

AutoML: Introduction

The Big Picture

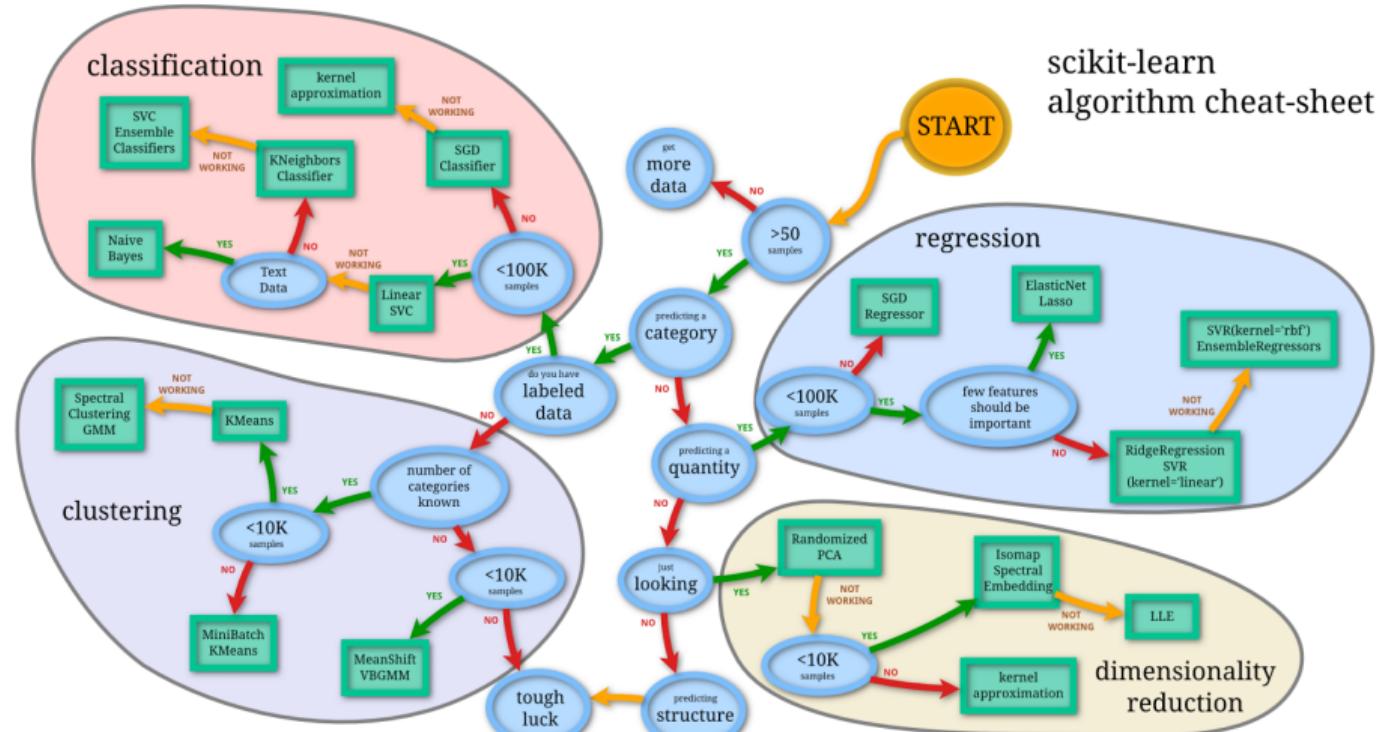
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Machine Learning

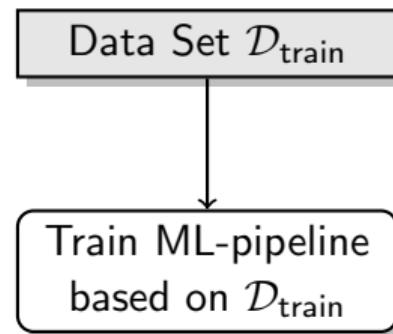
“Machine learning is the science of getting computers to act without being explicitly programmed.”

by Andrew Ng

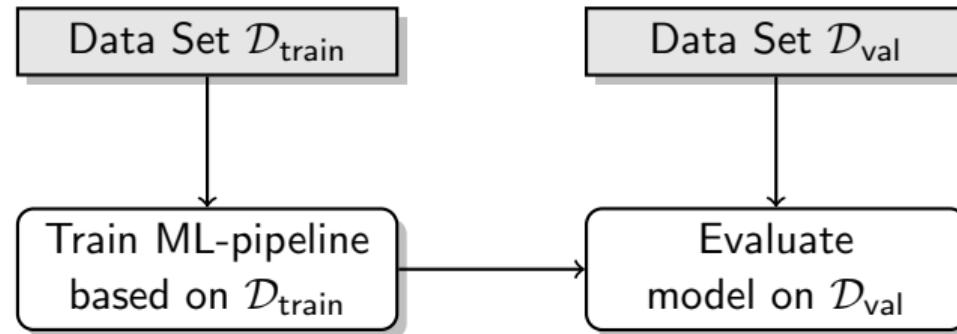
Machine Learning requires many design decisions



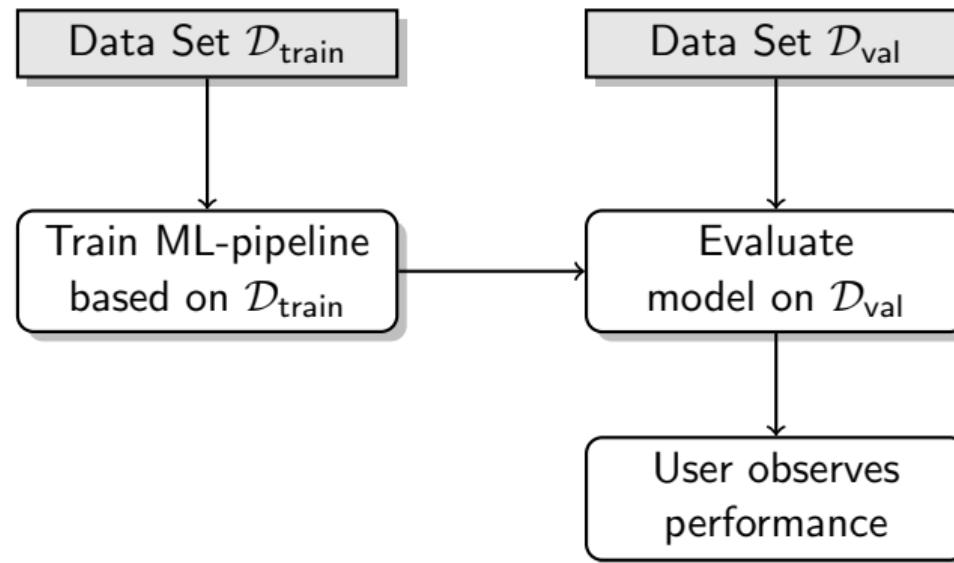
Machine Learning Workflow



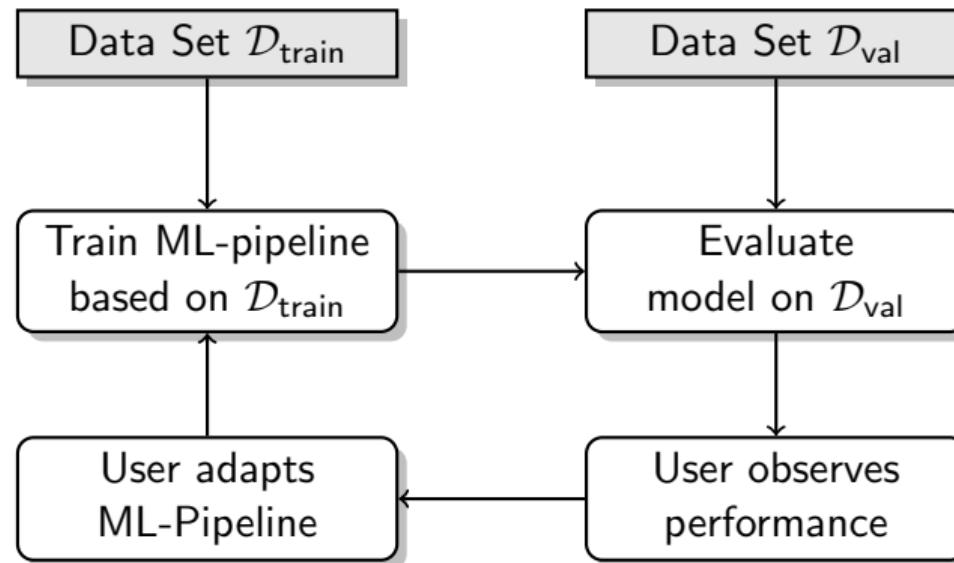
Machine Learning Workflow



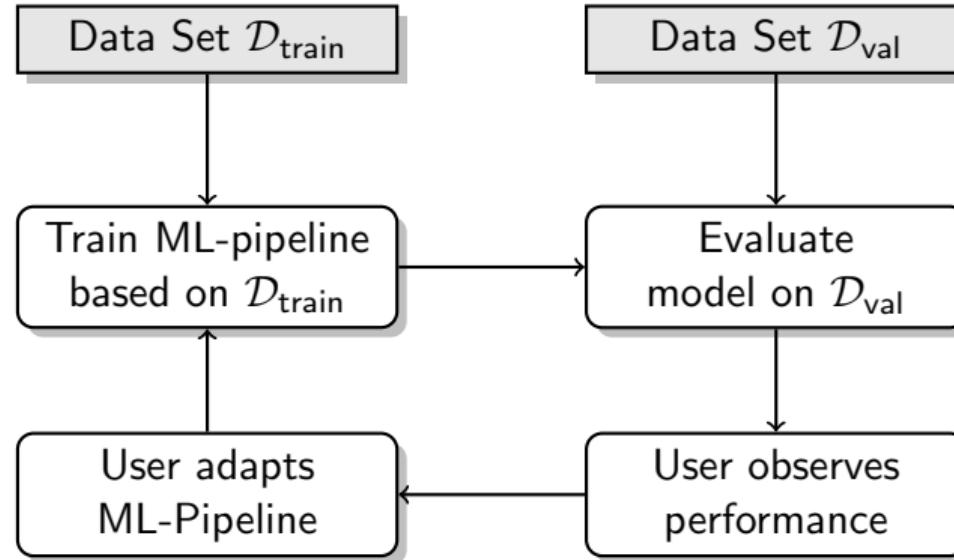
Machine Learning Workflow



Machine Learning Workflow



Machine Learning Workflow



~ Users indirectly teach machines how to learn.

Machine Learning does not scale up

- Basics in machine learning are not hard to grasp

Machine Learning does not scale up

- Basics in machine learning are not hard to grasp
- Achieving state-of-the-art performance is quite hard

Machine Learning does not scale up

- Basics in machine learning are not hard to grasp
- Achieving state-of-the-art performance is quite hard
- Design decisions are often not intuitive and require a lot of expertise
 - ▶ making these design decisions is a tedious and error-prone task

Machine Learning does not scale up

- Basics in machine learning are not hard to grasp
- Achieving state-of-the-art performance is quite hard
- Design decisions are often not intuitive and require a lot of expertise
 - ▶ making these design decisions is a tedious and error-prone task
- The job market for ML-experts is nearly empty

Machine Learning does not scale up

- Basics in machine learning are not hard to grasp
- Achieving state-of-the-art performance is quite hard
- Design decisions are often not intuitive and require a lot of expertise
 - ▶ making these design decisions is a tedious and error-prone task
- The job market for ML-experts is nearly empty
- Even with experts, developing new ML-applications takes time

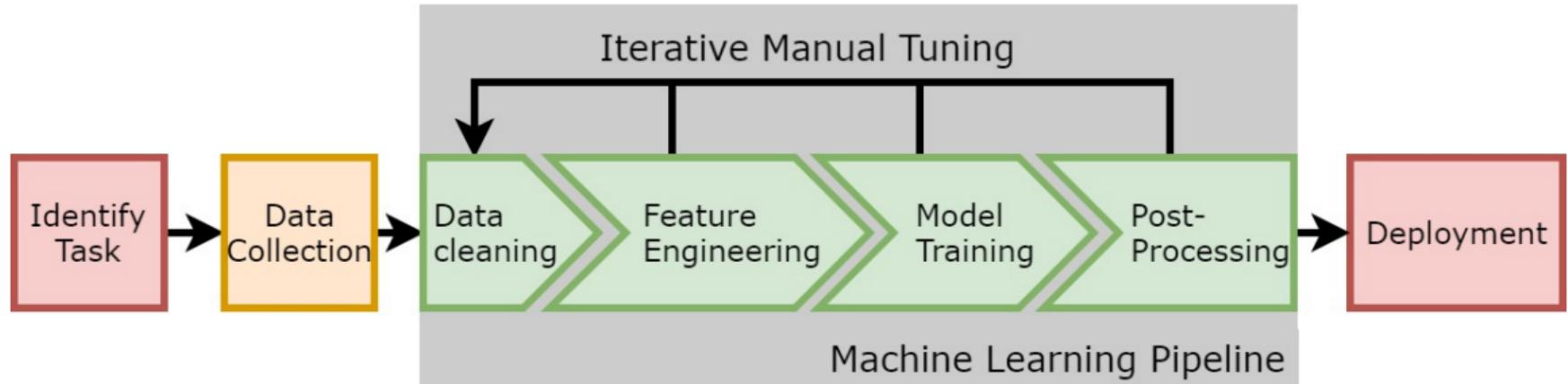
Machine Learning does not scale up

- Basics in machine learning are not hard to grasp
- Achieving state-of-the-art performance is quite hard
- Design decisions are often not intuitive and require a lot of expertise
 - ▶ making these design decisions is a tedious and error-prone task
- The job market for ML-experts is nearly empty
- Even with experts, developing new ML-applications takes time

Zoubin Ghahramani said that he often heard that:

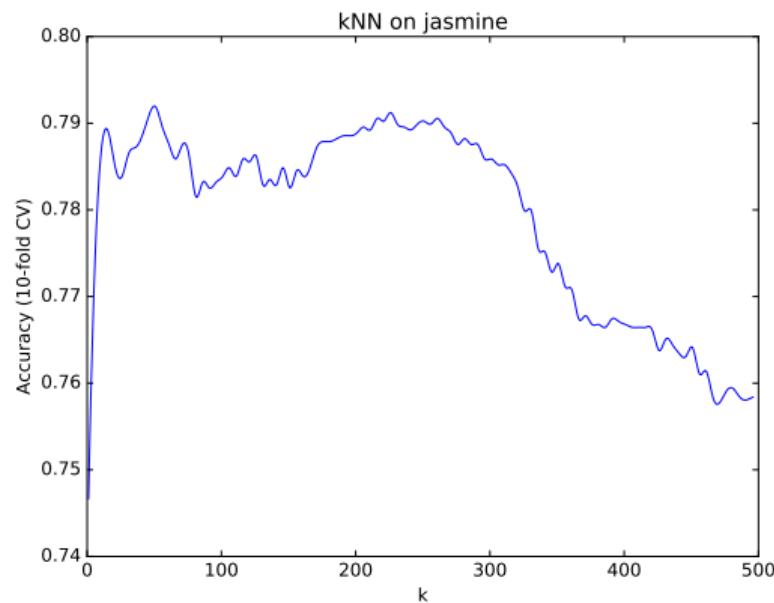
"I'd like to use machine learning, but I can't invest much time."

Why does ML development take a lot of time?



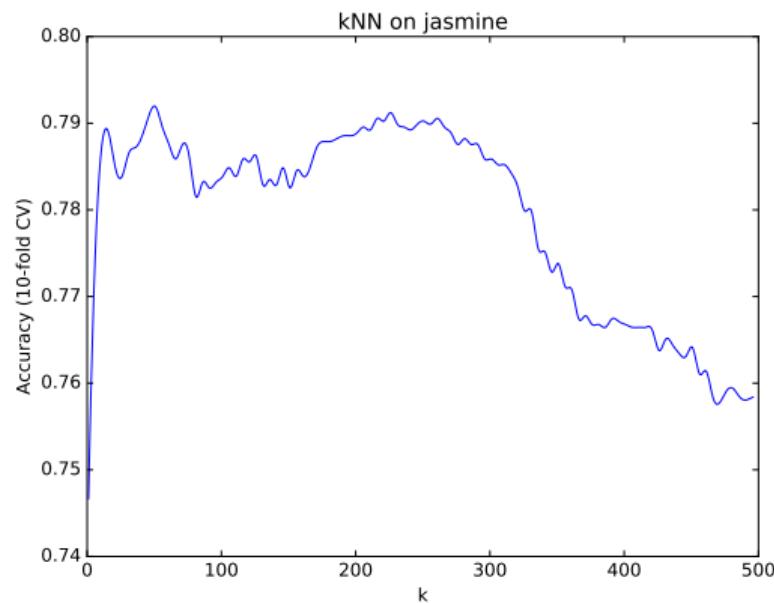
~~ To achieve state-of-the-art performance,
this manual tuning has to be done for each new dataset again.

A Simple Example with k -NN



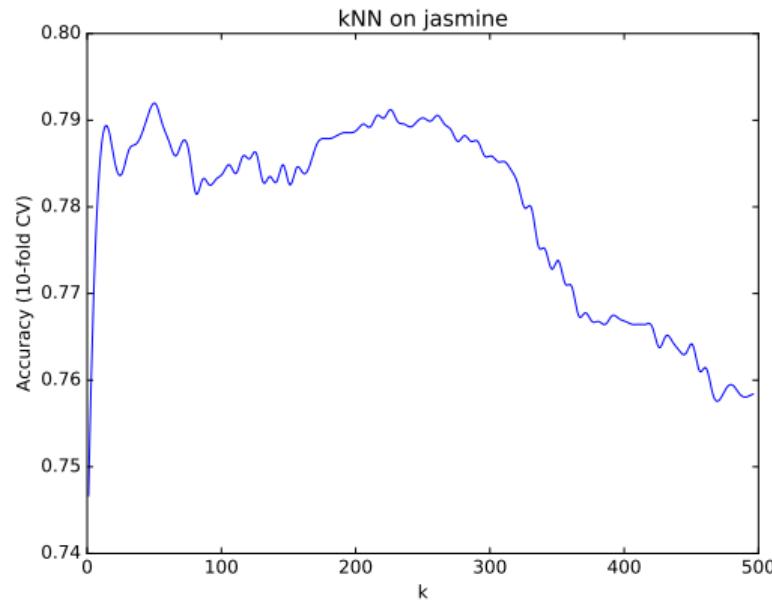
- k -nearest neighbors is one of the simplest ML algorithms

A Simple Example with k -NN



- k -nearest neighbors is one of the simplest ML algorithms
- Size of neighbourhood (k) is very important for its performance

A Simple Example with k -NN



- k -nearest neighbors is one of the simplest ML algorithms
- Size of neighbourhood (k) is very important for its performance
- The performance function depending on k is quite complex (not at all convex)

Goal of AutoML

AutoML

The goal of AutoML is to automate all parts of machine learning (as needed) to *support* users efficiently building their ML-applications.

Goal of AutoML

AutoML

The goal of AutoML is to automate all parts of machine learning (as needed) to *support* users efficiently building their ML-applications.

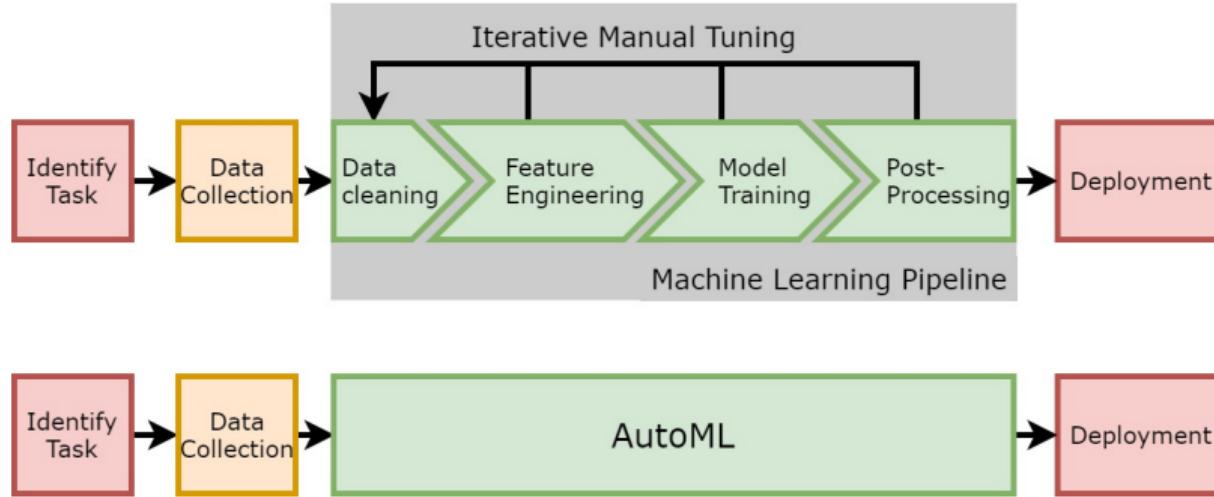
Informal Definition: AutoML System

Given

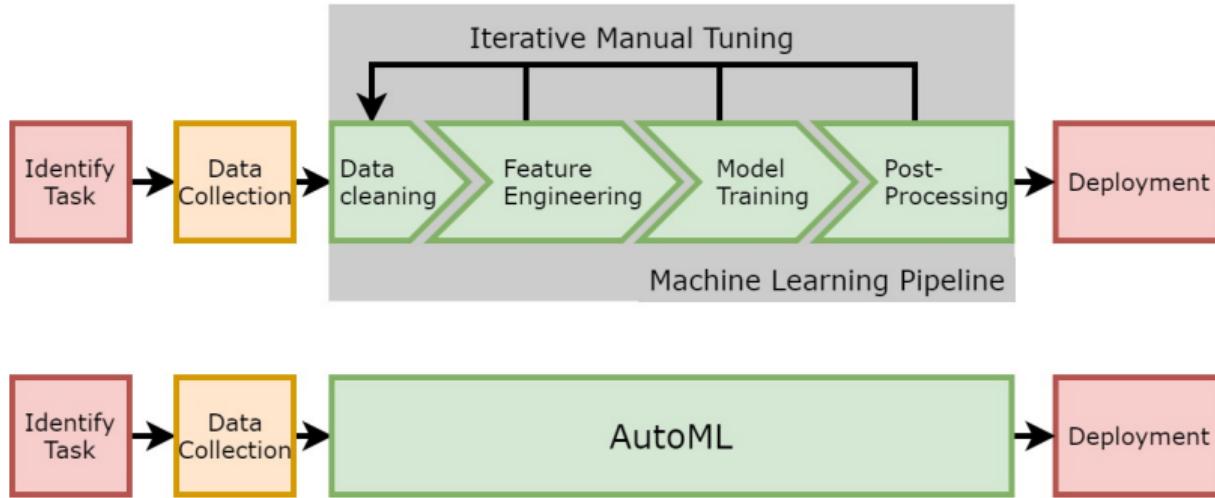
- a dataset
- a task (e.g., regression or classification)
- a cost metric (e.g., accuracy or RMSE)

an AutoML system automatically determines the approach that performs best for this particular application.

ML vs AutoML



ML vs AutoML



With AutoML, we ...

