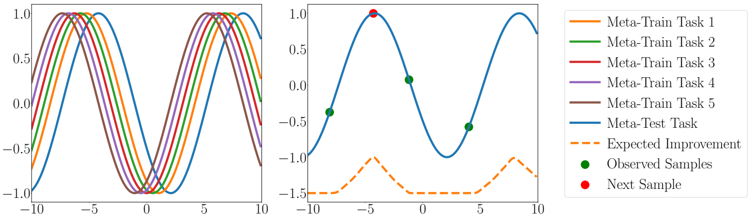
$$\arg\min_{\psi} \sum_{i=1}^{n} w\left(\pi(i)\right) \frac{e^{r\left(\lambda_{\pi(i)};\psi\right)}}{\sum_{j=k}^{n} e^{r\left(\lambda_{\pi(j)};\psi\right)}}, \quad w\left(\pi(i)\right) = \frac{1}{\log\left(\pi(i)+1\right)}$$

- Create a probabilistic ranker via ensembling
- Meta-learn the ranking ensemble from a meta-dataset

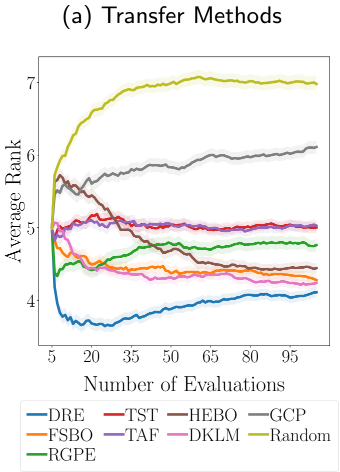
Abdus Salam Khazi, Sebastian Pineda Arango, and Josif Grabocka. "Deep Ranking Ensembles for Hyperparameter Optimization". In: *The Eleventh International Conference on Learning Representations*. 2023

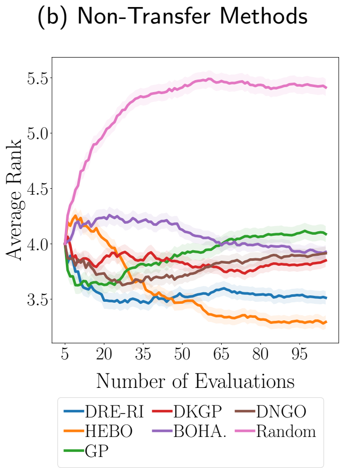
Ranking Ensembles - Impact of Transfer Learning





Ranking Ensembles - Performance





Outline

Transfer by Acquisition Function

Definition

An acquisition function is a mapping $a(\lambda, \mu(\lambda), \sigma(\lambda), \ell^{\text{best}})$ that measures the expected utility of a configuration λ.

We discuss

- Transfer with true acquisition functions
  - Acquisition functions according to above definition but which use transfer learning.
- Transfer with policies
  - Policies allow for sampling candidates. They do not evaluate their utility.

Transfer Acquisition Function

TAF is an acquisition function that combines EI with the predicted performance on other datasets.

$$a(\lambda) = \frac{w_{M+1}\mathrm{E}[\mathrm{I}_{M+1}(\lambda)] + \sum_{i=1}^{M} w_i \mathrm{I}_i(\lambda)}{\sum_{i=1}^{M+1} w_i}$$

with

$$\mathrm{I}_i(\lambda) = \max\left\{\ell_{\text{min}}^{(i)} - \hat{\ell}^{(i)}(\lambda), 0\right\}$$

- $\ell_{\text{min}}^{(i)}$ - Best value observed on D_i.
- $\hat{\ell}^{(i)}$ - Surrogate for D_i.
- w_i chosen as in TST.

Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. "Scalable Gaussian process-based transfer surrogates for hyperparameter optimization". In: *Mach. Learn.* 107.1 (2018), pp. 43–78

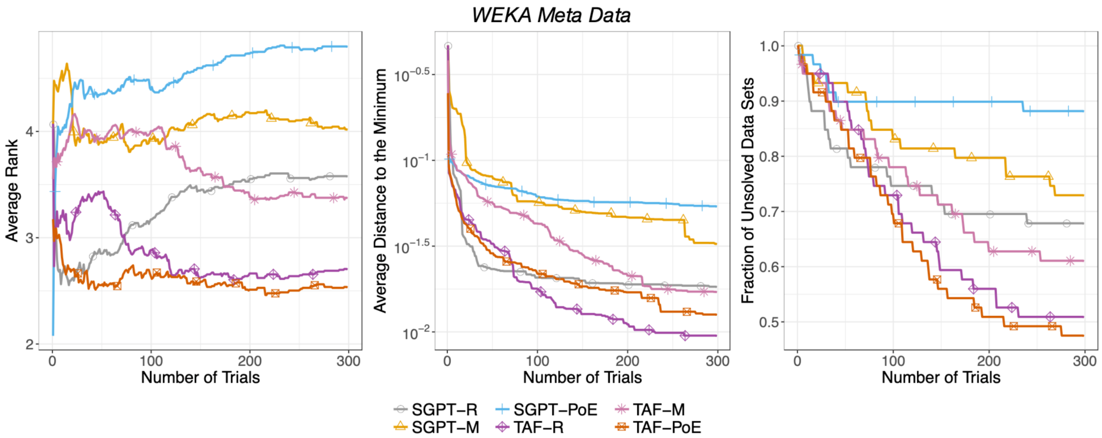
Transfer Acquisition Function - Effects

$$a(\lambda) = \frac{w_{M+1}\mathrm{E}[\mathrm{I}_{M+1}(\lambda)] + \sum_{i=1}^{M} w_i \mathrm{I}_i(\lambda)}{\sum_{i=1}^{M+1} w_i}$$

Effects

- Different scales between datasets are no longer a problem.
- Diminishing effect of other datasets over time. Avoids problems with negative transfer.
- Early phase: High uncertainty on $\ell^{(M+1)}$, search mostly guided by other datasets.
- Late phase: No further improvements on other datasets, converges to EI.

TAF - Empirical Results



Few-Shot Acquisition Function

FSAF combines meta-learning and Deep Q-Learning to learn an acquisition function:

- State representation: $(\mu(\lambda), \sigma(\lambda), \ell^{\text{best}}, t/T)$
- Tackle overfitting via Bayesian DQL:

$$\min_{q(\theta)} \left\{ \mathbb{E}_{\theta \sim q(\theta)} \left[C(\theta) \right] + \alpha D_{\text{KL}}(q \| q_0) \right\}$$

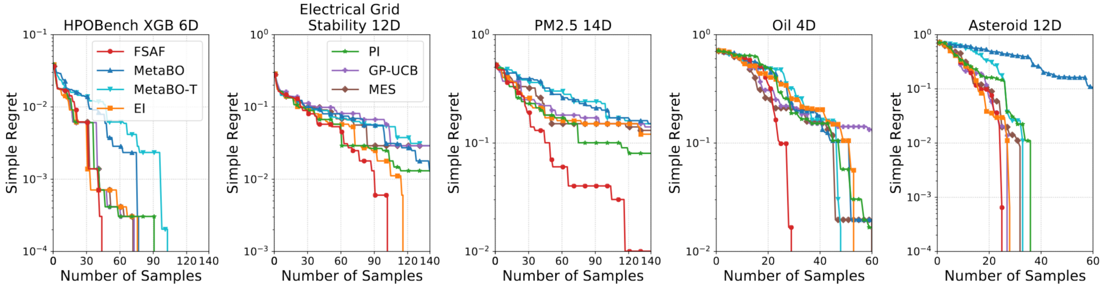
- Use a demo policy (EI) as prior

$$q_0(\theta) \propto \exp(\delta(\pi_\theta, \pi_D))$$

- Bayesian MAML loss as meta-loss

Bing-Jing Hsieh, Ping-Chun Hsieh, and Xi Liu. "Reinforced Few-Shot Acquisition Function Learning for Bayesian Optimization". In: *NeurIPS*. 2021, pp. 7718–7731

FSAF - Empirical Results

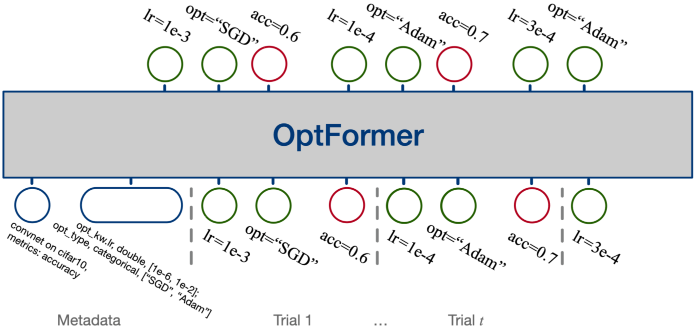


One task is used for few-shot adaptation, remaining serve for testing purposes.