- support ML users
- improve the efficiency of developing new ML applications
- reduce the required ML-expertise
- might achieve better performance than developers w/o AutoML

AutoML in Research

AutoML enables:

- ① more efficient research
 - ▶ AutoML has shown on subproblems to outperform human experts

AutoML in Research

AutoML enables:

- ① more efficient research
 - ▶ AutoML has shown on subproblems to outperform human experts
- ② more systematic research
 - ▶ humans tend to be unsystematic which leads to errors

AutoML in Research

AutoML enables:

- ① more efficient research
 - ▶ AutoML has shown on subproblems to outperform human experts
- ② more systematic research
 - ▶ humans tend to be unsystematic which leads to errors
- ③ more reproducible research
 - ▶ human's unsystematic approaches cannot be reproduced,
but AutoML is systematic

AutoML in Research

AutoML enables:

- ① more efficient research
 - ▶ AutoML has shown on subproblems to outperform human experts
- ② more systematic research
 - ▶ humans tend to be unsystematic which leads to errors
- ③ more reproducible research
 - ▶ human's unsystematic approaches cannot be reproduced,
but AutoML is systematic
- ④ broader use of ML also in other disciplines
 - ▶ ML should not be limited to computer scientists;
 - ▶ the most amazing applications of ML are often done
by either interdisciplinary teams or even non-computer scientists

Challenges in AutoML

- ① Each dataset potentially require different optimal ML-designs
 - ▶ Design decisions have to be made for each dataset again

Challenges in AutoML

- ① Each dataset potentially require different optimal ML-designs
 - ▶ Design decisions have to be made for each dataset again
- ② Training of a single ML model can be quite expensive (e.g., hours, days or weeks)
 - ▶ often, we cannot try many design decisions

Challenges in AutoML

- ① Each dataset potentially require different optimal ML-designs
 - ▶ Design decisions have to be made for each dataset again
- ② Training of a single ML model can be quite expensive (e.g., hours, days or weeks)
 - ▶ often, we cannot try many design decisions
- ③ the mathematical relation between design decisions and performance is (often) unknown
 - ▶ gradient-based optimization is not directly possible

Challenges in AutoML

- ① Each dataset potentially require different optimal ML-designs
 - ▶ Design decisions have to be made for each dataset again
- ② Training of a single ML model can be quite expensive (e.g., hours, days or weeks)
 - ▶ often, we cannot try many design decisions
- ③ the mathematical relation between design decisions and performance is (often) unknown
 - ▶ gradient-based optimization is not directly possible
- ④ optimization in highly complex spaces
 - ▶ incl. categorical choices, continuous parameters, conditional dependencies

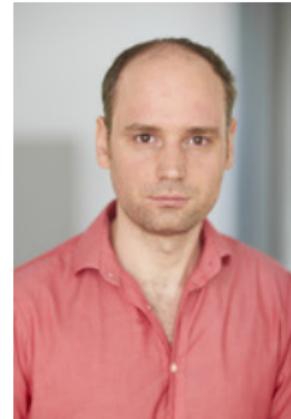
AutoML: Introduction

The Team

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Bernd Bischl

- Professor at
the Ludwig-Maximilians-University of Munich (Germany)
- Head of the statistical learning and data science group
- Co-Director of Munich Center for Machine Learning
- Founder of MLR (Machine Learning in R)
- Co-Founder of www.openml.org
- Co-Founder and advisory board member of COSEAL
(Configuration and Selection of Algorithms)



Frank Hutter

- Professor at the University of Freiburg (Germany)
- Head of the machine learning lab
- Won 1st and 2nd international AutoML challenge
- Co-author of popular AutoML tools Auto-WEKA, Auto-sklearn, Auto-PyTorch
- Co-founder and co-organizer of AutoML workshop series at ICML (since 2014)
- Co-Editor of book on *Automated Machine Learning: Methods, Systems, Challenges*
- Co-Head of automl.org



Lars Kotthoff

- Professor at the University of Wyoming (USA)
- Head of the Meta-Algorithmics, Learning and Large-Scale Empirical Testing lab (MALLET) and director of the Artificially Intelligent Manufacturing Center (AIM)
- Developer of MLR (Machine Learning in R)
- Organizer of competitions on algorithm selection
- Co-Editor of book on *Automated Machine Learning: Methods, Systems, Challenges*



Marius Lindauer

- Professor at
the Leibniz University of Hannover (Germany)
- Head of the machine learning group
- Won 1st and 2nd international AutoML challenge
- Co-author of popular AutoML tools Auto-sklearn &
Auto-PyTorch
- Co-Head of automl.org
- Co-Founder and advisory board member of COSEAL
(Configuration and Selection of Algorithms)



Janek Thomas

- Group Lead for AutoML & XAI at Fraunhofer IIS
- Phd from the Ludwig-Maximilians-University Munich (Germany)
- Developer of MLR, the amlb AutoML benchmark and autoxgboost
- Co-Head of the Munich Datageeks e.V.



Joaquin Vanschoren

- Professor at
Eindhoven University of Technology (Netherlands)
- Machine learning, Meta-Learning
- Co-Founder of www.openml.org
- Co-Editor of book on *Automated Machine Learning: Methods, Systems, Challenges*



Supported by

- Archit Bansal
- André Biedenkapp
- Omid Charrakh
- Difan Deng
- Katharina Eggensperger
- Matthias Feurer
- Maciej Janowski
- Julia Moosbauer
- Jakob Richter
- Julien Siems
- Guri Zabergja
- Arber Zela

... and many more

AutoML: Introduction

Overview of AutoML Problems

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Objectives of AutoML

Different metrics to measure the success of ML and AutoML:

- Accuracy on a validation dataset

Objectives of AutoML

Different metrics to measure the success of ML and AutoML:

- Accuracy on a validation dataset
- AUC-ROC

Objectives of AutoML

Different metrics to measure the success of ML and AutoML:

- Accuracy on a validation dataset
- AUC-ROC
- precision & recall
- ...

Objectives of AutoML

Different metrics to measure the success of ML and AutoML:

- Accuracy on a validation dataset
- AUC-ROC
- precision & recall
- ...
- Memory consumption
- Inference time
- ...

Objectives of AutoML

Different metrics to measure the success of ML and AutoML:

- Accuracy on a validation dataset
 - AUC-ROC
 - precision & recall
 - ...
 - Memory consumption
 - Inference time
 - ...
- ↝ AutoML optimizes an arbitrary cost function, denoted as c .

Objectives of AutoML

Different metrics to measure the success of ML and AutoML:

- Accuracy on a validation dataset
 - AUC-ROC
 - precision & recall
 - ...
 - Memory consumption
 - Inference time
 - ...
- ~~~ AutoML optimizes an arbitrary cost function, denoted as c .
- ~~~ $c(\cdot)$ can also return several cost metrics

Hyperparameters of an SVM



Home Installation Documentation Examples

Google Custom Search



Previous
sklearn.svm.
...
Next
sklearn.svm.
SVR
Up
API
Reference

scikit-learn v0.20.3

Other versions

Please cite us if you use
the software.

sklearn.svm.SVC
Examples using
sklearn.svm.SVC

sklearn.svm.SVC

```
class sklearn.svm.SVC (C=1.0, kernel='rbf', degree=3, gamma='auto_deprecated', coef0=0.0, shrinking=True,  
probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1,  
decision_function_shape='ovr', random_state=None)
```

[source]

C-Support Vector Classification.

The implementation is based on libsvm. The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples.

The multiclass support is handled according to a one-vs-one scheme.

For details on the precise mathematical formulation of the provided kernel functions and how gamma, coef0 and degree affect each other, see the corresponding section in the narrative documentation: [Kernel functions](#).

Read more in the [User Guide](#).

Parameters: `C : float, optional (default=1.0)`

Penalty parameter C of the error term.

`kernel : string, optional (default='rbf')`

Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape `(n_samples, n_samples)`.

`degree : int, optional (default=3)`

Degree of the polynomial kernel function ('poly'). Ignored by all other kernels.

`gamma : float, optional (default='auto')`

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

Hyperparameter Optimization

Definition

Let

- λ be the hyperparameters of an ML algorithm \mathcal{A} with domain Λ ,

Hyperparameter Optimization

Definition

Let

- λ be the hyperparameters of an ML algorithm \mathcal{A} with domain Λ ,
- \mathcal{D}_{opt} be a dataset which is split into $\mathcal{D}_{\text{train}}$ and \mathcal{D}_{val}

Hyperparameter Optimization

Definition

Let

- λ be the hyperparameters of an ML algorithm \mathcal{A} with domain Λ ,
- \mathcal{D}_{opt} be a dataset which is split into \mathcal{D}_{train} and \mathcal{D}_{val}
- $c(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of \mathcal{A}_λ trained on \mathcal{D}_{train} and evaluated on \mathcal{D}_{val} .

Hyperparameter Optimization

Definition

Let

- λ be the hyperparameters of an ML algorithm \mathcal{A} with domain Λ ,
- \mathcal{D}_{opt} be a dataset which is split into \mathcal{D}_{train} and \mathcal{D}_{val}
- $c(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of \mathcal{A}_λ trained on \mathcal{D}_{train} and evaluated on \mathcal{D}_{val} .

The *hyper-parameter optimization (HPO)* problem is to find a hyper-parameter configuration that minimizes this cost:

$$\lambda^* \in \arg \min_{\lambda \in \Lambda} c(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Hyperparameter Optimization

Definition

Let

- λ be the hyperparameters of an ML algorithm \mathcal{A} with domain Λ ,
- \mathcal{D}_{opt} be a dataset which is split into \mathcal{D}_{train} and \mathcal{D}_{val}
- $c(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of \mathcal{A}_λ trained on \mathcal{D}_{train} and evaluated on \mathcal{D}_{val} .

The *hyper-parameter optimization (HPO)* problem is to find a hyper-parameter configuration that minimizes this cost:

$$\lambda^* \in \arg \min_{\lambda \in \Lambda} c(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Remarks:

- $\arg \min$ returns a set of optimal points of a given function. It suffices to find one element of this set and thus we use \in instead of $=$.

Hyperparameter Optimization

Definition

Let

- λ be the hyperparameters of an ML algorithm \mathcal{A} with domain Λ ,
- \mathcal{D}_{opt} be a dataset which is split into \mathcal{D}_{train} and \mathcal{D}_{val}
- $c(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of \mathcal{A}_λ trained on \mathcal{D}_{train} and evaluated on \mathcal{D}_{val} .

The *hyper-parameter optimization (HPO)* problem is to find a hyper-parameter configuration that minimizes this cost:

$$\lambda^* \in \arg \min_{\lambda \in \Lambda} c(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Remarks:

- $\arg \min$ returns a set of optimal points of a given function. It suffices to find one element of this set and thus we use \in instead of $=$.
- Sometimes, we want to optimize for different metrics, instead of one
 - ↪ multi-objective optimization and Pareto fronts

Choosing an Algorithm

- Over the years, many ML-algorithms were proposed

Choosing an Algorithm

- Over the years, many ML-algorithms were proposed
- Most of these still have a reason for existence

Choosing an Algorithm

- Over the years, many ML-algorithms were proposed
- Most of these still have a reason for existence
- Examples for classification:
 - ▶ logistic regression
 - ▶ k-nearest neighbor
 - ▶ naïve Bayes
 - ▶ support vector machine
 - ▶ decision tree
 - ▶ random forest
 - ▶ gradient boosting
 - ▶ multi-layer perceptron
 - ▶ residual networks
 - ▶ ...

Choosing an Algorithm

- Over the years, many ML-algorithms were proposed
- Most of these still have a reason for existence
- Examples for classification:
 - ▶ logistic regression
 - ▶ k-nearest neighbor
 - ▶ naïve Bayes
 - ▶ support vector machine
 - ▶ decision tree
 - ▶ random forest
 - ▶ gradient boosting
 - ▶ multi-layer perceptron
 - ▶ residual networks
 - ▶ ...
- studied 179 classifiers on 121 datasets [Fernández-Delgado et al. 2015]

Choosing an Algorithm

- Over the years, many ML-algorithms were proposed
- Most of these still have a reason for existence
- Examples for classification:
 - ▶ logistic regression
 - ▶ k-nearest neighbor
 - ▶ naïve Bayes
 - ▶ support vector machine
 - ▶ decision tree
 - ▶ random forest
 - ▶ gradient boosting
 - ▶ multi-layer perceptron
 - ▶ residual networks
 - ▶ ...
- studied 179 classifiers on 121 datasets [Fernández-Delgado et al. 2015]
- In practice, we actually want to jointly choose the best ML-algorithm and its hyperparameters

CASH: Combined Algorithm Selection and Hyperparameter Optimization

Definition

Let

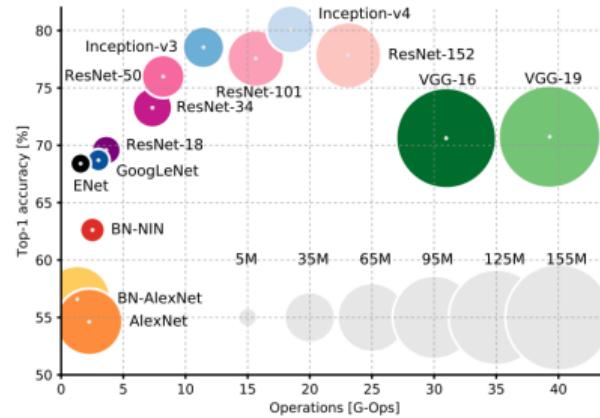
- $\mathbf{A} = \{\mathcal{A}_1, \mathcal{A}_2, \dots, \mathcal{A}_k\}$ be a set of algorithms (a.k.a. portfolio)
- Λ be a set of hyperparameters of each machine learning algorithm \mathcal{A}_i
- \mathcal{D}_{opt} be a dataset which is split into \mathcal{D}_{train} and \mathcal{D}_{valid}
- $c(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of \mathcal{A}_λ trained on \mathcal{D}_{train} and evaluated on \mathcal{D}_{valid} .

we want to find the best combination of algorithm $\mathcal{A} \in \mathbf{A}$ and its hyperparameter configuration $\lambda \in \Lambda$ minimizing:

$$(\mathcal{A}^*, \lambda^*) \in \arg \min_{\mathcal{A} \in \mathbf{A}, \lambda \in \Lambda} c(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Architectures of Neural Networks

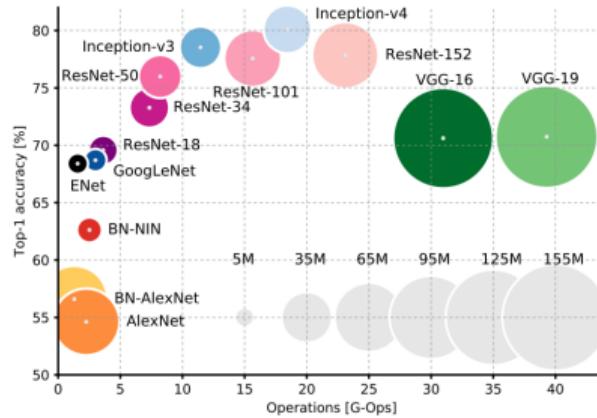
- Many architecture were proposed
- Differences in
 - ▶ Depth
 - ▶ Resolution
 - ▶ Width
 - ▶ Operators
 - ▶ Connections
 - ▶ ...



on Imagenet [Canzian et al. 2017]

Architectures of Neural Networks

- Many architecture were proposed
- Differences in
 - ▶ Depth
 - ▶ Resolution
 - ▶ Width
 - ▶ Operators
 - ▶ Connections
 - ▶ ...

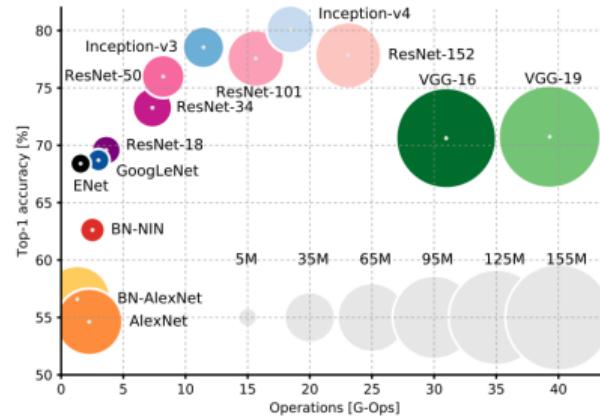


on Imagenet [Canzian et al. 2017]

- Already on a single dataset such as ImageNet, it is not obvious which architecture to choose

Architectures of Neural Networks

- Many architecture were proposed
- Differences in
 - ▶ Depth
 - ▶ Resolution
 - ▶ Width
 - ▶ Operators
 - ▶ Connections
 - ▶ ...

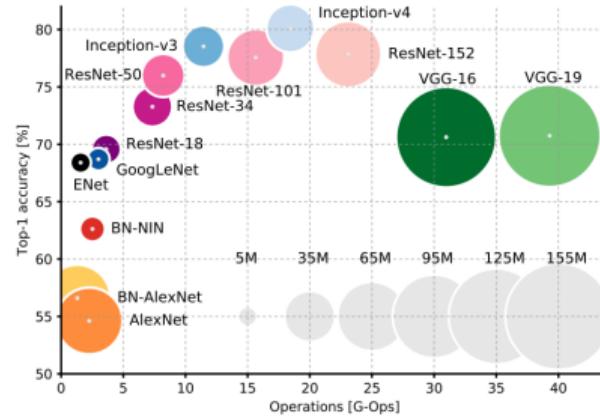


on Imagenet [Canzian et al. 2017]

- Already on a single dataset such as ImageNet, it is not obvious which architecture to choose
- For other datasets, you might need different architectures to achieve top-performance

Architectures of Neural Networks

- Many architecture were proposed
- Differences in
 - ▶ Depth
 - ▶ Resolution
 - ▶ Width
 - ▶ Operators
 - ▶ Connections
 - ▶ ...



on Imagenet [Canzian et al. 2017]

- Already on a single dataset such as ImageNet, it is not obvious which architecture to choose
- For other datasets, you might need different architectures to achieve top-performance
 - ▶ For similar datasets, you might use scaled versions of known architectures (e.g., CIFAR10 and Imagenet)

Neural Architecture Search (NAS)

Definition

Let

- λ be an architecture for a deep neural network N with domain Λ ,
- \mathcal{D}_{opt} be a dataset which is split into \mathcal{D}_{train} and \mathcal{D}_{valid}
- $c(N_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of N_λ trained on \mathcal{D}_{train} and evaluated on \mathcal{D}_{valid} .

The *neural architecture search (NAS)* problem is to find an architecture that minimizes this cost:

$$\lambda^* \in \arg \min_{\lambda \in \Lambda} c(N_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Neural Architecture Search (NAS)

Definition

Let

- λ be an architecture for a deep neural network N with domain Λ ,
- \mathcal{D}_{opt} be a dataset which is split into \mathcal{D}_{train} and \mathcal{D}_{valid}
- $c(N_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of N_λ trained on \mathcal{D}_{train} and evaluated on \mathcal{D}_{valid} .

The *neural architecture search (NAS)* problem is to find an architecture that minimizes this cost:

$$\lambda^* \in \arg \min_{\lambda \in \Lambda} c(N_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Remark:

- very similar to the HPO definition

Neural Architecture Search (NAS)

Definition

Let

- λ be an architecture for a deep neural network N with domain Λ ,
- \mathcal{D}_{opt} be a dataset which is split into \mathcal{D}_{train} and \mathcal{D}_{valid}
- $c(N_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the cost of N_λ trained on \mathcal{D}_{train} and evaluated on \mathcal{D}_{valid} .

The *neural architecture search (NAS)* problem is to find an architecture that minimizes this cost:

$$\lambda^* \in \arg \min_{\lambda \in \Lambda} c(N_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Remark:

- very similar to the HPO definition
- In practice, you want jointly optimize HPO and NAS [Zela et al. 2018]

Per-Instance Algorithm Selection

- For solving the HPO, CASH and NAS problem, we use some kind of optimization approach to [search](#) for a solution

Per-Instance Algorithm Selection

- For solving the HPO, CASH and NAS problem, we use some kind of optimization approach to **search** for a solution
- A different view: Can we learn from data and thus **predict** an algorithm/ hyperparameter configuration/ neural architecture?

Per-Instance Algorithm Selection

- For solving the HPO, CASH and NAS problem, we use some kind of optimization approach to **search** for a solution
- A different view: Can we learn from data and thus **predict** an algorithm/ hyperparameter configuration/ neural architecture?

Definition

Let

- $p(\mathcal{D})$ be a probability **distribution** over datasets $\mathcal{D} \in \mathbf{D}$,

Per-Instance Algorithm Selection

- For solving the HPO, CASH and NAS problem, we use some kind of optimization approach to **search** for a solution
- A different view: Can we learn from data and thus **predict** an algorithm/ hyperparameter configuration/ neural architecture?

Definition

Let

- $p(\mathcal{D})$ be a probability **distribution** over datasets $\mathcal{D} \in \mathbf{D}$,
- \mathbf{P} a portfolio of algorithms $\mathcal{A} \in \mathbf{P}$, and

Per-Instance Algorithm Selection

- For solving the HPO, CASH and NAS problem, we use some kind of optimization approach to **search** for a solution
- A different view: Can we learn from data and thus **predict** an algorithm/ hyperparameter configuration/ neural architecture?

Definition

Let

- $p(\mathcal{D})$ be a probability **distribution** over datasets $\mathcal{D} \in \mathbf{D}$,
- \mathbf{P} a portfolio of algorithms $\mathcal{A} \in \mathbf{P}$, and
- $c : \mathbf{P} \times \mathbf{D} \rightarrow \mathbb{R}$ be a cost metric

Per-Instance Algorithm Selection

- For solving the HPO, CASH and NAS problem, we use some kind of optimization approach to **search** for a solution
- A different view: Can we learn from data and thus **predict** an algorithm/ hyperparameter configuration/ neural architecture?

Definition

Let

- $p(\mathcal{D})$ be a probability **distribution** over datasets $\mathcal{D} \in \mathbf{D}$,
- \mathbf{P} a portfolio of algorithms $\mathcal{A} \in \mathbf{P}$, and
- $c : \mathbf{P} \times \mathbf{D} \rightarrow \mathbb{R}$ be a cost metric

the *per-instance algorithm selection problem* is to obtain a mapping $s : \mathcal{D} \mapsto \mathcal{A}$ such that

$$\arg \min_s \int_{\mathbf{D}} c(s(\mathcal{D}), \mathcal{D}) p(\mathcal{D}) d\mathcal{D}$$

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings
- However, some settings, such as learning rate, have to be adapted over time

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings
- However, some settings, such as learning rate, have to be adapted over time

Definition

Let

- λ be a hyperparameter configuration of an algorithm \mathcal{A} ,

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings
- However, some settings, such as learning rate, have to be adapted over time

Definition

Let

- λ be a hyperparameter configuration of an algorithm \mathcal{A} ,
- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings
- However, some settings, such as learning rate, have to be adapted over time

Definition

Let

- λ be a hyperparameter configuration of an algorithm \mathcal{A} ,
- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,
- s_t be a state description of \mathcal{A} solving \mathcal{D} at time point t ,

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings
- However, some settings, such as learning rate, have to be adapted over time

Definition

Let

- λ be a hyperparameter configuration of an algorithm \mathcal{A} ,
- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,
- s_t be a state description of \mathcal{A} solving \mathcal{D} at time point t ,
- $c : \Pi \times \mathbf{D} \rightarrow \mathbb{R}$ be a cost metric assessing the [cost of a conf.](#) policy $\pi \in \Pi$ on $\mathcal{D} \in \mathbf{D}$

Dynamic Algorithm Configuration

- So far, we assume that an algorithm runs with some given settings
- However, some settings, such as learning rate, have to be adapted over time

Definition

Let

- λ be a hyperparameter configuration of an algorithm \mathcal{A} ,
- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,
- s_t be a state description of \mathcal{A} solving \mathcal{D} at time point t ,
- $c : \Pi \times \mathbf{D} \rightarrow \mathbb{R}$ be a cost metric assessing the [cost of a conf.](#) policy $\pi \in \Pi$ on $\mathcal{D} \in \mathbf{D}$

the *dynamic algorithm configuration problem (DAC)* is to obtain a configuration policy $\pi^* : s_t \times \mathcal{D} \mapsto \lambda$ by optimizing its cost across a distribution of datasets:

$$\pi^* \in \arg \min_{\pi \in \Pi} \int_{\mathbf{D}} p(\mathcal{D}) c(\pi, \mathcal{D}) d\mathcal{D}$$

Summary

HPO Search for the best hyperparameter configuration of a ML algorithm

CASH Search for the best combination of algorithm and hyperparameter configuration

NAS Search for the architecture of neural network

Selection Predict the best algorithm (and its hyperparameter configuration)

DAC Predict the best hyperparameter configuration for an algorithm state
at a given time point

AutoML: Introduction

How is Everything connected?

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

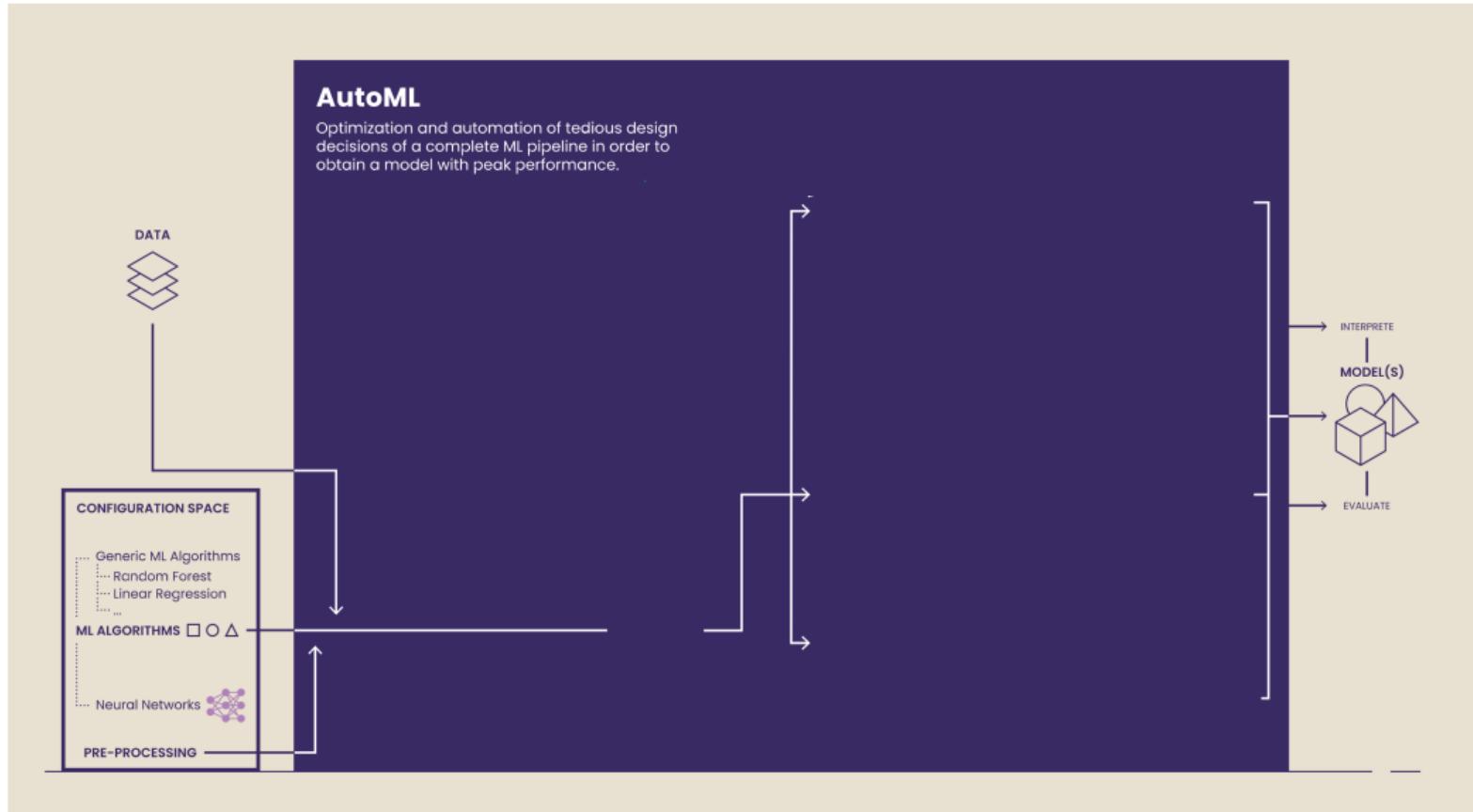
Building a Successful AutoML Tool

~~ There is not a single way for building successful AutoML tools,
but there are well established ideas.

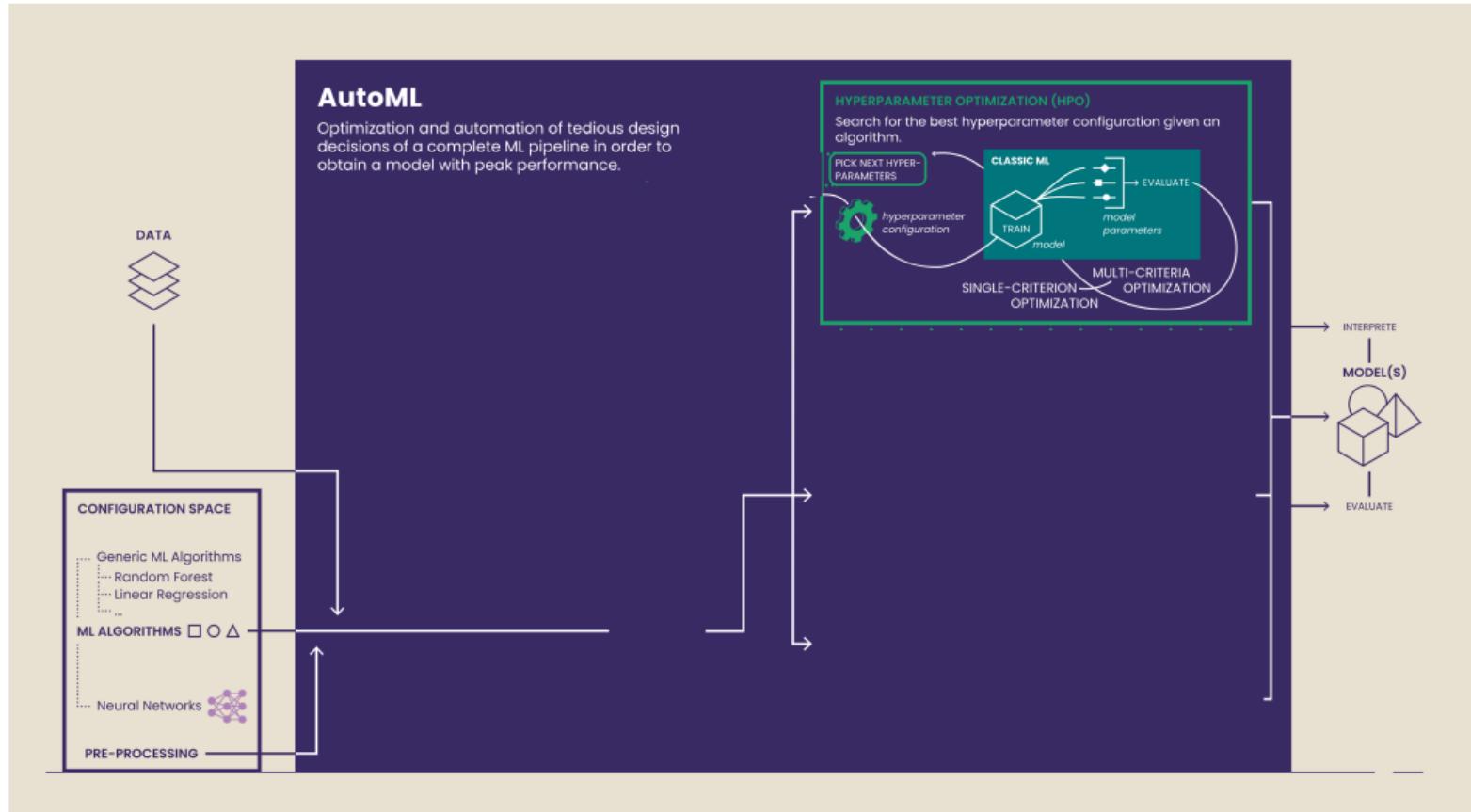
- Hyperparameter optimization
- Neural architecture search
- Model and algorithm selection
- Dynamic approaches

~~ But how can we combine all of these?

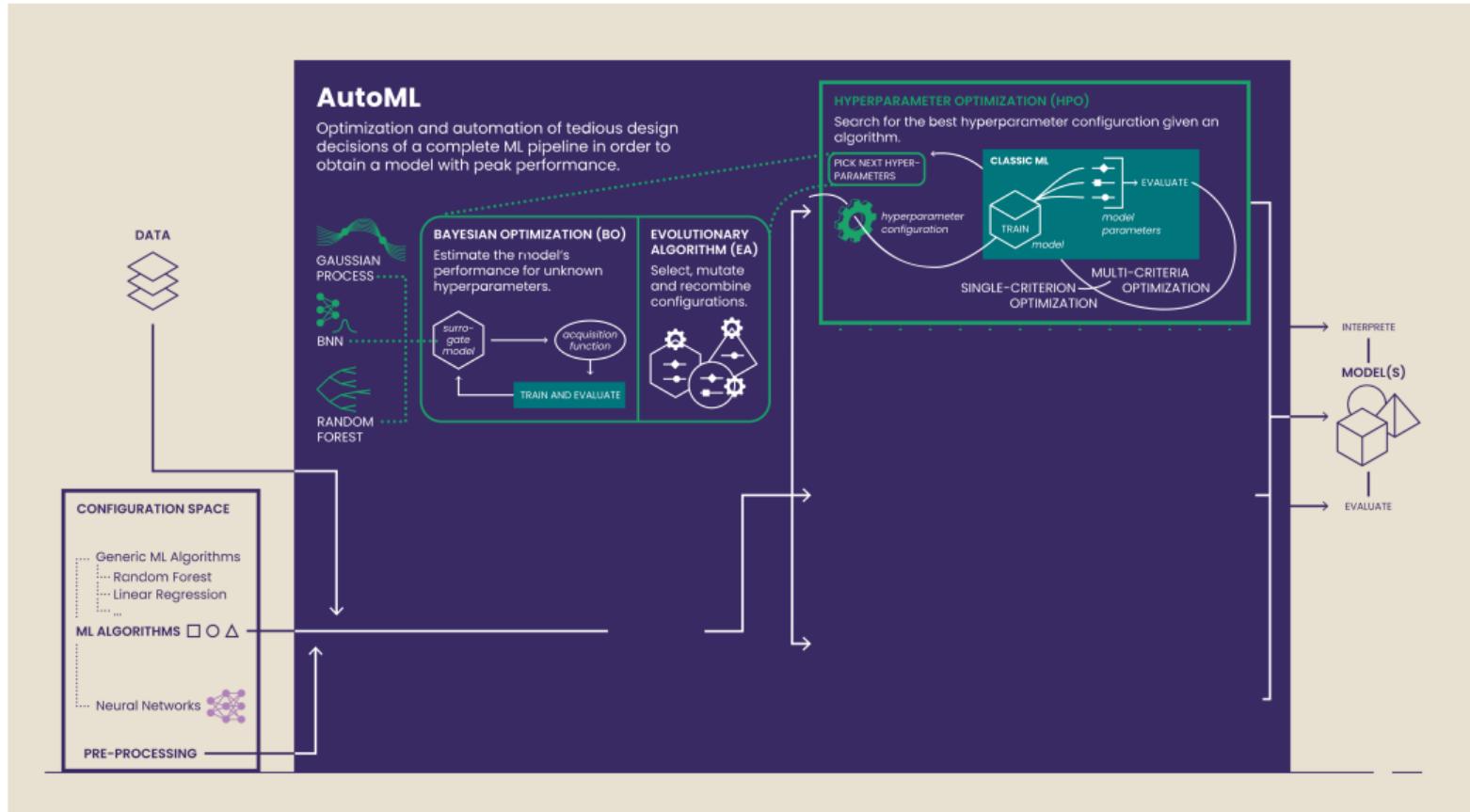
AutoML



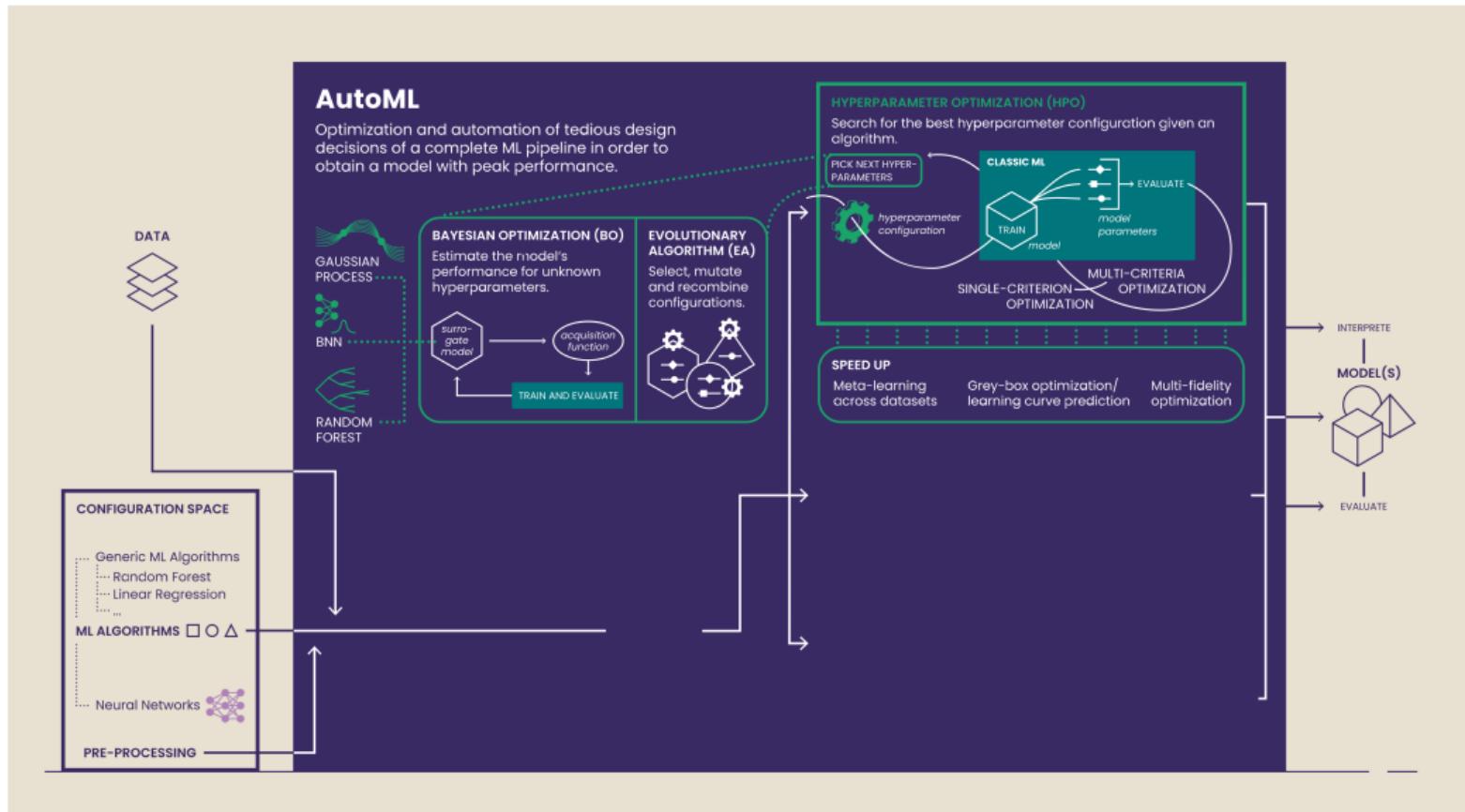
AutoML + HPO



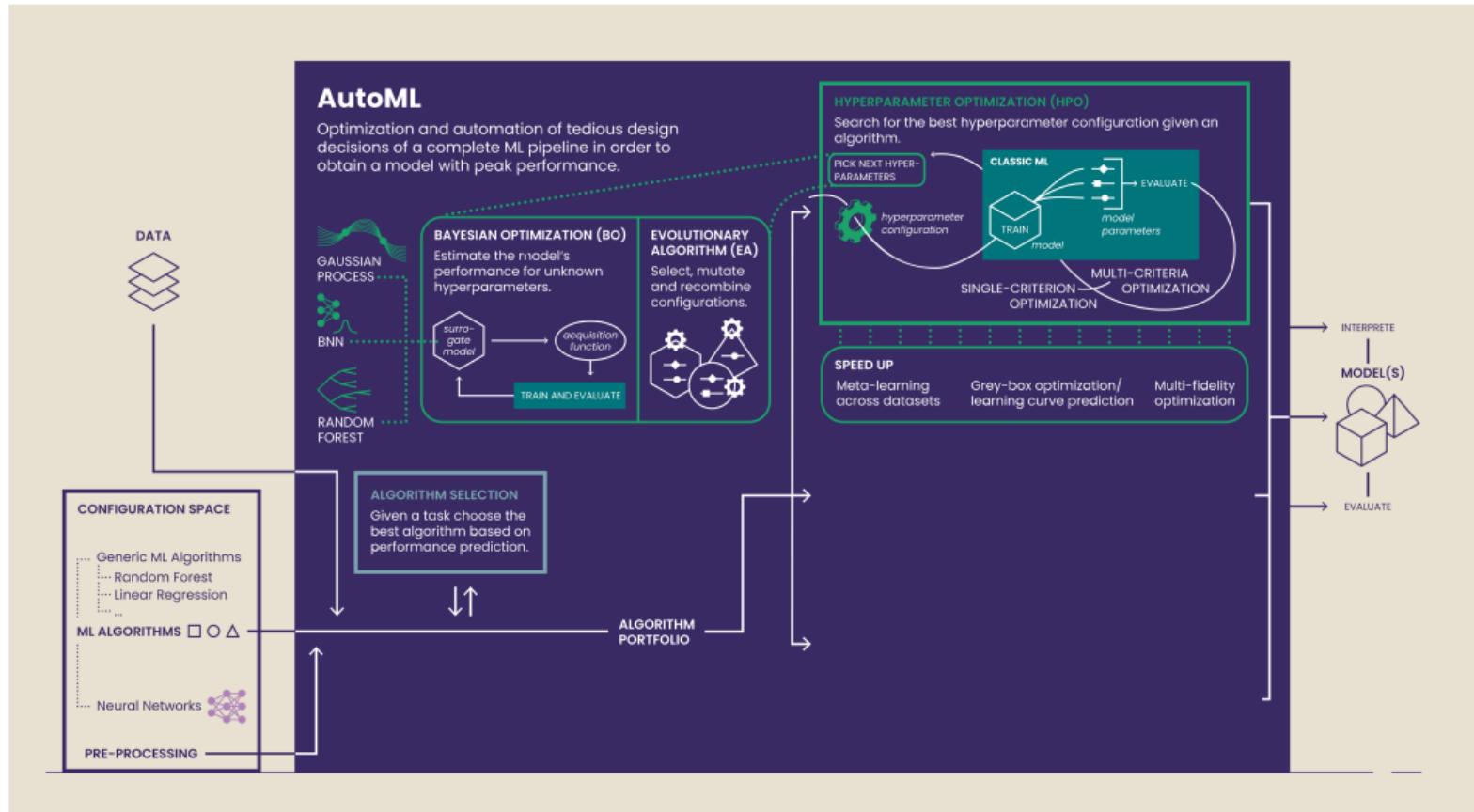
AutoML + HPO + BO



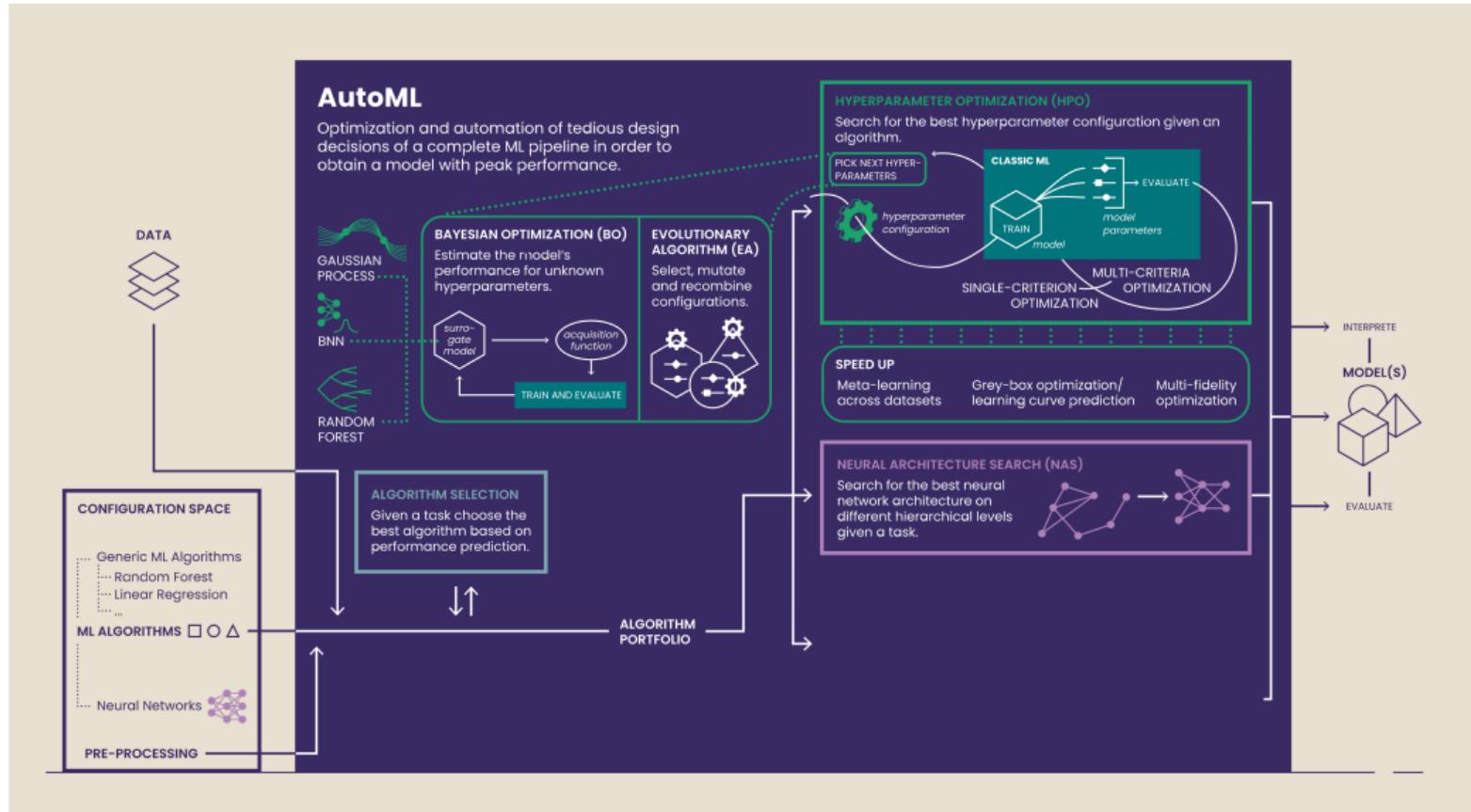
AutoML + HPO + BO + Speed up



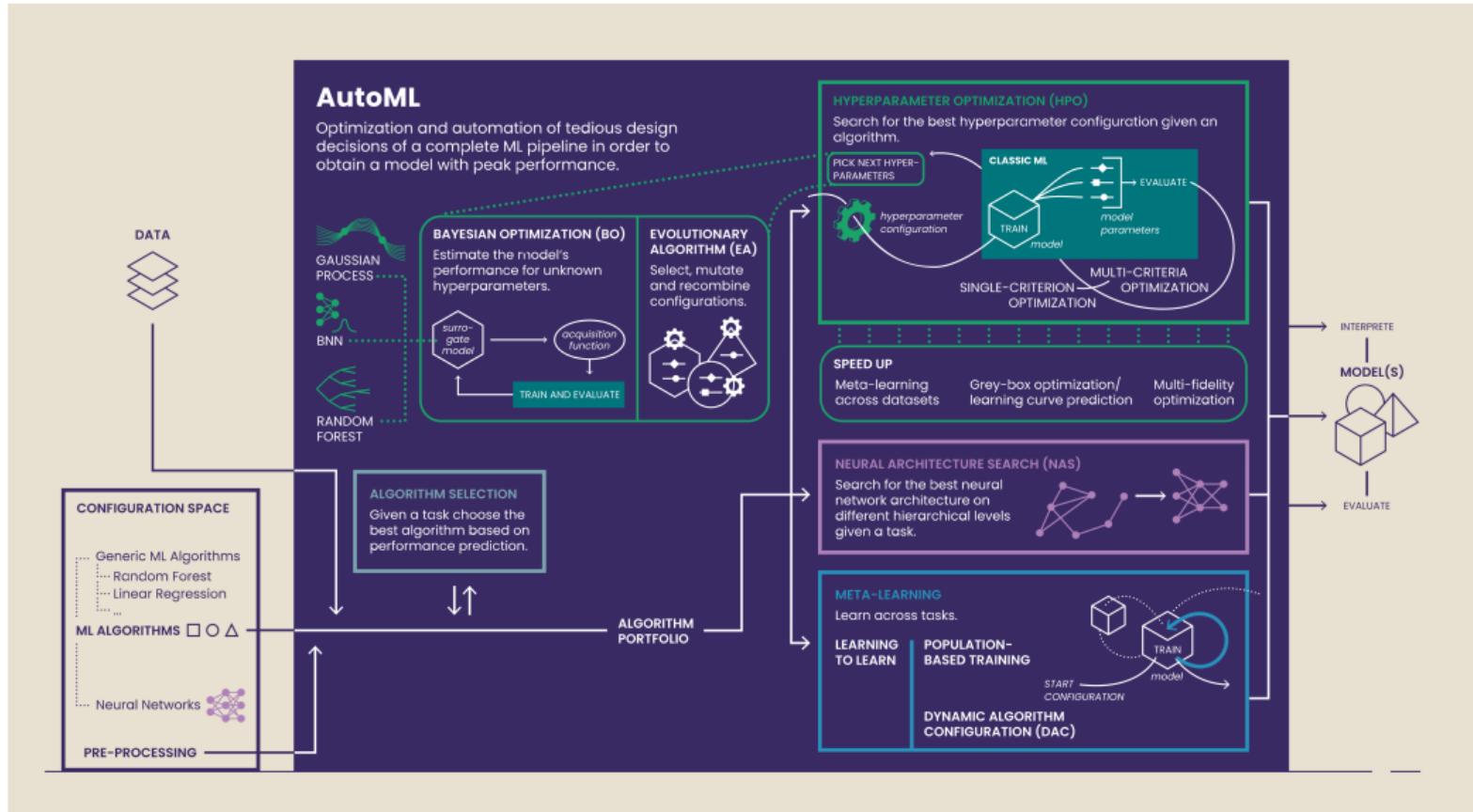
AutoML + HPO + BO + Speed up + AS



AutoML + HPO + BO + Speed up + AS + NAS



AutoML + HPO + BO + Speed up + AS + NAS + Dynamic



AutoML: Introduction

Risks of AutoML

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Risks of AutoML

- ➊ Users apply AutoML **without understanding** anything.
 - ▶ Users might wonder why (Auto-)ML does not perform well after they passed in poor data.

Risks of AutoML

- ② Users trust AutoML too much.
 - ▶ humans might not use human reasoning skills and do not second guess machine decisions

Risks of AutoML

- ③ We enable non-ML experts to use ML
without knowing the risks and consequences of ML.
 - ▶ E.g., bias in data and trained models

Risks of AutoML

- ④ Could result in deployment of ...

Risks of AutoML

- ④ Could result in deployment of ...
 - ▶ inaccurate models due to lack of understanding of statistical concepts, e.g., sampling bias, overfitting, concept drift, ...

Risks of AutoML

④ Could result in deployment of ...