OptFormer

OptFormer uses a transformer architecture to learn across search spaces, and is both a surrogate model and acquisition function at the same time.

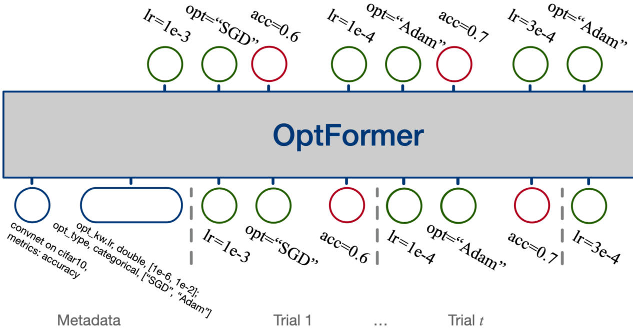
- Metadata: all information related to the task such as objective, search space, algorithm.
- At inference time: next token prediction.



Yutian Chen et al. "Towards Learning Universal Hyperparameter Optimizers with Transformers". In: *NeurIPS*. 2022

OptFormer - Training

- Optformer learns from data that was created by other optimization policies π_i.
- Objective: Learn a policy π_{prior} that simply clones the behavior of other optimizers.
- Auxiliary task: Predict the hyperparameter response.

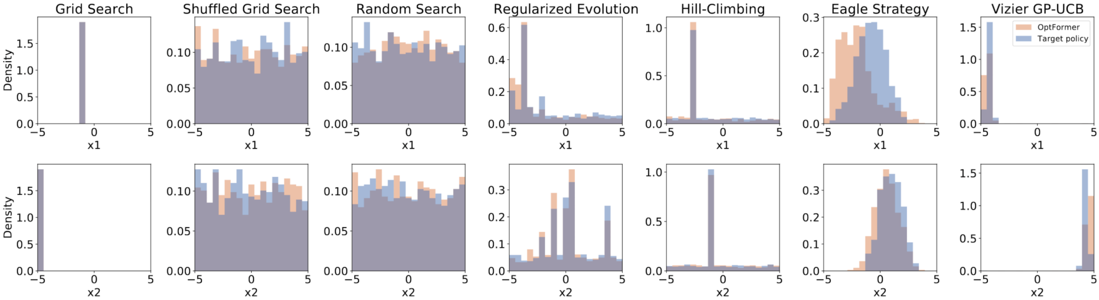


OptFormer - Beyond Imitation

- OptFormer: Prior policy
  - Sample from the prior policy π_{prior}.

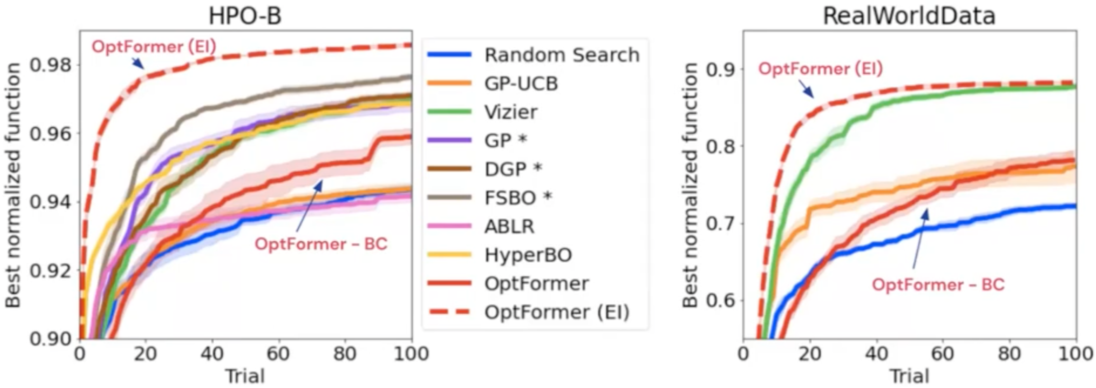
- OptFormer + Acquisition Function
  - Sample multiple candidates from the prior policy $\lambda^{(i)} \sim \pi_{\text{prior}}$.
  - Predict their performance with the OptFormer surrogate model.
  - Evaluate the candidate with highest measured utility according to EI.