- ▶ **inaccurate models** due to lack of understanding of statistical concepts, e.g., sampling bias, overfitting, concept drift, ...
- ▶ **biased and unfair models** due to lack of understanding ethical practices and use of features such as gender and race for predicting outcomes

Mitigating the Risks

① Raise awareness of the risks

- ▶ Before using AutoML, users should still have a basic understanding of how to apply ML properly

Mitigating the Risks

- ➊ Raise awareness of the risks
 - ▶ Before using AutoML, users should still have a basic understanding of how to apply ML properly

- ➋ Build systems that automatically warn of these risks
 - ▶ AutoML should go hand in hand with bias analysis
 - ~~ Future research topics (as of March 2020)

Mitigating the Risks

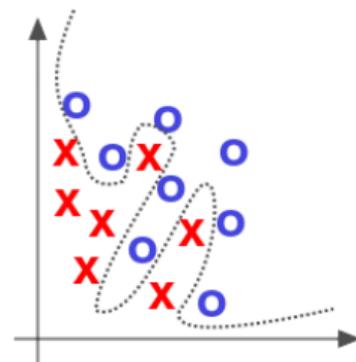
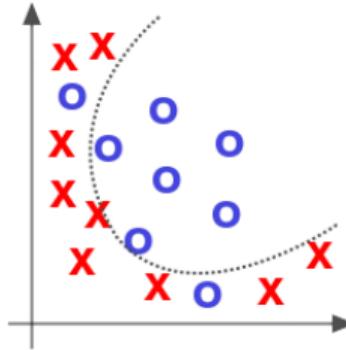
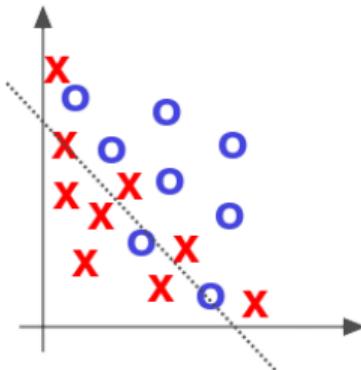
- ➊ Raise awareness of the risks
 - ▶ Before using AutoML, users should still have a basic understanding of how to apply ML properly
- ➋ Build systems that automatically warn of these risks
 - ▶ AutoML should go hand in hand with bias analysis
 - ~~ Future research topics (as of March 2020)
- ➌ Can we automatically find ML systems that are less prone to such risks?
 - ~~ Future research topics (as of March 2020)

AutoML: Evaluation Overview and Motivation

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Training Machine Learning Models

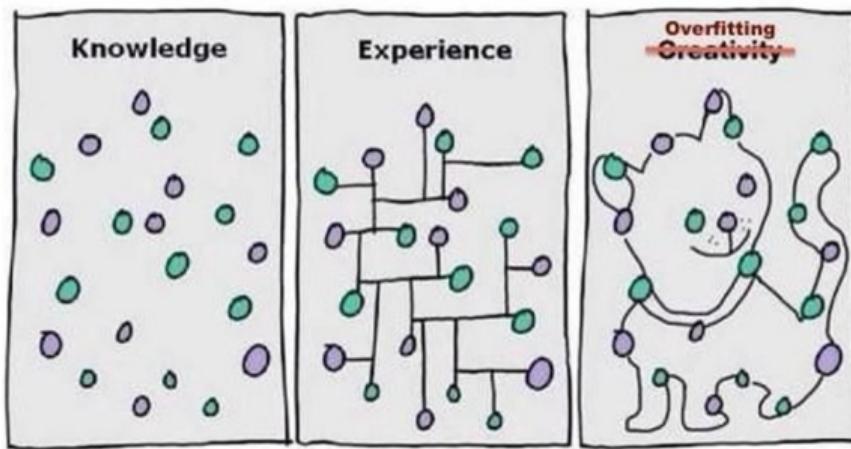
- fundamentally an optimization problem
- determine model parameters such that loss on data is minimized
- quality of fit depends on model class (i.e. degrees of freedom)



Which model is best?

Generalization

- we want models that *generalize* – make “reasonable” predictions on new data
 - ▶ ignore outliers
 - ▶ smooth
 - ▶ captures general trend

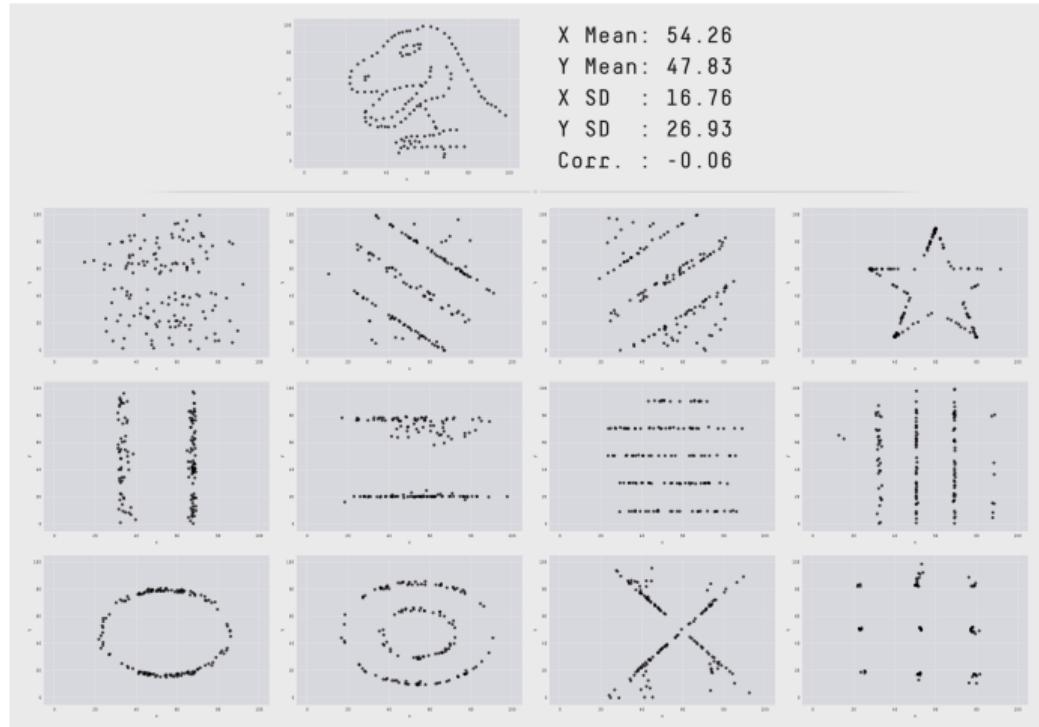


Usually model performance gets better with more data/higher model complexity and then worse, but see [Nakkiran et al. 2019]

Overview

- evaluating machine learning models and quantifying generalization performance
- learning curves
- comparing multiple models/learners on multiple data sets
- statistical tests
- higher levels of optimization, higher levels of evaluation
 - ▶ automated machine learning (meta-optimization) can lead to meta-overfitting
 - ▶ simple training/testing split(s) no longer sufficient → nested evaluation

Evaluate Early, Evaluate Often



AutoML: Evaluation

Evaluation of ML Models (Review)

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Introduction

Performance estimation of a model Estimate generalization error of a model on new (unseen) data, drawn from the same data generating process.

Performance estimation of an algorithm Estimate generalization error of a learning algorithm, trained on a data set of a certain size, on new (unseen) data, all drawn from the same data generating process.

Model selection Select the best model from a set of potential candidate models (e.g., different model classes, different hyperparameter settings, different feature sets).

Performance Evaluation

ML performance evaluation provides clear and simple protocols for reliable model validation.

- often simpler than classical statistical model diagnosis
- relies only on few assumptions
- still hard enough and offers **lots** of options to cheat and make mistakes

Performance Measures

We measure performance using a statistical estimator for the **generalization error** (GE).

GE = expected loss of a fixed model

\hat{GE} = estimated loss averaged across finite sample

Example: Mean squared error (L2 loss)

$$\hat{GE} = MSE = \frac{1}{n} \sum_{i=1}^n \left(y^{(i)} - \hat{y}^{(i)} \right)^2$$

Measures: Inner vs. Outer Loss

Inner loss = loss used in learning (training)
Outer loss = loss used in evaluation (testing)
= evaluation measure

Optimally: inner loss = outer loss

Not always possible:

some losses are hard to optimize or no loss is specified directly

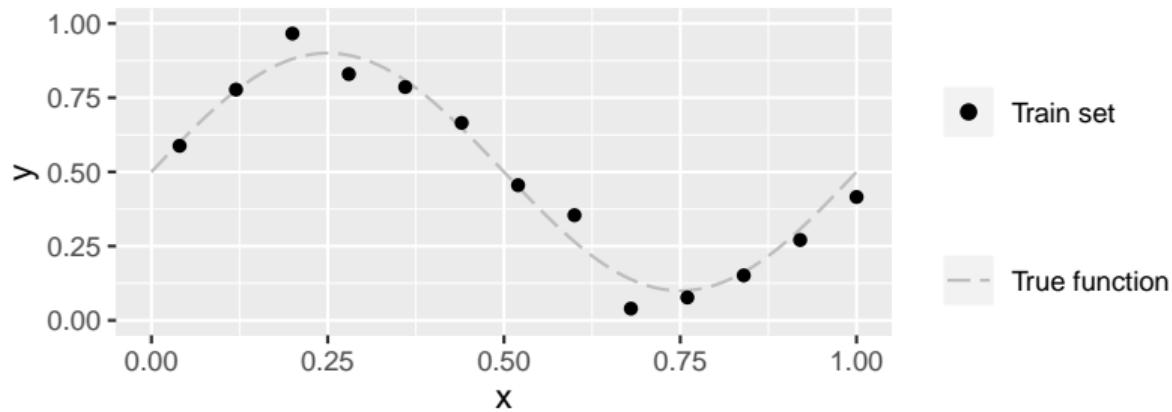
Example:

Logistic Regression → minimize binomial loss

kNN → no explicit loss minimization

Inner Loss Example: Polynomial Regression I

Sample data from sinusoidal function $0.5 + 0.4 \cdot \sin(2\pi x) + \epsilon$ with measurement error ϵ .

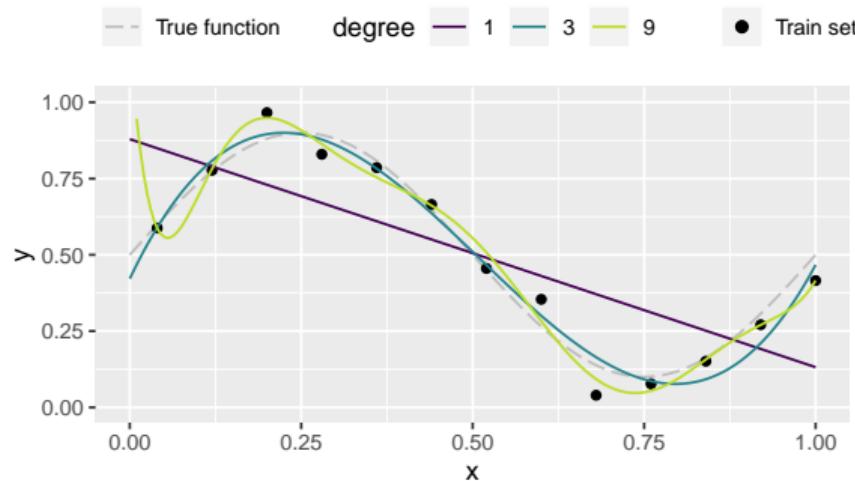


Assume data generating process unknown. Approximate with d th-degree polynomial:

$$f(\mathbf{x}|\theta) = \theta_0 + \theta_1 x + \cdots + \theta_d x^d = \sum_{j=0}^d \theta_j x^j$$

Inner Loss Example: Polynomial Regression II

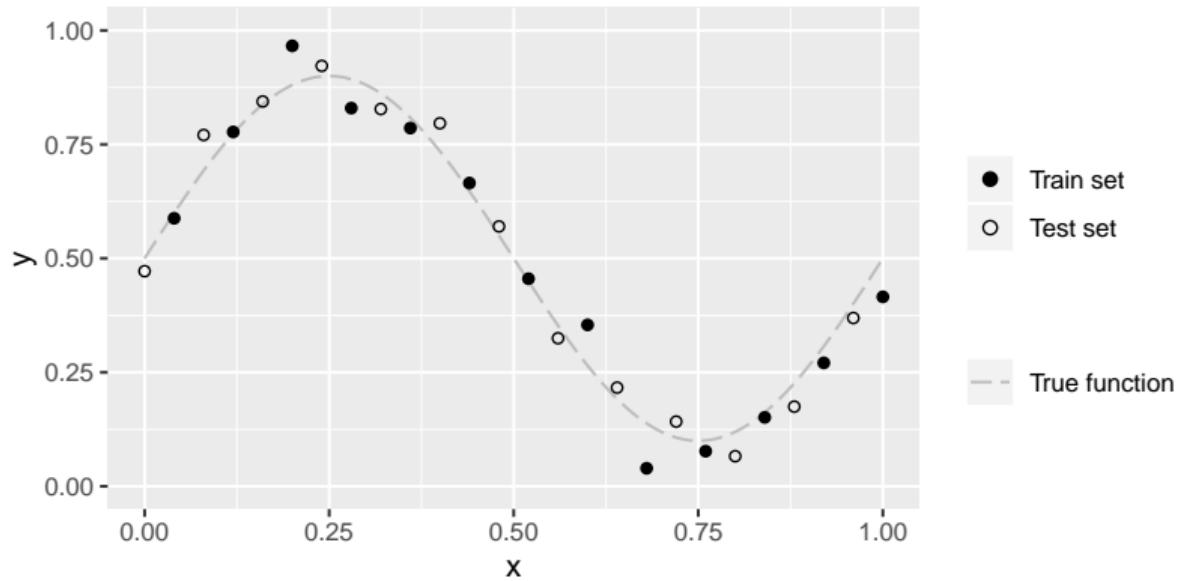
How should we choose d ?



$d=1$: $\text{MSE} = 0.036$ – clear underfitting, $d=3$: $\text{MSE} = 0.003$ – ok?, $d=9$: $\text{MSE} = 0.001$ – clear overfitting

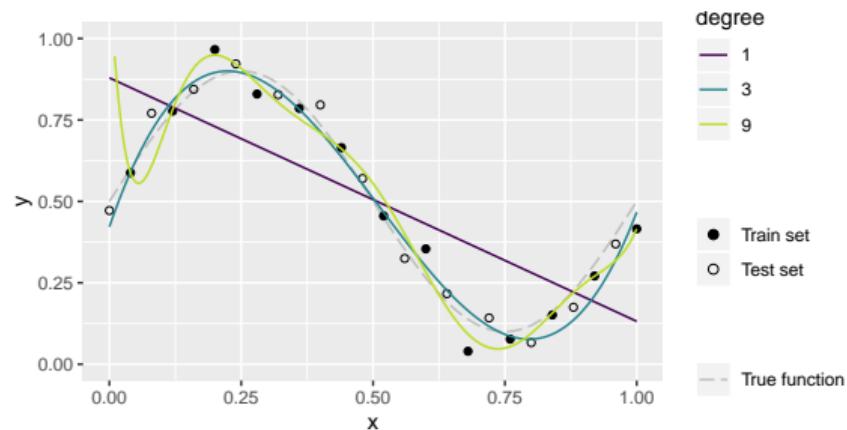
Simply using the training error seems to be a bad idea.

Outer Loss Example: Polynomial Regression I



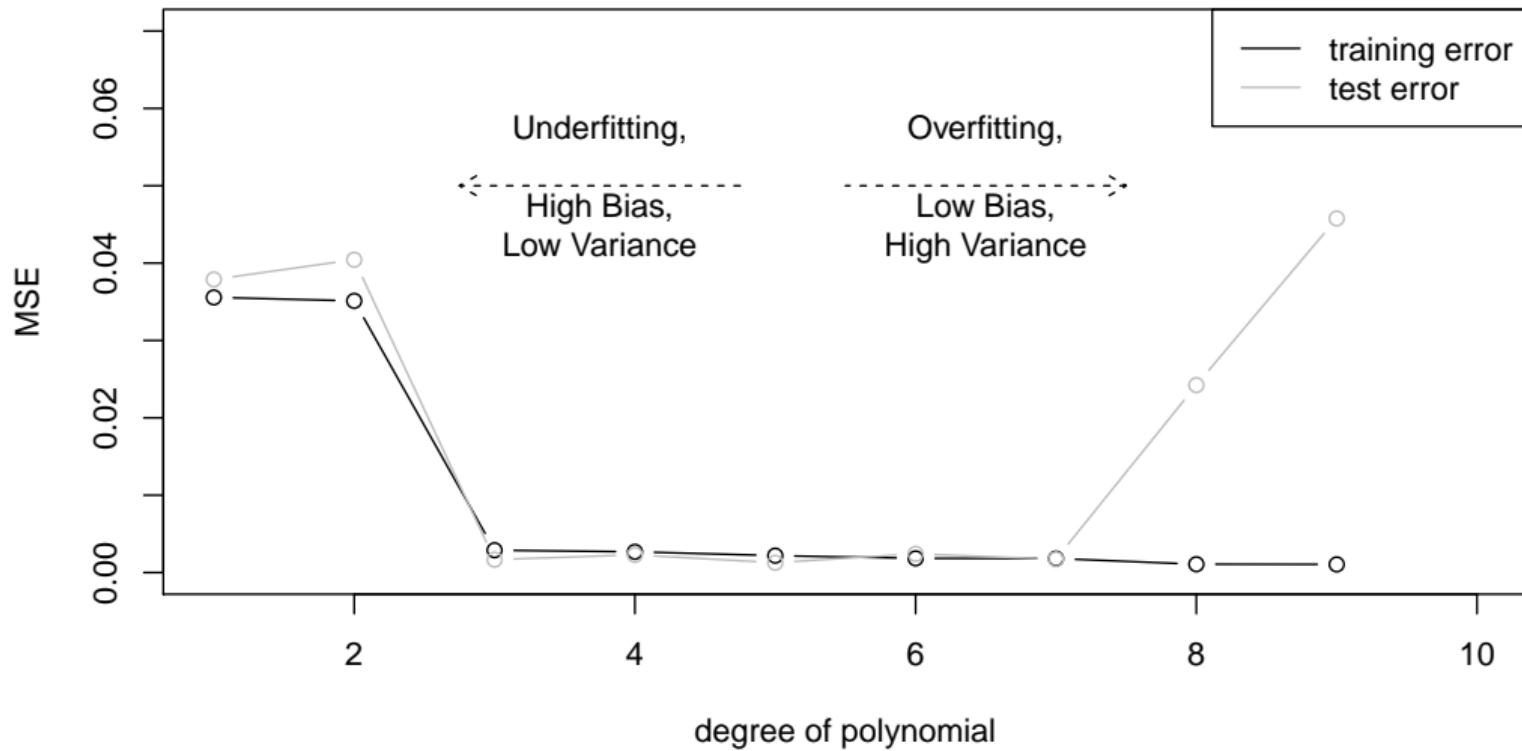
Outer Loss Example: Polynomial Regression II

How should we choose d ?

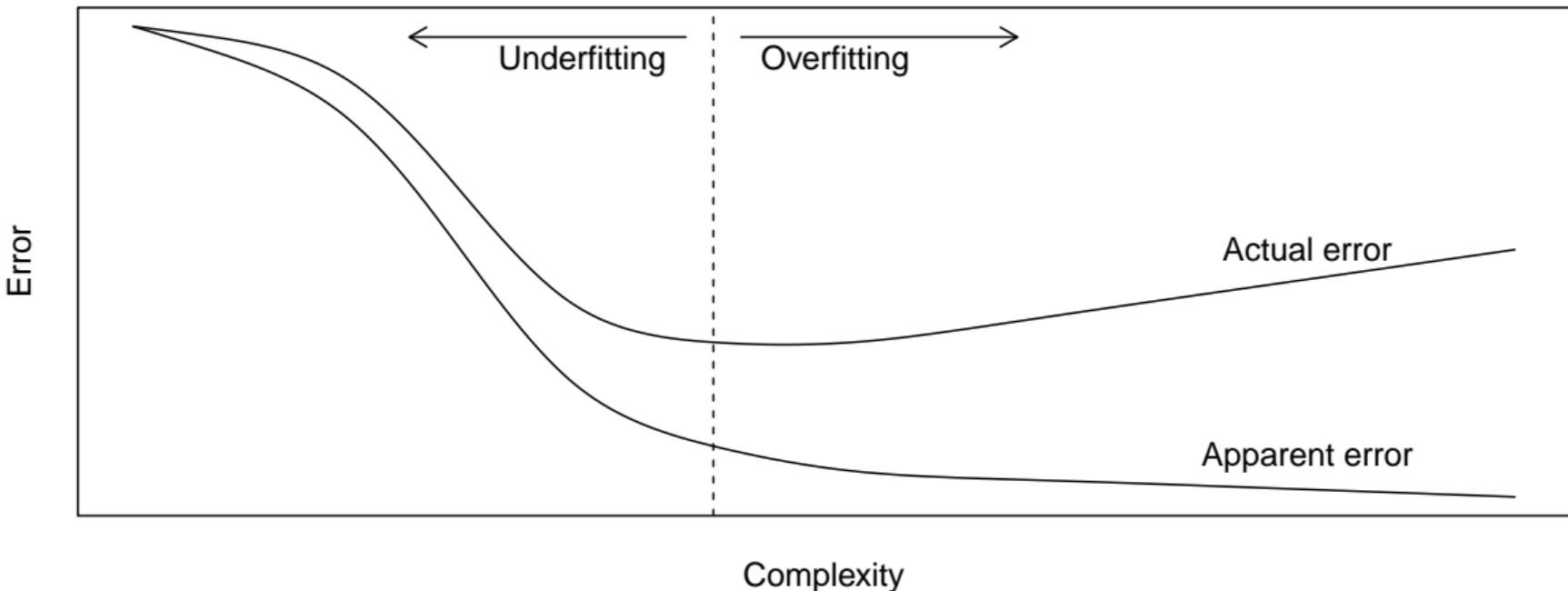


$d=1$: $\text{MSE} = 0.038$ – clear underfitting, $d=3$: $\text{MSE} = 0.002$ – ok?, $d=9$: $\text{MSE} = 0.046$ – clear overfitting

Outer Loss Example: Polynomial Regression III



General Trade-Off Between Error and Complexity



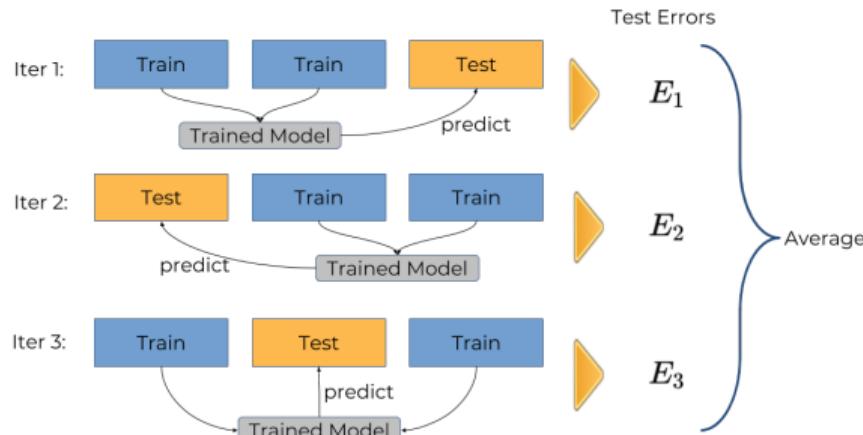
Resampling

- uses data efficiently
- repeatedly split in train and test, average results
- make training sets large (to keep the pessimistic bias small), reduce variance introduced by smaller test sets through many repetitions and averaging of results

Cross-Validation

- split data into k roughly equally-sized partitions
- use each part as test set and join the $k - 1$ others for training, repeat for all k combinations
- obtain k test errors and average

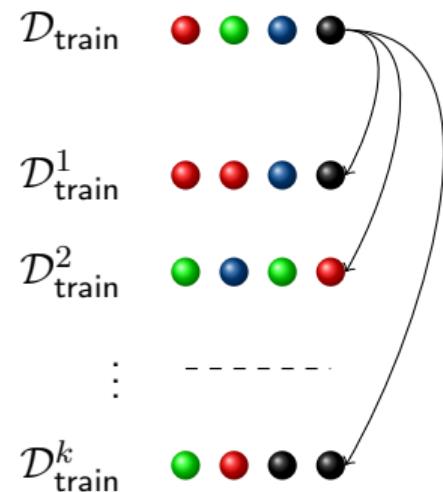
Example 3-fold cross-validation:



10-fold cross-validation is common.

Bootstrap

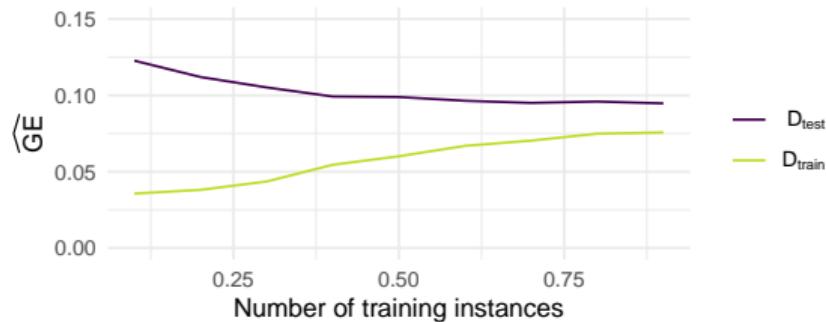
- randomly draw k training sets of size n with replacement from the data
- evaluate on observations from the original data that are not in the training set
- obtain k test errors and average



Training sets will contain about 63.2% of observations on average.

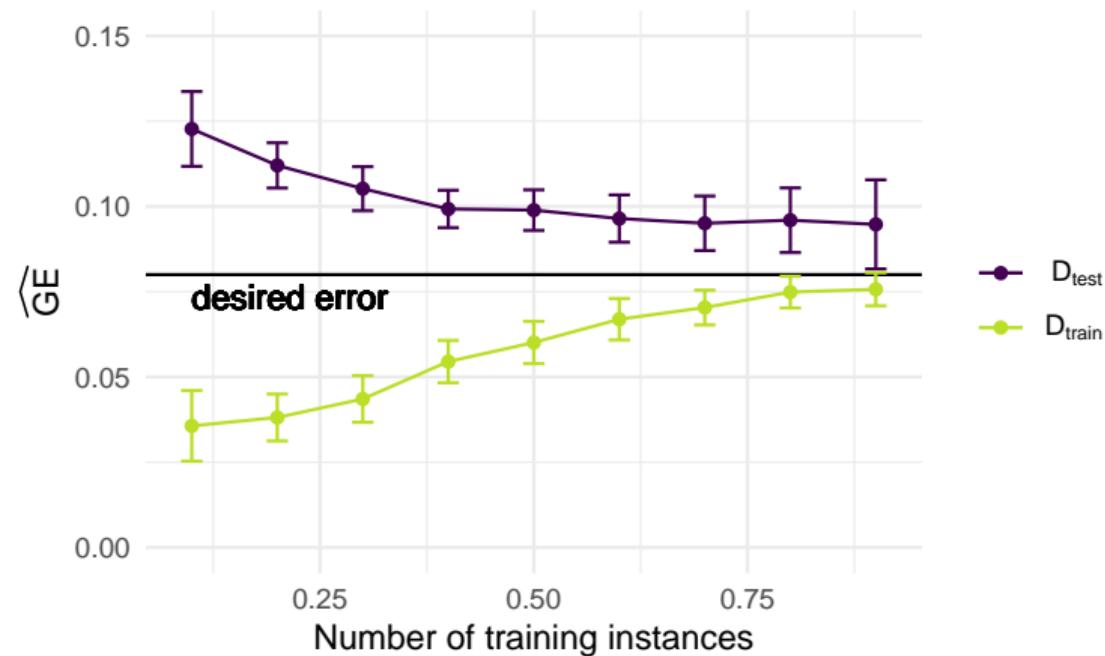
Learning Curves I

- compares performance of a model on training and test data over a varying number of training instances → how fast can a learner learn the given relationship in the data?
- can also be over number of iterations of a learner (e.g. epochs in deep learning), or AutoML system over time
- learning usually fast in the beginning
- visualizes when a learner has learned as much as it can:
 - ▶ when performance on training and test set reach a plateau
 - ▶ when gap between training and test error remains the same



Learning Curves II

Ideal learning curve:



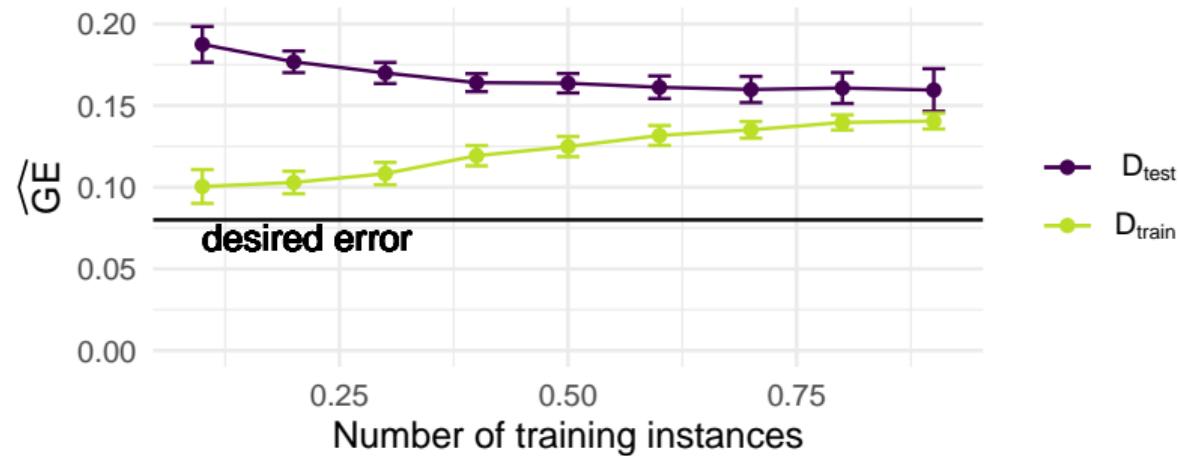
Learning Curves III

In general, there are two reasons for a bad learning curve:

①

high bias in model/underfitting

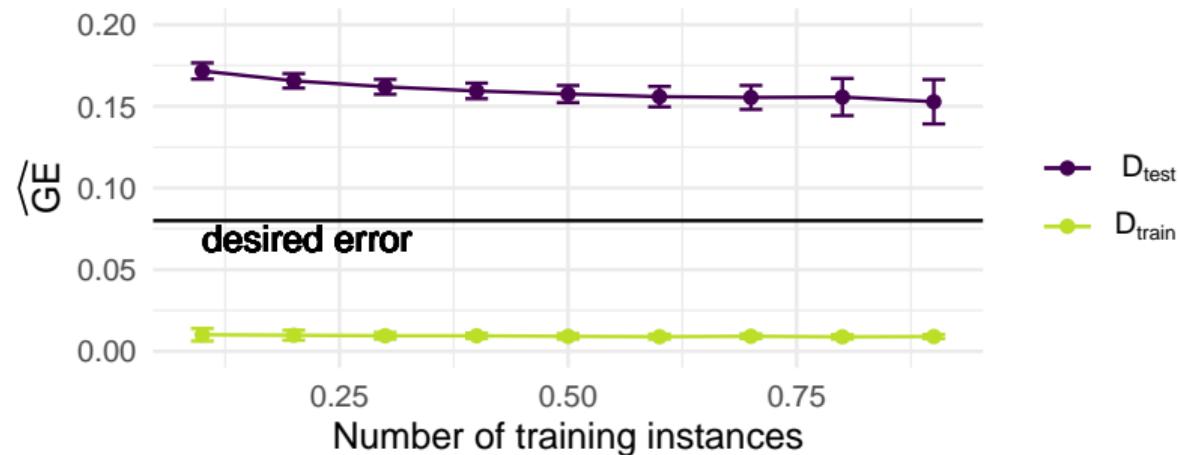
- ▶ training and test errors converge at a high value
- ▶ model can't learn underlying relationship and has high systematic errors, no matter how big the training set
- ▶ poor fit, which also translates to high test error



Learning Curves IV

② high variance in model/overfitting

- ▶ large gap between training and test errors
- ▶ model requires more training data to improve
- ▶ model has a poor fit and does not generalize well



AutoML: Evaluation

Benchmarking and Comparing Learners

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Benchmark Experiments

- different learning algorithms applied to one or more data sets to compare and rank their performances
- synchronized train and test sets, i.e. the same resampling method with the same train-test splits should be used to determine performance

Example: Benchmark results (per CV-fold) of CART and random forest using 2-fold CV with MSE as performance measure:

data set	k-th fold	MSE (rpart)	MSE (randomForest)
BostonHousing	1	29.4	17.13
	2	20.5	8.90
mtcars	1	35.0	7.53
	2	38.9	6.73

Hypothesis Testing in Benchmarking I

We want to know if the difference in performance between models (or algorithms) is significant or only by chance.

Null Hypothesis Statistical Testing (NHST) in Benchmarking:

- formulate null hypothesis H_0 (e.g. the expected generalization error of two algorithms is equivalent) and alternative hypothesis H_1
- use hypothesis test to reject H_0 (not rejecting H_0 does not mean that we accept it)
- rejecting H_0 gives some confidence that the observed results may not be random

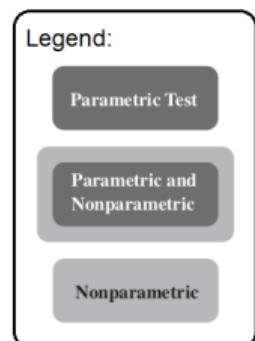
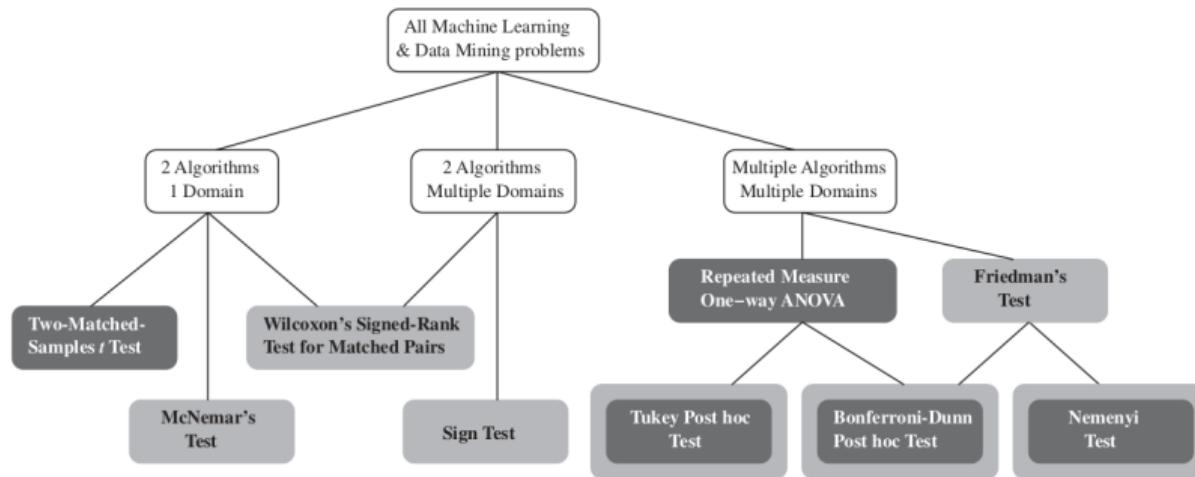
Typical example in machine learning:

- H_0 : on average, model 1 does not perform better than model 2
- H_1 : on average, model 1 outperforms model 2
- Aim: Reject H_0 with confidence level of $1 - \alpha$ (common values for α include 0.05 and 0.01)

Hypothesis Testing in Benchmarking II

Selection of an appropriate hypothesis test is at least based on the type of problem, i.e. if the aim is to compare

- 2 models / algorithms on a single domain (i.e. on a single data set)
- 2 algorithms across different domains (i.e. on multiple data sets)
- multiple algorithms across different domains / data sets



McNemar Test I

- non-parametric test used on paired dichotomous nominal data; does not make any distributional assumptions beyond statistical independence of samples
- pairs are e.g. labels predicted by different models on the same data
- compares the classification accuracy of two **models**
- both models trained and evaluated on the exact same training and test set; **contingency table** based on two paired vectors that indicate whether each model predicted an observation correctly

		Model 2 correct	Model 2 wrong
		A	B
Model 1 correct	correct	A	
	wrong	C	D
Model 1 wrong	correct		
	wrong		

- A: #obs. correctly classified by both
- B: #obs. only correctly classified by model 1
- C: #obs. only correctly classified by model 2
- D: #obs. misclassified by both

McNemar Test II

Error of each model can be computed as follows:

- Model 1: $(C+D)/(A+B+C+D)$
- Model 2: $(B+D)/(A+B+C+D)$

Even if the models have the **same** errors (indicating equal performance), cells B and C may be different because the models may misclassify different instances.

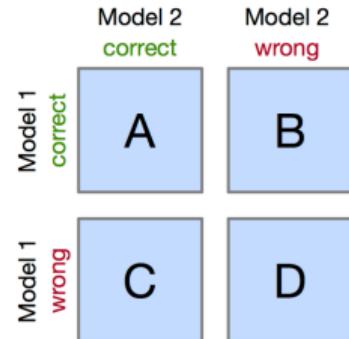
McNemar tests the following hypothesis:

- H_0 : both models have the same performance (we expect $B = C$)
- H_1 : performances of the two models are not equal

The test statistic is computed as

$$\chi_{Mc}^2 = \frac{(|B-C|-1)^2}{B+C} \sim \chi_1^2.$$

Note: The McNemar test should only be used if $B + C > 20$.



McNemar Test III

Example:

		Random Forest	
		0	1
Tree	0	30	5
	1	17	42

Calculating the test statistic:

$$\chi_{Mc}^2 = \frac{(|5 - 17| - 1)^2}{5 + 17} = 5.5 > 3.841 = \chi_{1,0.95}^2$$

We can reject H_0 at a significance level of 0.05, i.e. we reject the hypothesis that the tree and the random forest have the same performance.

Significance level must be chosen before applying the test (avoid p-value hacking).

Two-Matched-Samples t-Test I

- two-matched-samples t-test (i.e. a paired t-test) is the simplest hypothesis test to compare two **models** on a single test set based on arbitrary performance measures
- parametric test and distributional assumptions must be made (which are often problematic):
(pseudo-)normality usually met when sample size > 30
i.i.d. samples usually met if loss of individual observations from single test set considered
equal variances of populations can be investigated through plots

Two-Matched-Samples t-Test II

Compare two different models \hat{f}_1 and \hat{f}_2 w.r.t. performance measure calculated on test set of size n_{test} :

- $H_0: GE(\hat{f}_1) = GE(\hat{f}_2)$ vs. $H_1: GE(\hat{f}_1) \neq GE(\hat{f}_2)$
- test statistic $T = \sqrt{n_{\text{test}}} \frac{\bar{d}}{\sigma_d}$ where
 - ▶ mean performance difference of both models is $\bar{d} = \hat{GE}_{\mathcal{D}_{\text{test}}}(\hat{f}_1) - \hat{GE}_{\mathcal{D}_{\text{test}}}(\hat{f}_2)$
 - ▶ standard deviation of this mean difference is

$$\sigma_d = \sqrt{\frac{1}{n_{\text{test}} - 1} \sum_{i=1}^{n_{\text{test}}} (d_i - \bar{d})^2},$$

where $d_i = L(y^{(i)}, \hat{f}_1(\mathbf{x}^{(i)})) - L(y^{(i)}, \hat{f}_2(\mathbf{x}^{(i)}))$ and $\bar{d} = \frac{1}{n_{\text{test}}} \sum_{i=1}^{n_{\text{test}}} d_i$

Note: d_i is the difference of the outer loss of individual observations from the test set between the two models to be compared.

Two-Matched-Samples t-Test III

- could also use a **k -fold CV paired t-test** to compare two **algorithms** (instead of two models) on single data set
- instead of comparing outer loss of individual observations, compare generalization errors per CV fold (i.e. k generalization errors for k CV folds)
- performance differences are not independent across CV folds due to overlapping training sets (which violates the assumption of i.i.d. samples)
- to partly overcome issue of overlapping training sets across folds, Dietterich suggests using 5 times 2-fold CV so that at least within each repetition neither training nor test sets overlap [Dietterich. 1998]

Friedman Test I

Compare multiple classifiers on multiple data sets:

- H_0 : all algorithms are equivalent in their performance and hence their average ranks should be equal
- H_1 : the average ranks for at least one algorithm is different

To evaluate n data sets and k algorithms:

- rank each algorithm on each data set from best-performing algorithm (rank 1) to worst-performing algorithm using any performance measure
- R_{ij} is the rank of algorithm j on data set i
- if there is a d -way tie after rank r , assign rank of $[(r + 1) + (r + 2) + \dots + (r + d)] / d$ to each tied classifier

Friedman Test II

Can now compute:

- overall mean rank $\bar{R} = \frac{1}{nk} \sum_{i=1}^n \sum_{j=1}^k R_{ij}$
- sum of squares total $SS_{Total} = n \sum_{j=1}^k (\bar{R}_{.j} - \bar{R})^2$ where $\bar{R}_{.j} = \frac{1}{n} \sum_{i=1}^n R_{ij}$
- sum of squares error $SS_{Error} = \frac{1}{n(k-1)} \sum_{i=1}^n \sum_{j=1}^k (R_{ij} - \bar{R})^2$

Test statistic calculated as:

$$\chi_F^2 = \frac{SS_{Total}}{SS_{Error}} \sim \chi_{k-1}^2 \text{ for large } n (>15) \text{ and } k (>5)$$

For smaller n and k, the χ^2 approximation is imprecise and a look up of χ_F^2 values that were approximated specifically for the Friedman test is suggested.

Post-Hoc Tests I

- Friedman test checks if all algorithms are ranked equally
- does not allow to identify best-performing algorithm

→ post-hoc tests

Post-hoc Nemenyi test:

- compares all pairs of algorithms to find best-performing algorithm after H_0 of the Friedman-test was rejected
- for n data sets and k algorithms, $\frac{k(k-1)}{2}$ comparisons
- calculate average rank of algorithm j on all n data sets: $\bar{R}_{.j} = \frac{1}{n} \sum_{i=1}^n R_{ij}$

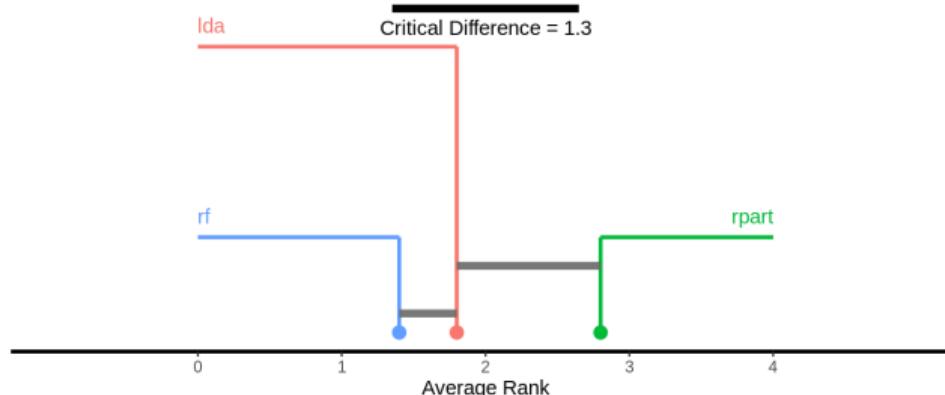
For any two algorithms j_1 and j_2 , test statistic computed as:

$$q = \frac{\bar{R}_{.j_1} - \bar{R}_{.j_2}}{\sqrt{\frac{k(k+1)}{6n}}}$$

Post-Hoc Tests II

Critical Difference Plot:

- quick way to see what differences are significant across all compared learners
- all learners that do not differ by at least the critical difference are connected by line
- a learner not connected to another learner and of lower rank can be considered better according to the chosen significance level



Post-Hoc Tests III

Post-hoc Bonferroni-Dunn test:

- compares all algorithms with baseline (i.e. $k - 1$ comparisons)
- used after Friedman test to find which algorithms differ from the baseline significantly
- uses Bonferonni correction to prevent randomly accepting one of the algorithms as significant due to multiple testing

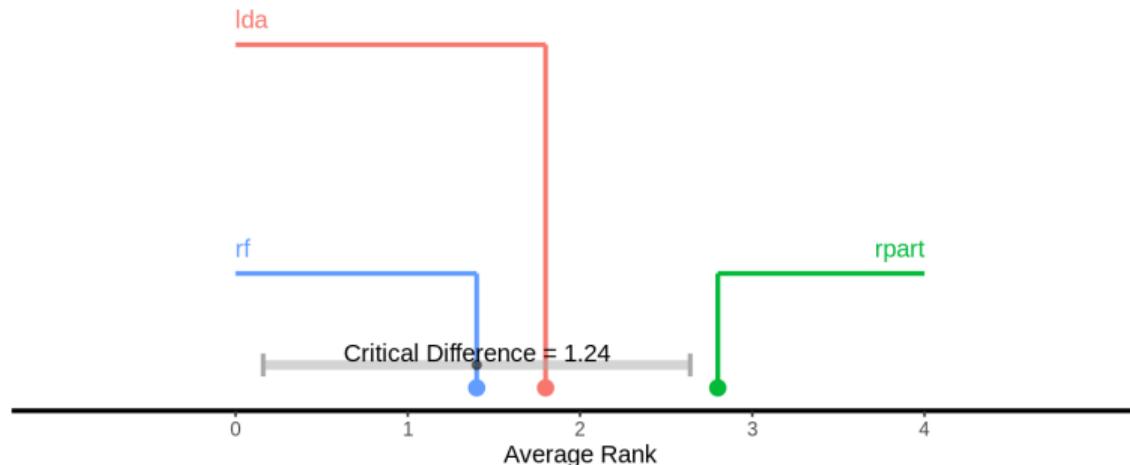
The test statistic is the same as before:

$$q = \frac{\bar{R}_{j_1} - \bar{R}_{j_2}}{\sqrt{\frac{k(k+1)}{6n}}}.$$

The performance of j_1 and j_2 are significantly different when $|q| > q_\alpha$, where the critical value q_α is obtained from a table of the studentized range statistic divided by $\sqrt{2}$.

Post-Hoc Tests IV

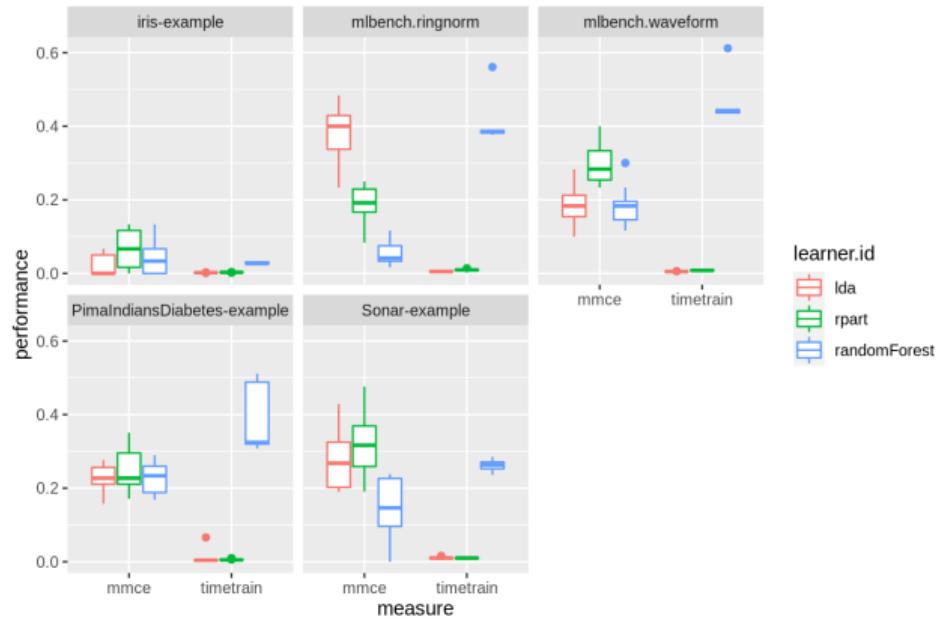
- learners within the baseline interval (gray line) perform not significantly different from the baseline



Comparing Visually I

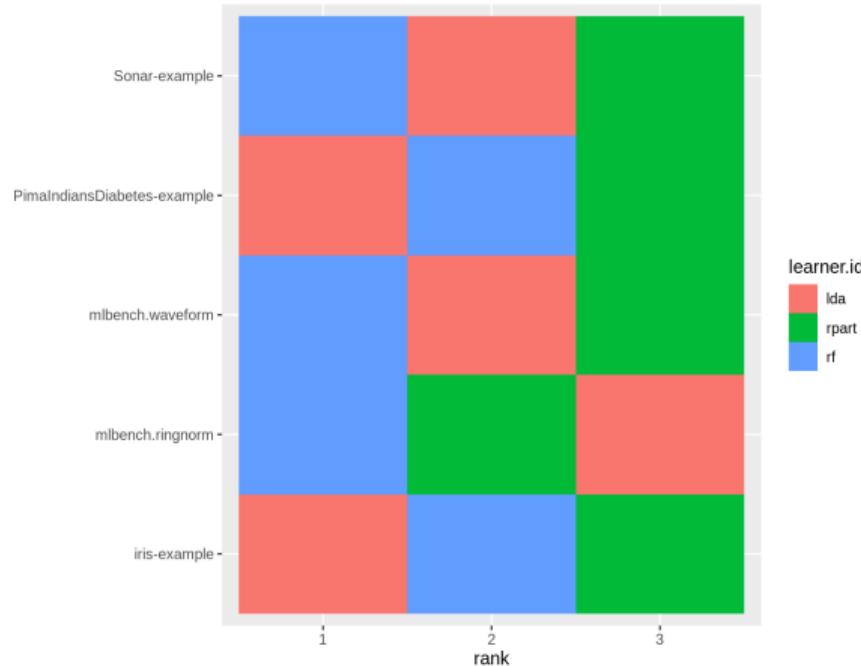
It can be helpful to inspect distributions visually for additional insights, e.g.