OptFormer - Imitation Performance



Changing the algorithm in the metadata allows to imitate different optimizers.

OptFormer - Results



π_{prior} allows for successfully pruning the candidate space.

Outline

Conclusion

- Introduction to (Transfer) HPO
  - Flavors of HPO (black-box, gray-box, white-box)
  - Motivation for transferring knowledge in HPO
  - Meta-Features
  - Evaluation Metrics
- Overview over Transfer Methods
  - Initialization
  - Surrogate Models
  - Acquisition Functions

Thank you for your attention.

Questions? Comments?

References I

[1] Atilim Gunes Baydin et al. "Online Learning Rate Adaptation with Hypergradient Descent". In: *International Conference on Learning Representations*. 2018.

[2] Yutian Chen et al. "Towards Learning Universal Hyperparameter Optimizers with Transformers". In: *NeurIPS*. 2022.

[3] Katharina Eggensperger et al. "HPOBench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for HPO". In: *NeurIPS Datasets and Benchmarks*. 2021.

[4] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. "Initializing Bayesian Hyperparameter Optimization via Meta-Learning". In: *AAAI*. AAAI Press, 2015, pp. 1128–1135.

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *ICML*. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1126–1135.

References II

[6] Bing-Jing Hsieh, Ping-Chun Hsieh, and Xi Liu. "Reinforced Few-Shot Acquisition Function Learning for Bayesian Optimization". In: *NeurIPS*. 2021, pp. 7718–7731.

[7] Hadi S. Jomaa, Lars Schmidt-Thieme, and Josif Grabocka. "Dataset2Vec: learning dataset meta-features". In: *Data Min. Knowl. Discov.* 35.3 (2021), pp. 964–985.

[8] Abdus Salam Khazi, Sebastian Pineda Arango, and Josif Grabocka. "Deep Ranking Ensembles for Hyperparameter Optimization". In: *The Eleventh International Conference on Learning Representations*. 2023.

[9] Juho Lee et al. "Set transformer: A framework for attention-based permutation-invariant neural networks". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3744–3753.

[10] Luke Metz et al. "Using a thousand optimization tasks to learn hyperparameter search strategies". In: *CoRR* abs/2002.11887 (2020).

References III

[11] Sebastian Pineda-Arango et al. "HPO-B: A Large-Scale Reproducible Benchmark for Black-Box HPO based on OpenML". In: *NeurIPS Datasets and Benchmarks*. 2021.

[12] Matthias Reif et al. "Automatic classifier selection for non-experts". In: *Pattern Anal. Appl.* 17.1 (2014), pp. 83–96.

[13] Martin Wistuba and Josif Grabocka. "Few-Shot Bayesian Optimization with Deep Kernel Surrogates". In: *ICLR*. OpenReview.net, 2021.

[14] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. "Hyperparameter Search Space Pruning - A New Component for Sequential Model-Based Hyperparameter Optimization". In: *ECML/PKDD (2)*. Vol. 9285. Lecture Notes in Computer Science. Springer, 2015, pp. 104–119.

[15] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. "Learning hyperparameter optimization initializations". In: *DSAA*. IEEE, 2015, pp. 1–10.

References IV

[16] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. "Scalable Gaussian process-based transfer surrogates for hyperparameter optimization". In: *Mach. Learn.* 107.1 (2018), pp. 43–78.

[17] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. "Sequential Model-Free Hyperparameter Tuning". In: *ICDM*. IEEE Computer Society, 2015, pp. 1033–1038.

[18] Martin Wistuba, Nicolas Schilling, and Lars Schmidt-Thieme. "Two-Stage Transfer Surrogate Model For Automatic Hyperparameter Optimization". In: *European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 9851*. ECML PKDD 2016. Riva del Garda, Italy: Springer-Verlag, 2016, pp. 199–214.

[19] Lucas Zimmer, Marius Lindauer, and Frank Hutter. "Auto-Pytorch: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 43.9 (2021), pp. 3079–3090.