Boxplots



Comparing Visually II

Rank plots



AutoML: Evaluation

Nested Resampling

Bernd Bischl¹ Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

¹Some slides taken from Bernd Bischl's "Introduction to Machine Learning" lecture at LMU. [Bischl]

Motivation

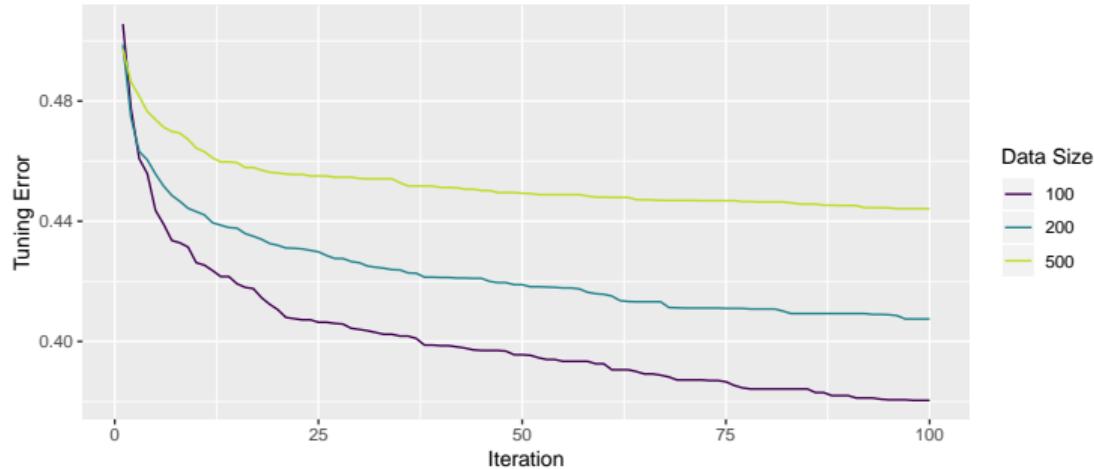
Selecting the best model from a set of potential candidates (e.g. different classes of learners, different hyperparameter settings, different feature sets, different preprocessing. . .) is an important part of most machine learning problems. However,

- cannot evaluate selected learner on the same resampling splits used to select it
- repeatedly evaluating learner on same test set or same CV splits “leaks” information about test set into evaluation
- danger of overfitting to the resampling splits or overtuning
- final performance estimate would be optimistically biased
- similar to multiple hypothesis testing

Motivating Example I

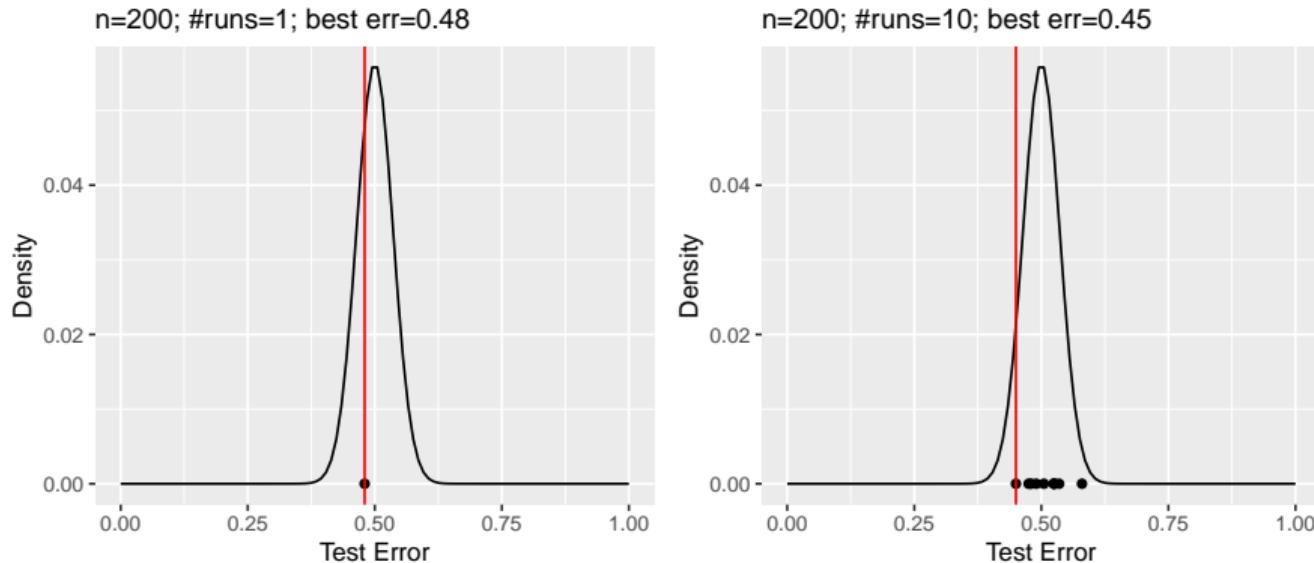
- binary classification problem with equal class sizes
- learner with hyperparameter λ
- learner is (nonsense) feature-independent classifier that picks random labels with equal probability, λ has no effect
- true generalization error is 50%
- cross-validation of learner (with any fixed λ) will easily show this (if the partitioned data set for CV is not too small)
- let's "tune" it by trying out 100 different λ values
- repeat this experiment 50 times and average results

Motivating Example II



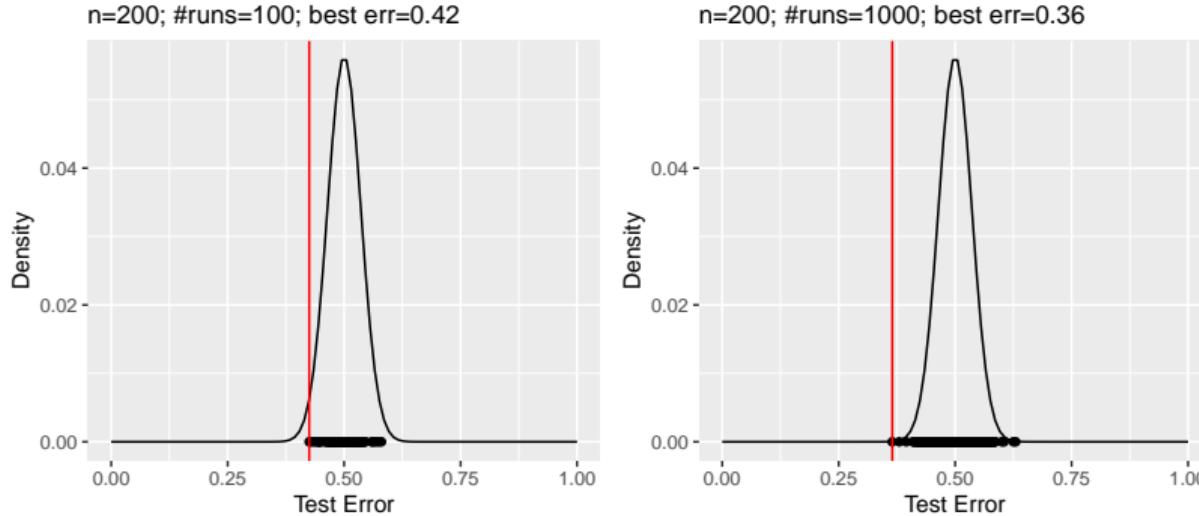
- shown is best “tuning error” (i.e. performance of model with fixed λ in cross-validation) after k tuning iterations
- evaluated for different data set sizes

Motivating Example III



- for one experiment, CV score is close to 0.5, as expected
- errors essentially sampled from (rescaled) binomial distribution
- scores from multiple experiments also arranged around expected mean of 0.5

Motivating Example IV



- tuning means we take the minimum of the scores
- not estimate of average performance, but best-case performance
- the more we sample, the more “biased” this value becomes → unrealistic generalization performance estimate

Untouched Test Set Principle I

Instead: simulate what actually happens when applying machine learning models

- all parts of model construction (including model selection, preprocessing) evaluated **on training data**
- test set only touched once, so no way of “cheating”
- test dataset is only used once *after* model is completely trained (including e.g. deciding hyperparameter values)
- performance estimates from test set now **unbiased estimates** of the true performance

Untouched Test Set Principle II

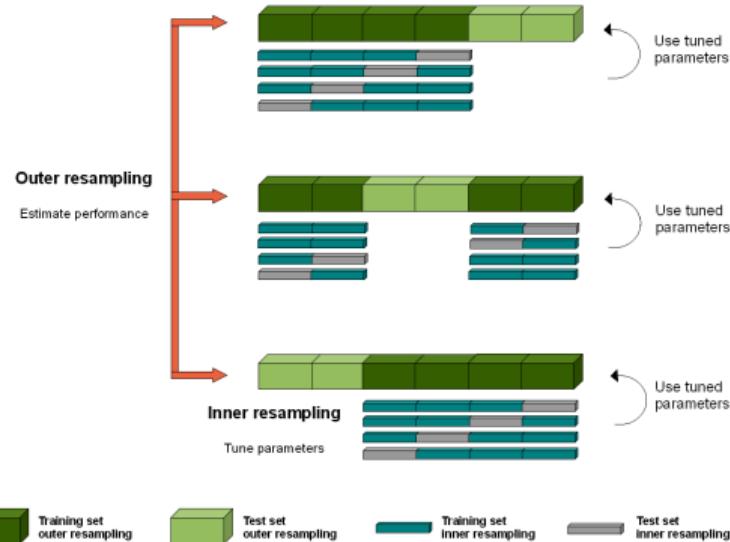
- for steps that themselves require resampling (e.g. hyperparameter tuning) this results in **nested resampling**, i.e. resampling strategies for both
 - ▶ inner evaluation to find what works best based on training data
 - ▶ outer evaluation on data not used in inner evaluation to get unbiased estimates of expected performance on new data

Nested Resampling I

- holdout can be generalized to resampling for more reliable generalization performance estimates
- resampling can be generalized to nested resampling
- nested resampling loops for inner and outer evaluation for hyperparameter tuning

Nested Resampling II

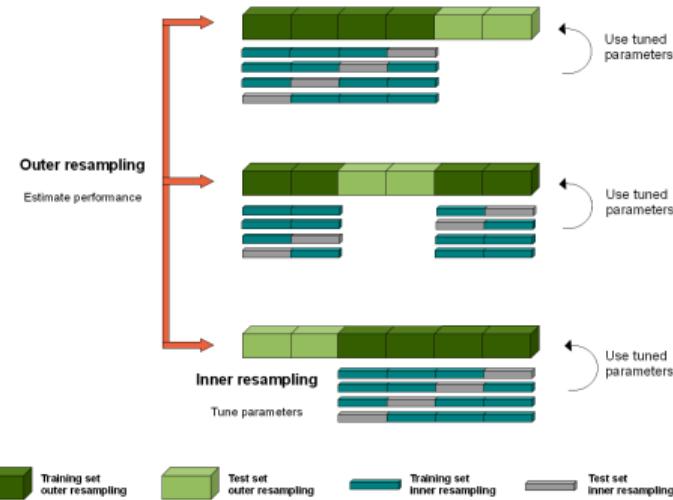
Example: four-fold CV for inner resampling, three-fold CV in outer resampling



Nested Resampling III

In each iteration of the outer loop:

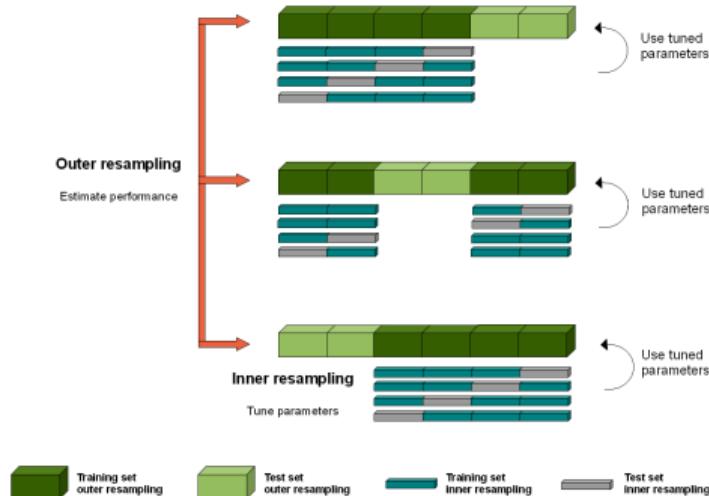
- split light green testing data
- run hyperparameter tuner on dark green part of data, i.e. evaluate each λ_i through four-fold CV on dark green part



Nested Resampling IV

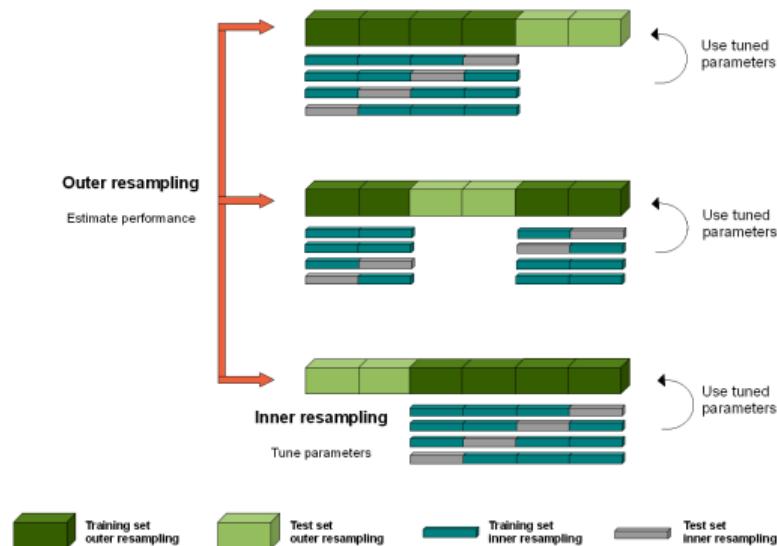
In each iteration of the outer loop:

- return winning $\hat{\lambda}$ that performed best on the grey inner test sets
- re-train model on full outer dark green training set
- evaluate model on outer light green test set



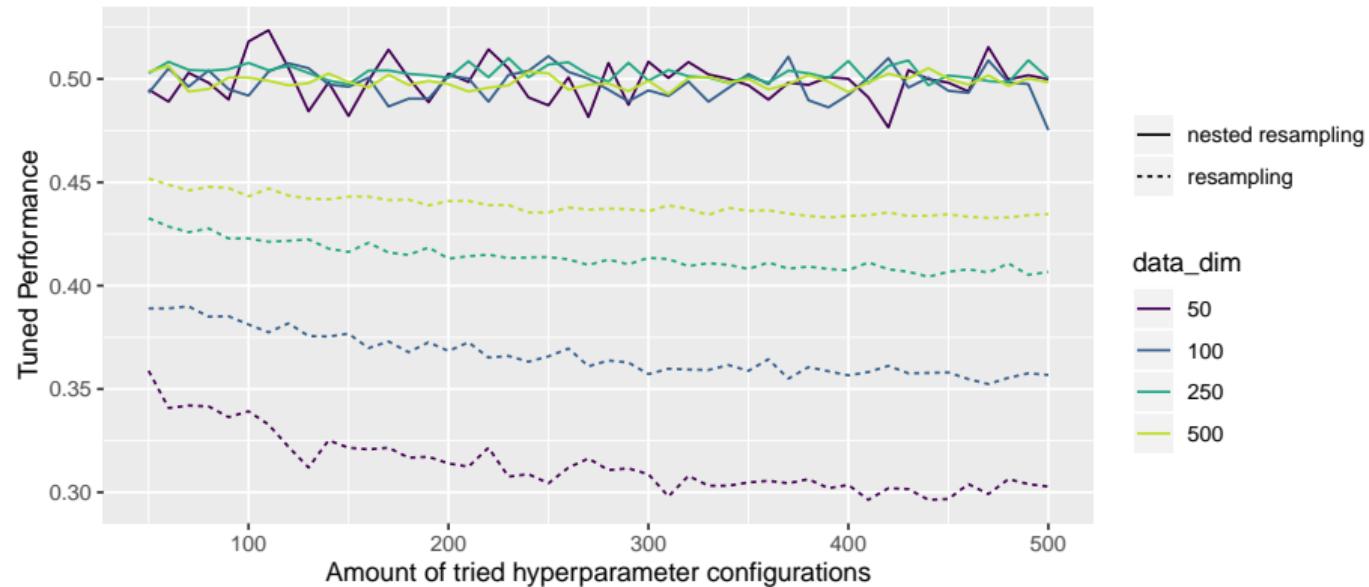
Nested Resampling V

→ error estimates on outer samples (light green) are unbiased because this data was not used process constructing the tested model



Nested Resampling Example

Revisited motivating example: expected performance estimate with nested resampling:



AutoML: Algorithm Selection Overview and Motivation

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Algorithm Selection

Given a problem, choose the best algorithm to solve it. [Rice. 1975]

Algorithm Selection

More formally

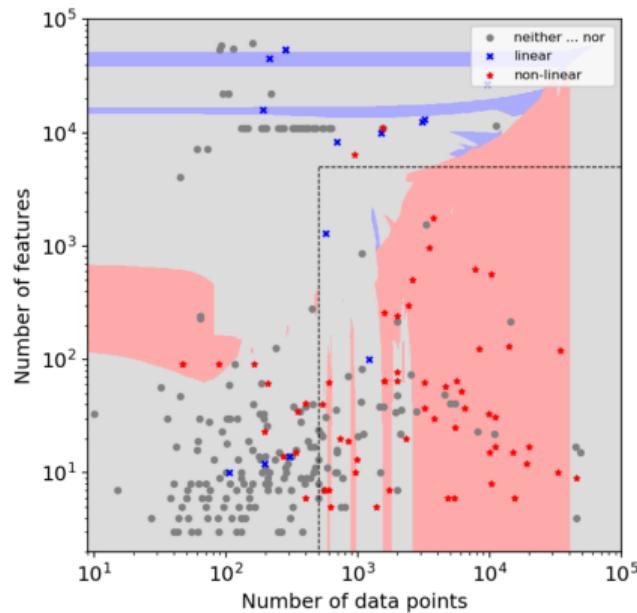
Let

- $p(\mathcal{D})$ be a probability distribution over datasets $\mathcal{D} \in \mathbf{D}$,
- \mathbf{P} a portfolio of algorithms $\mathcal{A} \in \mathbf{P}$, and
- $c : \mathbf{P} \times \mathbf{D} \rightarrow \mathbb{R}$ be a cost metric

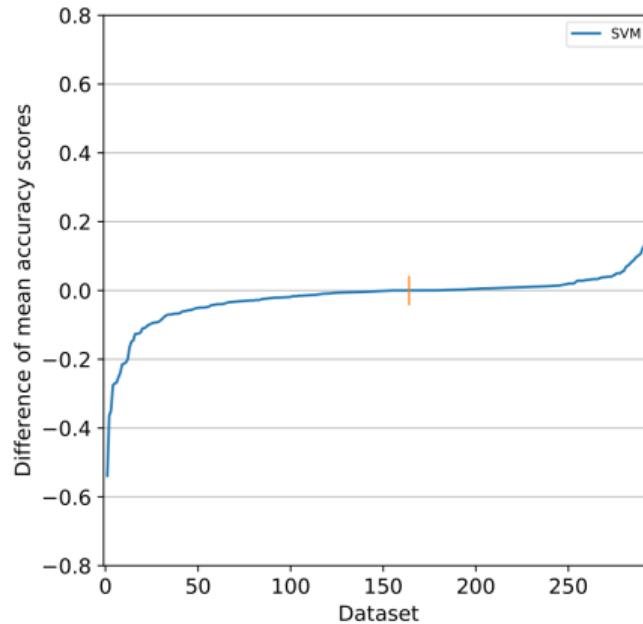
the *per-instance algorithm selection problem* is to obtain a mapping $s : \mathcal{D} \mapsto \mathcal{A}$ such that

$$\arg \min_s \int_{\mathbf{D}} c(s(\mathcal{D}), \mathcal{D}) p(\mathcal{D}) d\mathcal{D}$$

Motivation: Performance Differences [Strang et al. 2018] |



Motivation: Performance Differences [Strang et al. 2018] II

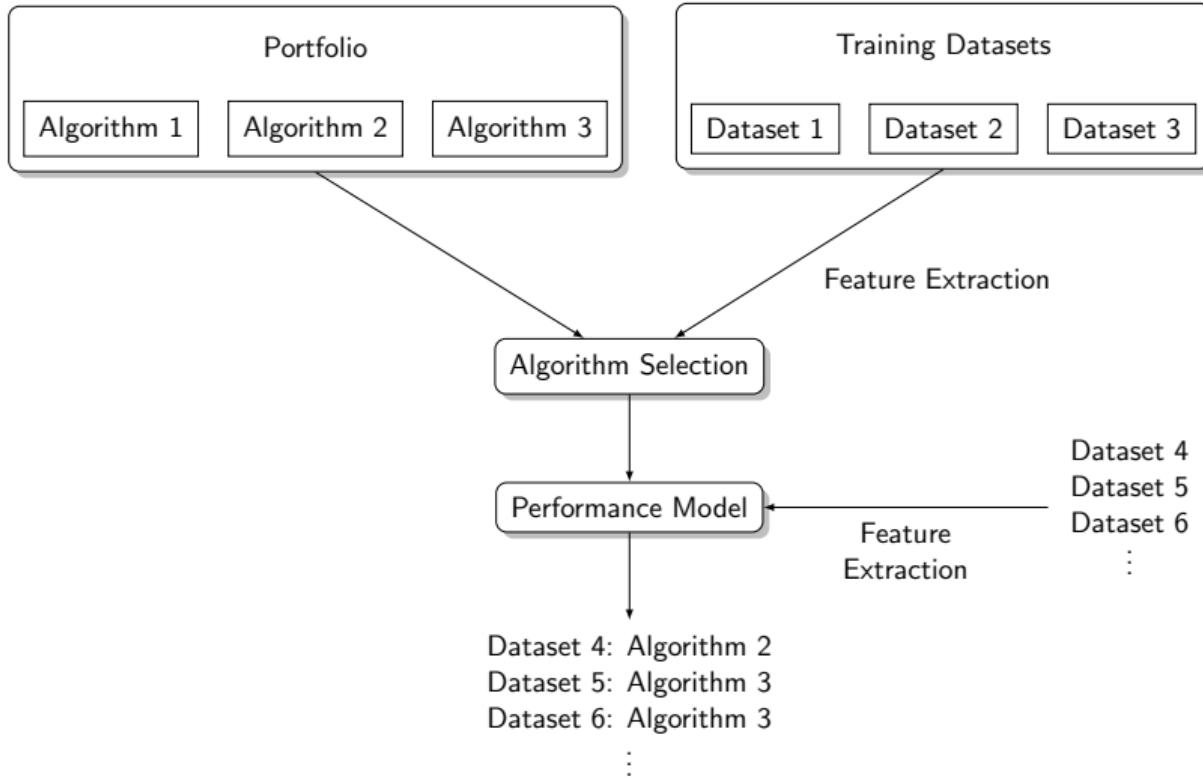


AutoML: Algorithm Selection

Algorithm Selection

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Algorithm Selection



Algorithm Portfolios

- instead of a single algorithm, use several (hopefully complementary) algorithms
- idea from Economics – minimize risk by spreading it out across several securities
- same here – minimize risk of algorithm performing poorly
- in practice often constructed from algorithms known to perform well
- idea similar to ensembles or boosting – leverage strengths and alleviate weaknesses, but learn which algorithm to choose for a particular dataset

Algorithms

“algorithm” used in a very loose sense

- different learners
- different parameterizations of the same learner
- different ensembles, boosted learners
- different machine learning workflows/pipelines
- ...

Evaluation of Portfolios

- single best algorithm
 - ▶ algorithm with the best performance across all datasets
 - ▶ lower bound for performance of portfolio – hopefully we are better!
- virtual best algorithm
 - ▶ choose the best algorithm for each dataset
 - ▶ corresponds to oracle predictor or overhead-free parallel portfolio
 - ▶ upper bound on portfolio performance

Parallel Portfolios

Why not simply run all algorithms in parallel?

- not enough resources may be available/waste of resources
- algorithms may be parallelized themselves
- memory contention
- ...

Building an Algorithm Selection System

- most approaches rely on (meta-)machine learning
- train with representative data, i.e. performance of all algorithms in portfolio on representative datasets
- evaluate performance on separate set of datasets
- potentially large amount of prep work
- existing repositories of machine learning performances (e.g. OpenML) can help

Choosing Datasets

- we want selectors that generalize, i.e. good for more than one dataset
- split datasets into training set (which we learn a selector on) and test set (which we only evaluate performance on)
- need to balance easy/hard datasets in both sets
- may need a lot of data

Key Components of an Algorithm Selection System

- feature extraction
- performance model
- prediction-based selector

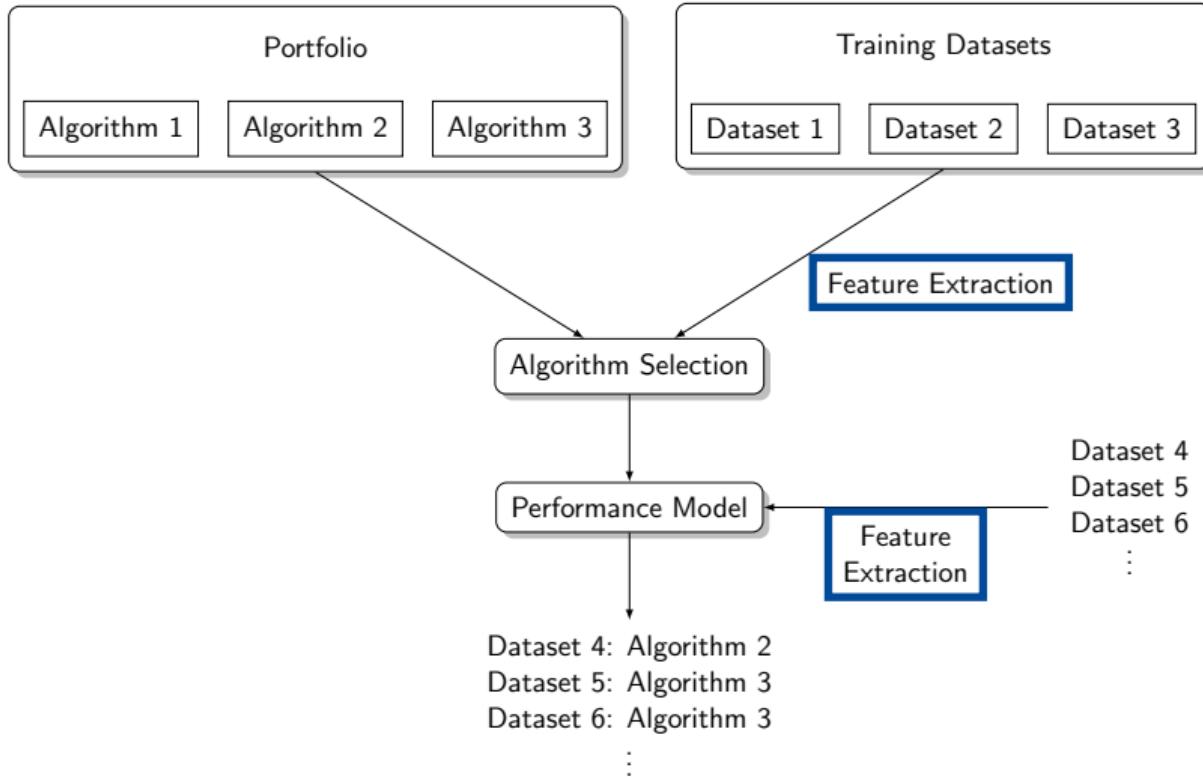
optional:

- presolver
- secondary/hierarchical models and predictors (e.g. for feature extraction time to avoid spending a long time for small performance gains)

AutoML: Algorithm Selection Features

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Algorithm Selection



Features

- relate properties of datasets to algorithm performance
- relatively cheap to compute – must be cheaper than running the algorithm to see what its performance is
- often specified by domain expert
- syntactic and information-theoretic – analyze dataset
- probing – run an algorithm for short time or on subset of data

Syntactic and Information-Theoretic Features

- number of binary/numeric/categorical features
- number of classes
- class entropy
- skewness of classes
- fraction of missing values
- correlation between features and target
- ...

Probing Features (Landmarkers)

- performance of majority class/mean value predictor
- decision stump performance
- simple rule model performance
- performance of algorithm of interest on 1% of data
- ...

→ usually leads to much better results than using just syntactic and information-theoretic features

No Features

- use deep learning to process dataset or problem instance as-is
- no need for expert-designed features
- only preliminary applications so far, performance not good, no widespread adoption yet

Aside: Algorithm Features

- can characterize algorithm in addition to datasets
- allows to relate performance to specific aspects of an algorithm rather than black boxes
- for example size of code base, properties of abstract syntax tree. . .
- ongoing work

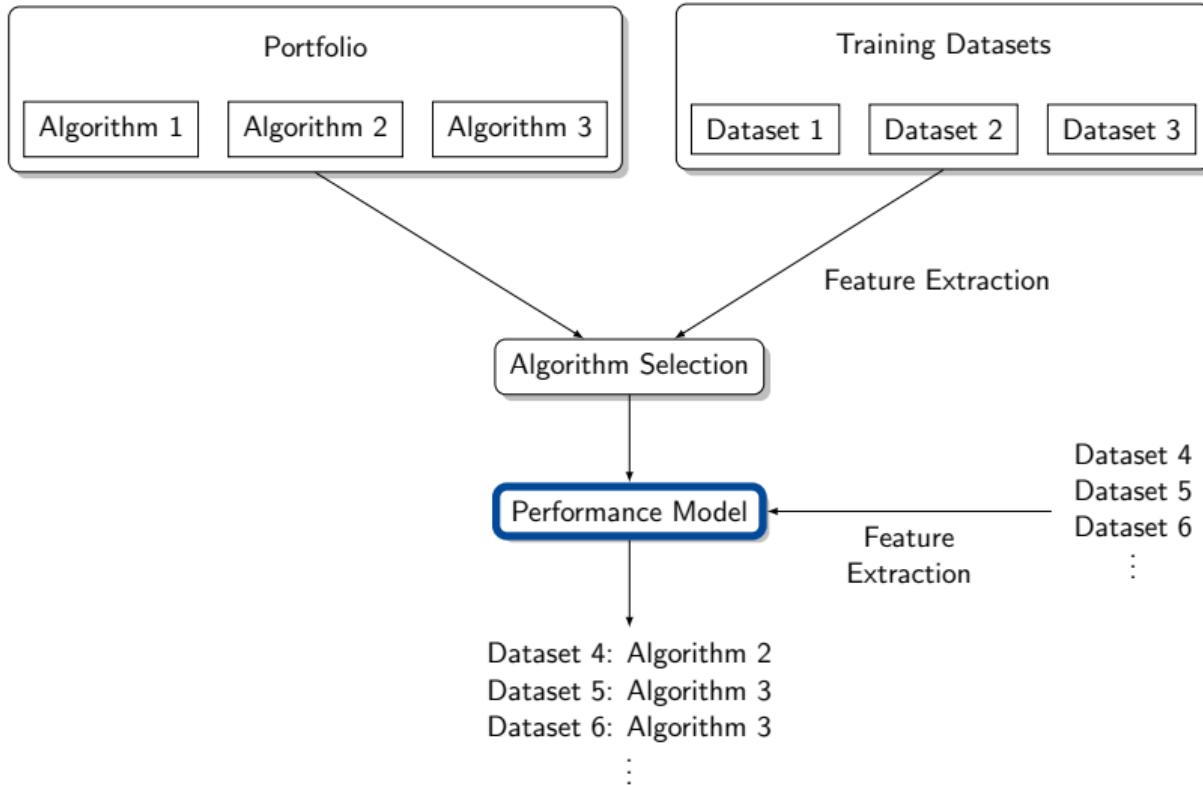
What Features Do We Need in Practice?

- trade-off between complex features and complex models
- in practice, very simple features can perform well
- often only few features of a set are needed (e.g. 5 out of >100)
- in the end, whatever works best

AutoML: Algorithm Selection Performance Models

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Algorithm Selection



Types of Performance Models

- models for entire portfolios
 - models for individual algorithms
 - models that are somewhere in between (e.g. pairs of algorithms)
- for each of these, many different machine learning approaches are suitable

Models for Entire Portfolios

- predict the best algorithm in the portfolio (e.g. classifier to use)
 - alternatively: cluster in meta-feature space and assign best algorithm to each cluster
- optional (but important):
- attach a “weight” during learning (e.g. the difference between best and worst algorithm) to bias model towards the “important” datasets
 - special loss metric

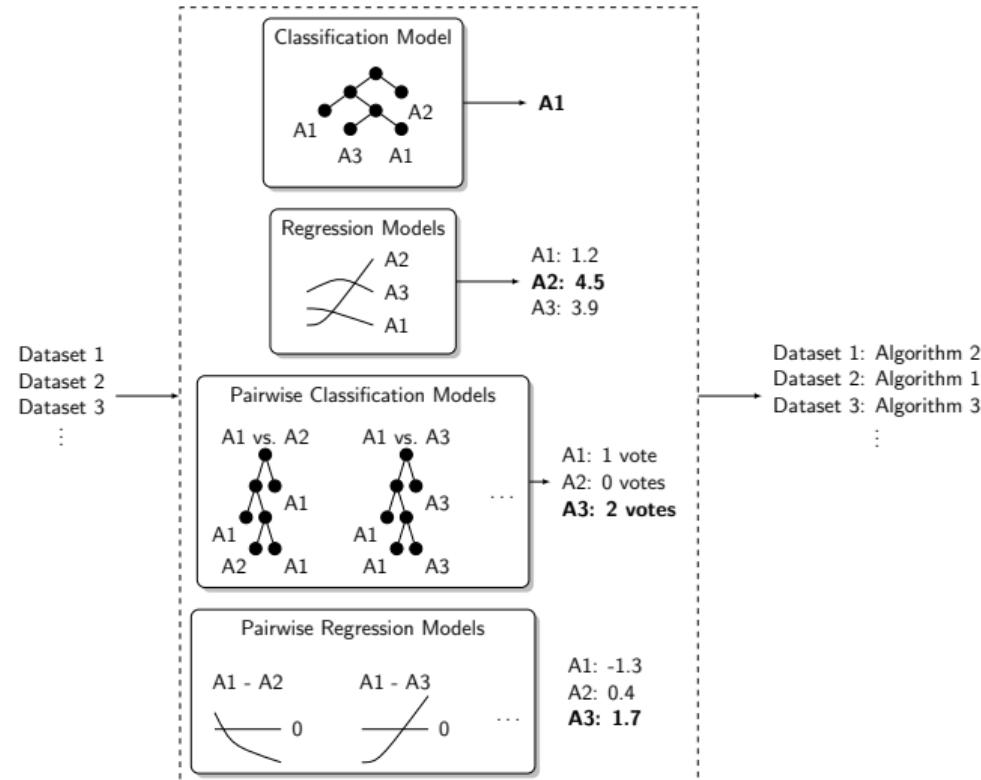
Models for Individual Algorithms

- predict the performance for each algorithm separately
- combine the predictions to choose the best one
- for example: predict accuracy, choose algorithm with highest predicted accuracy

Hybrid Models

- for example: consider pairs of algorithms to take relations between them into account
- for each pair of algorithms, learn model that predicts which one has better performance, or predicts performance difference
- . . . or collaborative filtering approaches

Types of Performance Models



Types of Predictions/Algorithm Selectors

- best algorithm (and its performance)
- n best algorithms ranked
- ensemble of n best algorithms

Time/Frequency of Prediction

- one-shot
 - ▶ select algorithm(s) once
 - ▶ want to process single dataset and choose the best approach
- multi-shot
 - ▶ continuously monitor dataset(s) features and/or performance
 - ▶ for example on data streams or to process sets of datasets

AutoML: Algorithm Selection

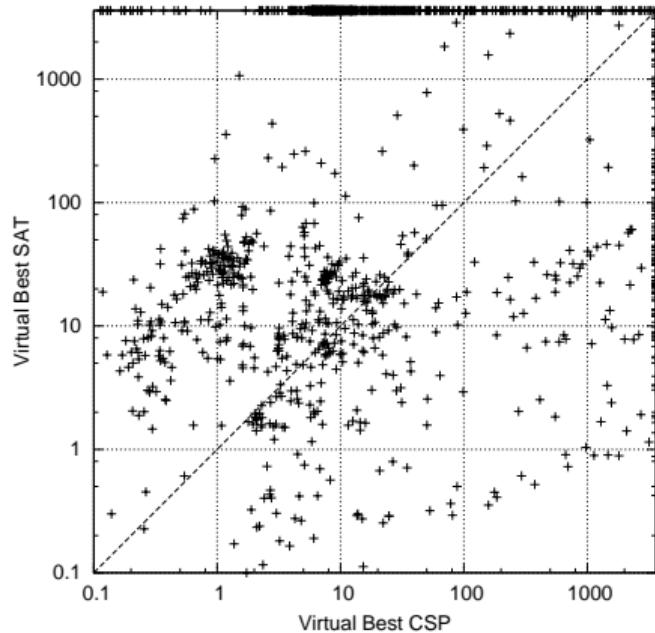
Bonus: Combinatorial Problems

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

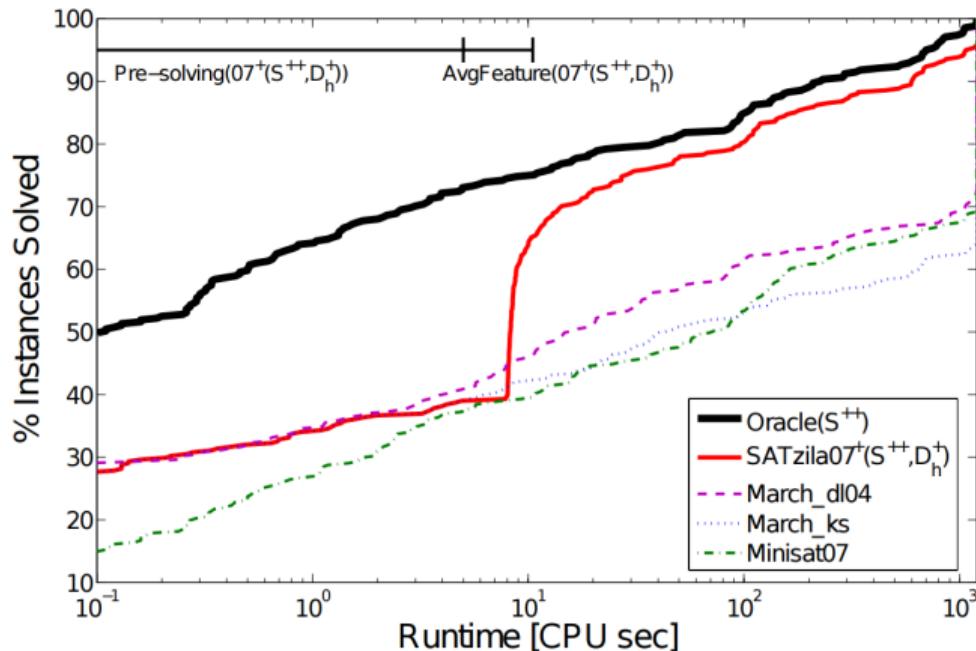
Motivation

- Algorithm Selection applied in many other domains
- success and performance improvements for combinatorial and optimization problems in AI dwarfs those in machine learning
- important application area of AI facilitating cross-disciplinary collaborations and advances

Motivation: Performance Differences [Barry et al. 2014] |



Motivation: Leveraging the Differences [Xu et al. 2008]



Algorithms [Huberman et al. 1997]

- constraint solvers
- search strategies
- modeling choices
- different types of consistency

Features

- number of variables, number of clauses/constraints/...
- ratios
- order of variables/values
- connectivity clause/constraints–variable graph or variable graph
- number of nodes/propagations within time limit
- estimate of search space size
- tightness of problem/constraints
- ...

Example System – SATzilla [Xu et al. 2008]

- portfolio of 7 SAT solvers, trained on 4811 problem instances
- syntactic (33) and probing features (15)
- ridge regression to predict log runtime for each solver, choose the solver with the best predicted performance
- later version uses random forests to predict better algorithm for each pair, aggregation through simple voting scheme
- pre-solving, feature computation time prediction, hierarchical model, selection of algorithms to include in portfolio based on overall performance
- won several competitions

Benchmark library – ASlib [Bischl et al. 2015]

- https://github.com/coseal/aslib_data
- SAT, CSP, QBF, ASP, MAXSAT, OR, ML...
- includes data used frequently in the literature that you may want to evaluate your approach on
- more scenarios in the pipeline
- <http://aslib.net>

Tools

[autofolio](https://bitbucket.org/mlindauer/autofolio/) <https://bitbucket.org/mlindauer/autofolio/>

[LLAMA](https://bitbucket.org/lkotthoff/llama) <https://bitbucket.org/lkotthoff/llama>

[SATzilla](http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/) <http://www.cs.ubc.ca/labs/beta/Projects/SATzilla/>

(Much) More Information [Kotthoff. 2014]

Comments? Suggestions? Corrections?
Let me know!

Algorithm Selection Literature Summary
Last update 21 November 2018

click headings to sort
click citations to expand
View in GitHub

citation	domain	features	predict what	predict how	predict when	portfolio	year
Langley 1983b; Langley 1983a Cohen et al. 1981	search planning	part performance, problem domain features, search statistics	algorithms control rules	hand-crafted and learned rules explanation-based rule construction	offline and online	dynamic	1983
Groß and De Jong 1992	planning	problem domain features, search statistics	control rules	probabilistic rule construction	online	dynamic	1992
Smith and Stoff 1992	software design	features of abstract representations	algorithms and data structures	simulated annealing	offline	static	1992
Aha 1992 Bradley 1983 Kemer et al. 1993	machine learning	instance features	algorithms	learned rules	offline	static	1992
Breiman 1980 Casali 1984 Tsang et al. 1995 Bauer 1986	machine learning differential equations	instance and algorithm features part performance, instance feature	algorithms	hand-crafted rules hand-crafted rules	offline	static	1983
Minton 1993b; Minton 1993a; Minton 1990 Castil 1984 Tsang et al. 1995 Bauer 1986	CSP	runtime performance	algorithms	hand-crafted and learned rules frame-based knowledge base	offline	dynamic	1993
Wesierska et al. 1996; Joshi et al. 1996	software design	instance features	algorithms and data structures	-	-	static	1996
Borod et al. 1996 Allen and Minton 1996 Sakakura et al. 1996 Habermann et al. 1997 Gomes and Selman 1997a Coste and Vensel 1997	CSP SAT, CSP pooling search graph colouring parallel search	search statistics switch algorithms?	algorithms	hand-crafted rules hand-crafted rules hand-crafted rules resource allocation statistical model	offline	static, static, order	1996
Pini 1997; Pini 1998 Latora and Lemmke 1998 Cesa et al. 1999 Hoover et al. 1999 Tennenholtz et al. 1999 Wilson et al. 2000 Betz and Por 2000	planning branch and bound vehicle routing problem	part performance peeling	resource allocation resource performance	Bayesian belief propagation, rule-based classifier, heaviest neighbour, neural net	offline	static	1997
Brasil and Soares 2000 Lagoudakis and Litzman 2000	classification order selection, sorting	instance features	ranking remaining cost for each sub-problems	distillation model MDP	offline	static	2000
Simo 2000 Phamnayek et al. 2000 Fukunaga 2000	CSP classification TSP	problem instance features, peeling	cost of solving problems algorithms	multinomial model 0 different classifiers	offline	static	2000
Soares and Brasil 2000 Gomes and Selman 2001	machine learning CSP instead integer preprocessing	instance features	ranking resource allocation algorithms	linear regression genetic algorithm nearest neighbor	offline	static, dynamic	2000
Epshteyn and Preud'homme 2001; Epshteyn et al. 2002; Epshteyn et al. 2003; Epshteyn and Petrosic 2001 Lagoudakis and Litzman 2001	COP DP/LB branching rules	variable characteristics instance features	weights, hand-crafted rules remaining cost for each sub-problem	weights, hand-crafted rules MDP	offline and online	dynamic	2001
Naujoks 2001 Horvitz et al. 2001	hybridization CSP	search statistics instance and instance generator features, search statistics	expected utility of algorithm parameters	refinement learning Bayesian model	offline and online	static	2001

<http://larskotthoff.github.io/assurvey/>

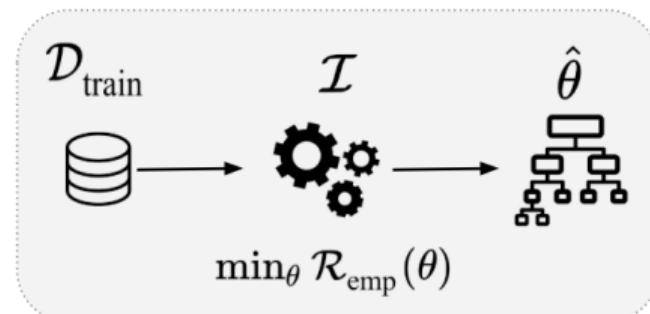
AutoML: Hyperparameter Optimization

Overview and Introduction

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Motivating Example I

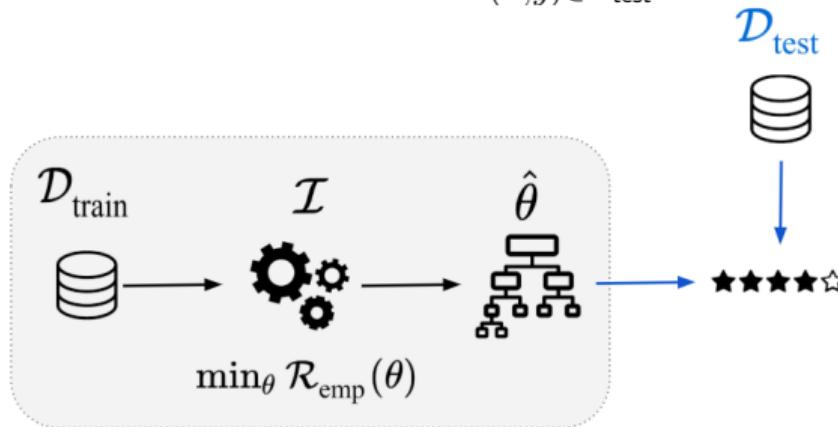
- Given a dataset, we want to train a classification tree.
- We feel that a maximum tree depth of 4 has worked out well for us previously, so we decide to set this hyperparameter to 4.
- The learner ("inducer") \mathcal{I} takes the input data, internally performs **empirical risk minimization**, and returns a fitted tree model $\hat{f}(\mathbf{x}) = f(\mathbf{x}, \hat{\theta})$ of at most depth $\lambda = 4$ that minimizes the empirical risk.



Motivating Example II

- We are **actually** interested in the **generalization performance** $GE(\hat{f})$ of the estimated model on new, previously unseen data.
- We estimate the generalization performance by evaluating the model \hat{f} on a test set $\mathcal{D}_{\text{test}}$:

$$\widehat{GE}_{\mathcal{D}_{\text{test}}}(\hat{f}) = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{test}}} L(y, \hat{f}(\mathbf{x}))$$



Motivating Example III

- But many ML algorithms are sensitive w.r.t. a good setting of their hyperparameters, and generalization performance might be bad, if we have chosen a suboptimal configuration:
 - ▶ The data may be too complex to be modeled by a tree of depth 4
 - ▶ The data may be much simpler than we thought, and a tree of depth 4 overfits
- ⇒ algorithmically try out different values for the tree depth. For each maximal depth λ , we have to train the model **to completion** and evaluate its performance on the test set.
- We choose the tree depth λ that is **optimal** w.r.t. the generalization error of the model.

Model Parameters vs. Hyperparameters I

It is critical to understand the difference between model parameters and hyperparameters.

Model parameters are optimized during training, typically via loss minimization. They are an **output** of the training. Examples:

- The splits and terminal node constants of a tree learner
- Coefficients θ of a linear model $f(\mathbf{x}) = \theta^\top \mathbf{x}$

Model Parameters vs. Hyperparameters II

In contrast, **hyperparameters** (HPs) are not decided during training. They must be specified before the training, they are an **input** of the training. Hyperparameters often control the complexity of a model, i.e., how flexible the model is. But they can in principle influence any structural property of a model or computational part of the training process.

Examples:

- Tree: The maximum depth of a tree
- k Nearest Neighbours: Number of neighbours k and distance measure
- Linear regression: Number and maximal order of interactions

Types of hyperparameters I

We summarize all hyperparameters we want to tune over in a vector $\lambda \in \Lambda$ of (possibly) mixed type. HPs can have different types:

- Real-valued parameters, e.g.:
 - ▶ Minimal error improvement in a tree to accept a split
 - ▶ Bandwidths of the kernel density estimates for Naive Bayes
- Integer parameters, e.g.:
 - ▶ Neighbourhood size k for k -NN
 - ▶ Minimum number of samples for a split in a random forest
- Categorical parameters, e.g.:
 - ▶ Which split criterion for classification trees?
 - ▶ Which distance measure for k -NN?

Hyperparameters are often **hierarchically dependent** on each other, e.g., if we use a kernel-density estimate for Naive Bayes, what is its width?

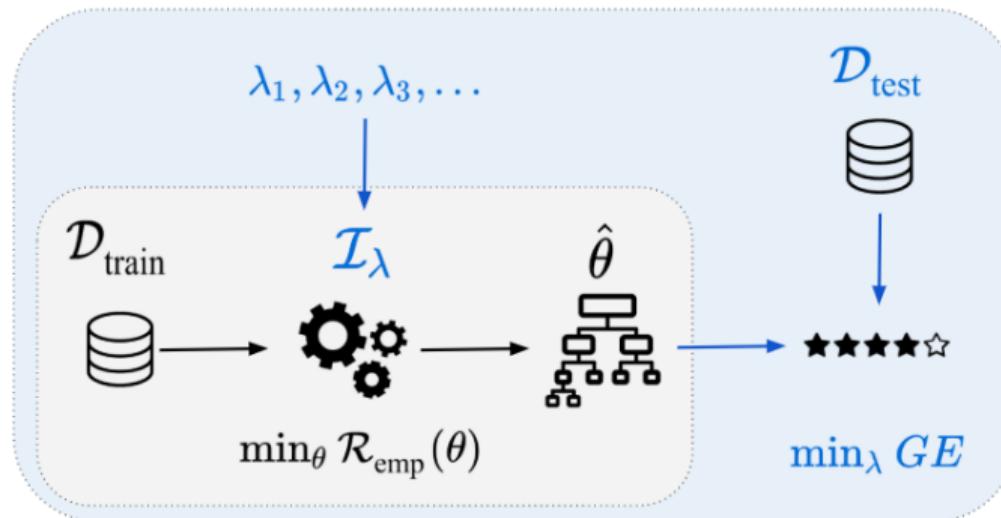
Tuning

Recall: **Hyperparameters** λ are parameters that are *inputs* to the training problem, in which a learner \mathcal{I} minimizes the empirical risk on a training data set in order to find optimal **model parameters** θ which define the fitted model \hat{f} .

(Hyperparameter) Tuning is the process of finding good model hyperparameters λ .

Tuning: A bi-level optimization problem I

We face a **bi-level** optimization problem: The well-known risk minimization problem to find \hat{f} is **nested** within the outer hyperparameter optimization (also called second-level problem):



Tuning: A bi-level optimization problem II

- For a learning algorithm \mathcal{I} (also inducer) with d hyperparameters, the **hyperparameter configuration space** is:

$$\Lambda = \Lambda_1 \times \Lambda_2 \times \dots \times \Lambda_d$$

where Λ_i is the domain of the i -th hyperparameter.

- The domains can be continuous, discrete or categorical.
- For practical reasons, the domain of a continuous or integer-valued hyperparameter is typically bounded.
- A vector in this configuration space is denoted as $\lambda \in \Lambda$.
- A learning algorithm \mathcal{I} takes a (training) dataset \mathcal{D} and a hyperparameter configuration $\lambda \in \Lambda$ and returns a trained model (through risk minimization).

$$\begin{aligned}\mathcal{I} : (\mathcal{X} \times \mathcal{Y})^n \times \Lambda &\rightarrow \mathcal{H} \\ (\mathcal{D}, \lambda) &\mapsto \mathcal{I}(\mathcal{D}, \lambda) = \hat{f}_{\mathcal{D}, \lambda}\end{aligned}$$

Tuning: A bi-level optimization problem III

We formally state the nested hyperparameter tuning problem as:

$$\min_{\lambda \in \Lambda} c(\lambda) = \widehat{GE}_{\mathcal{D}_{\text{test}}} (\mathcal{I}(\mathcal{D}_{\text{train}}, \lambda))$$

- The learner $\mathcal{I}(\mathcal{D}_{\text{train}}, \lambda)$ takes a training dataset as well as hyperparameter settings Λ (e.g. the maximal depth of a classification tree) as an input.
- $\mathcal{I}(\mathcal{D}_{\text{train}}, \lambda)$ performs empirical risk minimization on the training data and returns the optimal model \hat{f} for the given hyperparameters.
- Note that for the estimation of the generalization error, more sophisticated resampling strategies like cross-validation can be used.

Tuning: A bi-level optimization problem IV

The components of a tuning problem are:

- The dataset
- The learner (possibly: several competing learners?) that is tuned
- The learner's hyperparameters and their respective regions-of-interest over which we optimize
- The performance measure, as determined by the application.
Not necessarily identical to the loss function that defines the risk minimization problem for the learner!
- A (resampling) procedure for estimating the predictive performance according to the performance measure.

Why is tuning so hard?

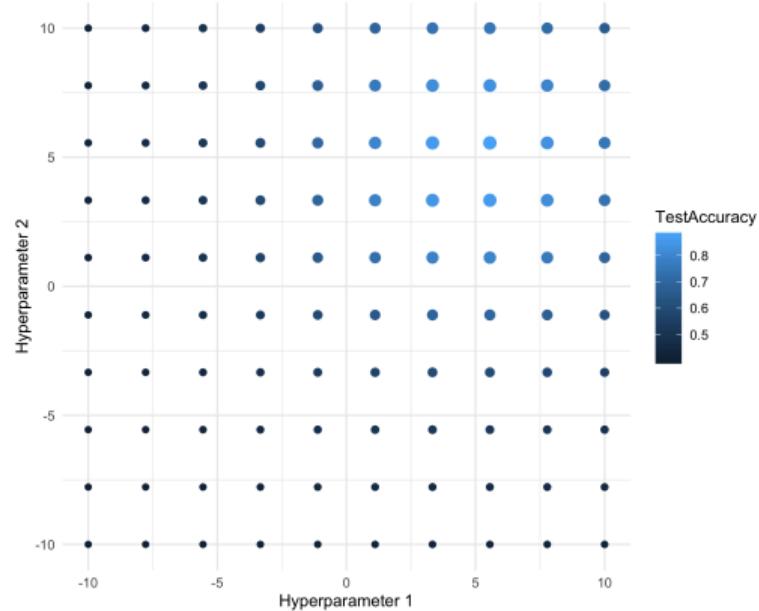
- Tuning is derivative-free (black box problem): It is usually impossible to compute derivatives of the objective (i.e., the resampled performance measure) that we optimize with regard to the HPs. All we can do is evaluate the performance for a given hyperparameter configuration.
- Every evaluation requires one or multiple train and predict steps of the learner. I.e., every evaluation is very **expensive**.
- Even worse: the answer we get from that evaluation is **not exact, but stochastic** in most settings, as we use resampling (and often stochastic learners).
- Categorical and dependent hyperparameters aggravate our difficulties: the space of hyperparameters we optimize over has a non-metric, complicated structure.

AutoML: Hyperparameter Optimization Grid and Random Search

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Grid search I

- Simple technique which is still quite popular, tries all HP combinations on a multi-dimensional discretized grid
- For each hyperparameter a finite set of candidates is predefined
- Then, we simply search all possible combinations in arbitrary order



Grid search over 10x10 points

Grid search II

Advantages

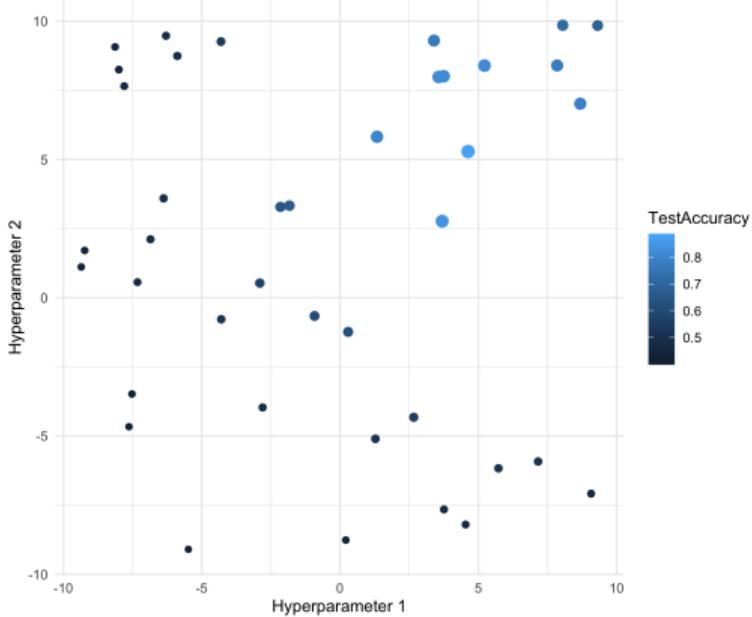
- Very easy to implement
- All parameter types possible
- Parallelizing computation is trivial

Disadvantages

- Scales badly: Combinatorial explosion
- Inefficient: Searches large irrelevant areas
- Low resolution in each dimension
- Arbitrary: Which values / discretization?

Random search I

- Small variation of grid search
- Uniformly sample from the region-of-interest



Random search over 100 points

Random search II

Advantages

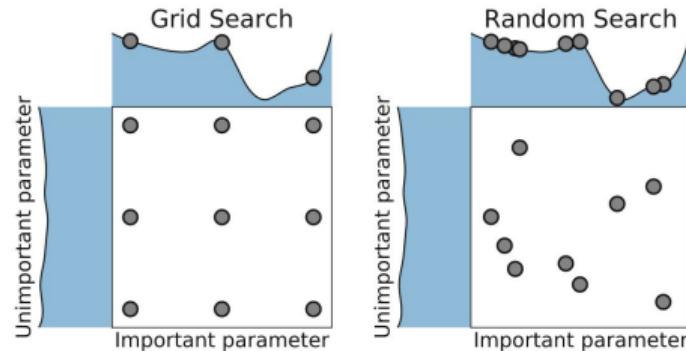
- Like grid search: Very easy to implement, all parameter types possible, trivial parallelization
- Anytime algorithm: Can stop the search whenever our budget for computation is exhausted, or continue until we reach our performance goal.
- No discretization: each individual parameter is tried with a different value every time

Disadvantages

- Inefficient: many evaluations in areas with low likelihood for improvement
- Scales badly: high dimensional hyperparameter spaces need *lots* of samples to cover.

Grid search vs. Random search

- With a tuning budget of T only $T^{\frac{1}{d}}$ unique hyperparameter values are explored for each $\lambda_1, \dots, \lambda_d$ in a grid search.
- Random search will (most likely) see T different values for each hyperparameter.
- Grid search can be disadvantageous if some hyperparameters have little or no influence on the model.



Comparison of grid search and random search.
[Hutter et al. 2019]

AutoML: Hyperparameter Optimization Evolutionary Algorithms

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Evolutionary algorithms

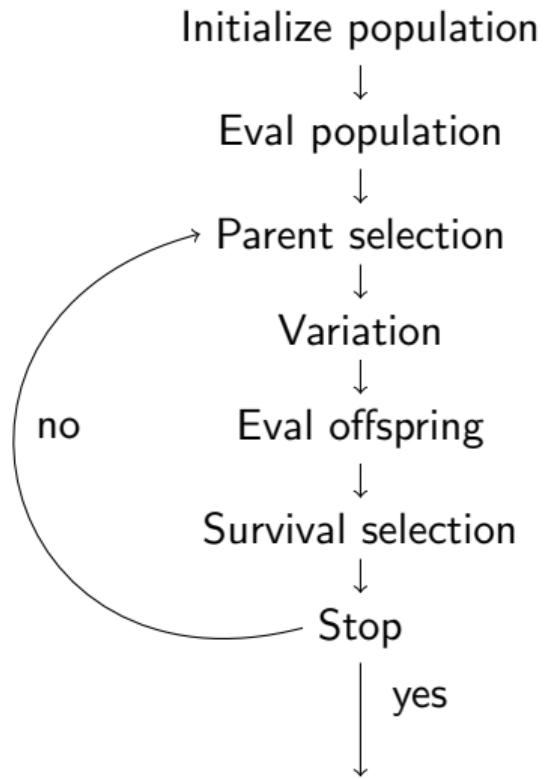
Evolutionary algorithms (EA) are a class of stochastic, metaheuristic optimization techniques whose mode of operation is inspired by the evolution of natural organisms.

History of evolutionary algorithms:

- **Genetic algorithms**: Use binary problem representation, therefore closest to the biological model of evolution.
- **Evolution strategies**: Use direct problem representation, e.g., vector of real numbers.
- **Genetic programming**: Create structures that convert an input into a fixed output (e.g. computer programs); solution candidates are represented as trees.
- **Evolutionary programming**: Similar to GP, but solution candidates are not represented by trees, but by finite state machines.

The boundaries between the terms become increasingly blurred and are often used synonymously.

Structure of an evolutionary algorithm



Notation and Terminology

Symbols	EA Terminology
$\lambda \in \Lambda$	Chromosome of an individual
λ_i	i -th gene of chromosome
\mathcal{P}	Population and size
$\mu = \mathcal{P} $	Number of generated offsprings
$c : \Lambda \rightarrow \mathbb{R}$	Fitness function

$$c(\boldsymbol{\lambda}) = \widehat{GE}_{\mathcal{D}_{\text{test}}}(\mathcal{I}(\mathcal{D}_{\text{train}}, \boldsymbol{\lambda}))$$

Notation clash:

- In EAs the objective function is often denoted $f(x)$.
- As these symbols are used for ML already we use $c(\boldsymbol{\lambda})$ and $\boldsymbol{\lambda}$ instead of f and x .
- Be careful: The offspring size λ is different from the candidate $\boldsymbol{\lambda}$ (bold symbol!).

Step 1: Initialize population

- An evolutionary algorithm is started by generating an initial population $\mathcal{P} = \{\boldsymbol{\lambda}^{(1)}, \dots, \boldsymbol{\lambda}^{(\mu)}\}$.
- Usually we sample this uniformly at random.
- We could introduce problem prior knowledge via a smarter init procedure.
- This population is evaluated, i.e., the objective function is computed for every individual in the initial population.
- The initialization can have a large influence on the quality of the found solution, so many EAs employ *restarts* with new randomly generated populations.

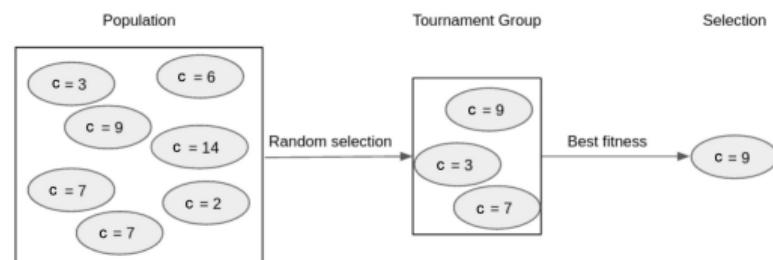
Step 2: Parent selection I

In the first step of an iteration, λ parents are chosen, who create offspring in the next step.

Possibilities for selection of parents:

- **Neutral selection:** choose individual with a probability $1/\mu$.
- **Fitness-proportional selection:** draw individuals with probability proportional to their fitness.

- **Tournament Selection:** randomly select k individuals for a "Tournament Group". Of the drawn individuals, the best one (with the highest fitness value) is then chosen. Procedure is performed λ -times.



Step 3: Variation

New individuals are now generated from the parent population. This is done by

- Recombination/Crossover: combine two parents into one offspring.
- Mutation: (locally) change an individual.

Sometimes only one operation is performed.

Recombination for numeric representations

Two individuals $\lambda, \tilde{\lambda} \in \mathbb{R}^n$ in numerical representation can be recombined as follows:

- **Uniform crossover:** choose gene i with probability p of 1st parent and probability $1 - p$ of 2nd parent.
- **Intermediate recombination:** new individual is created from the mean value of two parents $\frac{1}{2}(\lambda + \tilde{\lambda})$
- **Simulated Binary Crossover (SBX):** generate **two offspring** based on

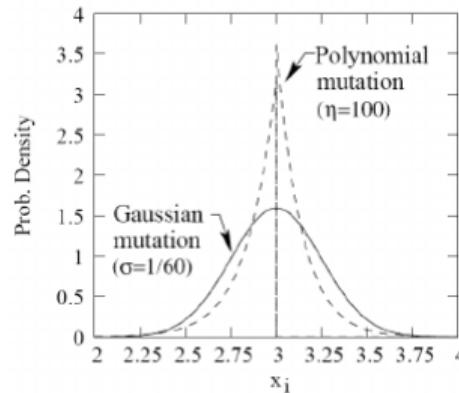
$$\bar{\lambda} \pm \frac{1}{2}\beta(\tilde{\lambda} - \lambda)$$

with $\bar{\lambda} = \frac{1}{2}(\lambda + \tilde{\lambda})$ and β randomly sampled from a certain distribution.

Mutation for numeric representations [K. Deb and D. Deb. 2014]

Mutation: individuals are changed, for example for $\lambda \in \mathbb{R}^n$

- **Uniform mutation:** choose a random gene λ_i and replace it with a value uniformly distributed (within the feasible range).
- **Gauss mutation:** $\tilde{\lambda} = \lambda \pm \sigma \mathcal{N}(0, I)$
- **Polynomial mutation:** polynomial distribution instead of normal distribution



Recombination for bit strings

Two individuals $\lambda, \tilde{\lambda} \in \{0, 1\}^n$ encoded as bit strings can be recombined as follows:

- **1-point crossover:** select crossover $k \in \{1, \dots, n - 1\}$ randomly and select the first k bits from 1st parent, the last $n - k$ bits from 2nd parent.

$$\begin{array}{ccc|c} 1 & 1 & 1 \\ 0 & 0 & 0 \\ \hline 0 & 1 & \Rightarrow & 1 \\ 1 & 1 & & 1 \\ 1 & 0 & & 0 \end{array}$$

- **Uniform crossover:** select bit i with probability p of 1st parent and $1 - p$ of 2nd parent.

$$\begin{array}{ccc|c} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 1 & \Rightarrow & 1 \\ 0 & 1 & & 1 \\ 1 & 0 & & 1 \end{array}$$

Mutation for bit strings

An individual $\lambda \in \{0, 1\}^n$ encoded as a bit string can be mutated as follows:

- **Bitflip:** for each index $k \in \{1, \dots, n\}$: bit k is flipped with probability $p \in (0, 1)$.

1	0
0	0
0	\Rightarrow 0
0	1
1	1

Step 4: Survival selection

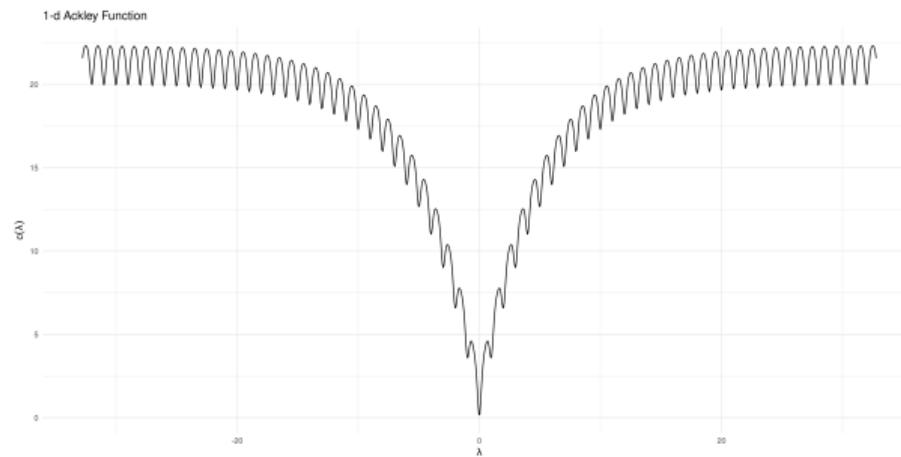
Now individuals are chosen who survive. Two common strategies are:

- **(μ, λ) -selection:** we select from the λ descendants the μ best ($\lambda \geq \mu$ necessary). **But:** best overall individual can get lost!
- **$(\mu + \lambda)$ -selection:** μ parents and λ offspring are lumped together and the μ best individuals are chosen. Best individual safely survives.

Example of an evolutionary algorithm I

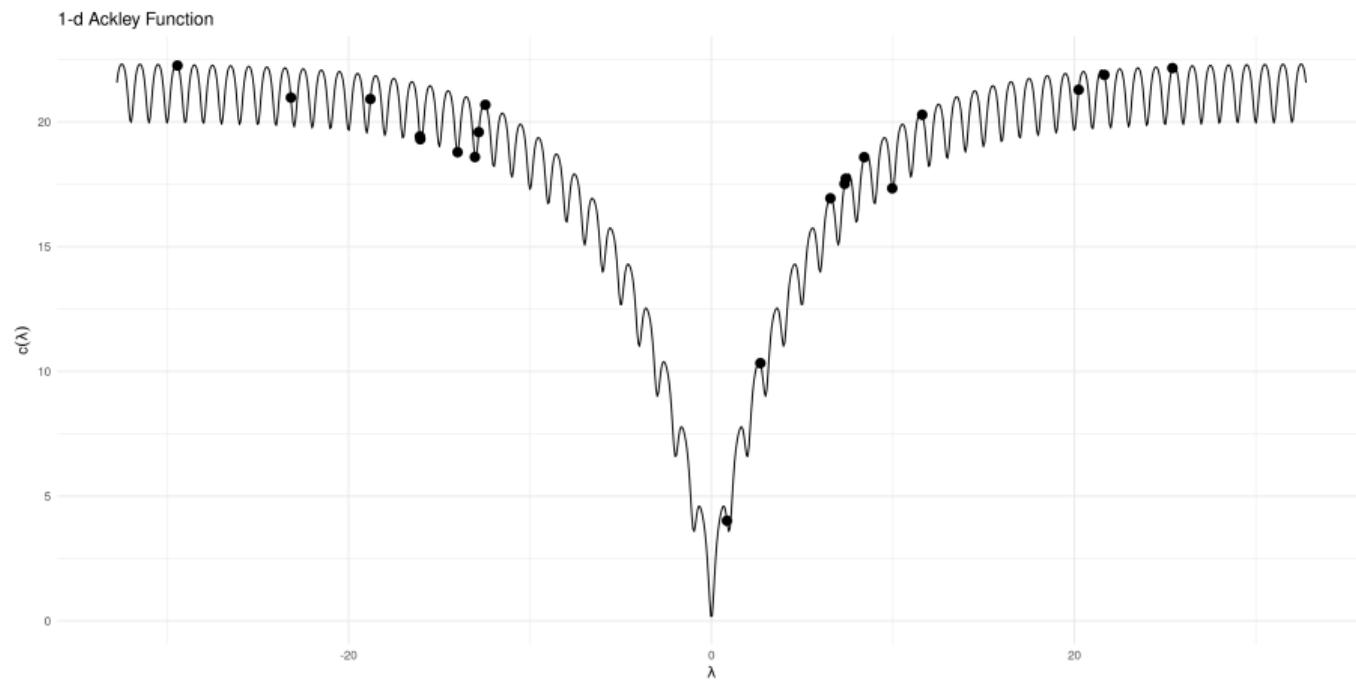
In the following, a (simple) EA is shown on the 1-dim Ackley function, optimized on $[-30, 30]$.

Usually for the optimization of a function $c : \mathbb{R}^n \rightarrow \mathbb{R}$ individuals are coded as real vectors $\lambda \in \mathbb{R}^n$, so here we use simply one real number as chromosome.



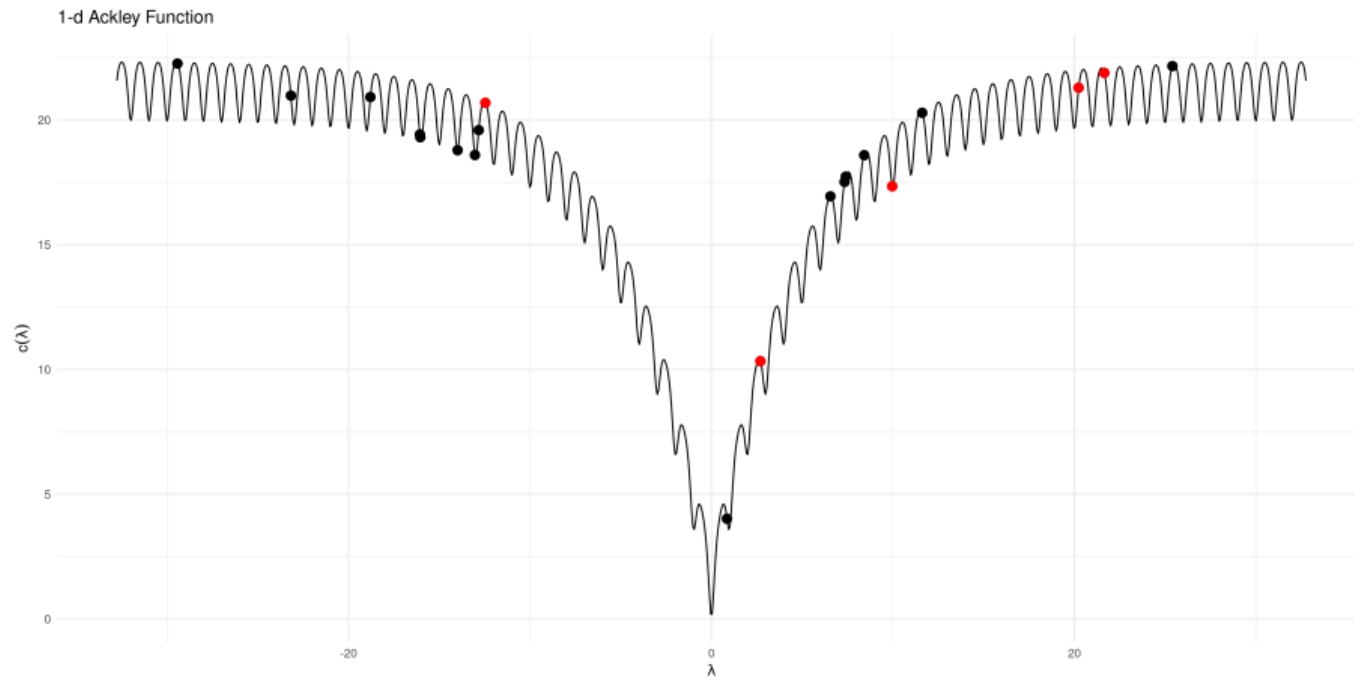
Example of an evolutionary algorithm II

Randomly init population with size $\mu = 20$.



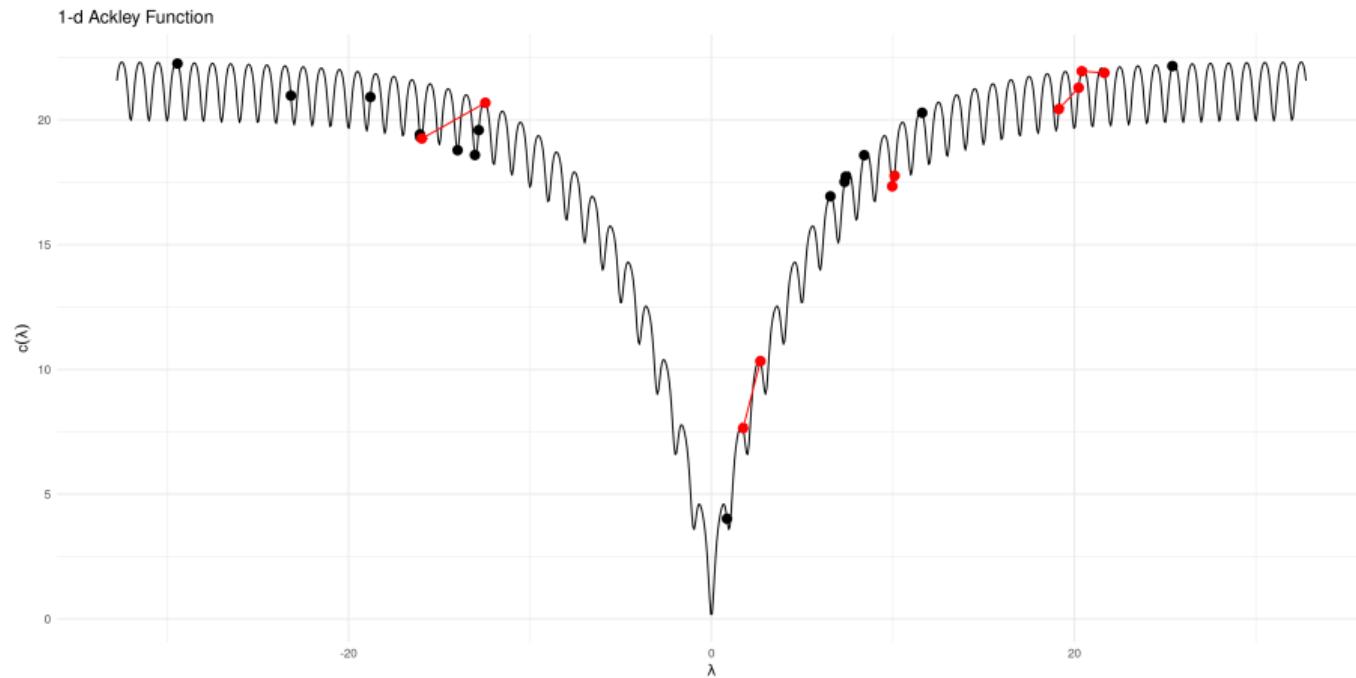
Example of an evolutionary algorithm III

We choose $\lambda = 5$ offspring by neutral selection (red individuals).



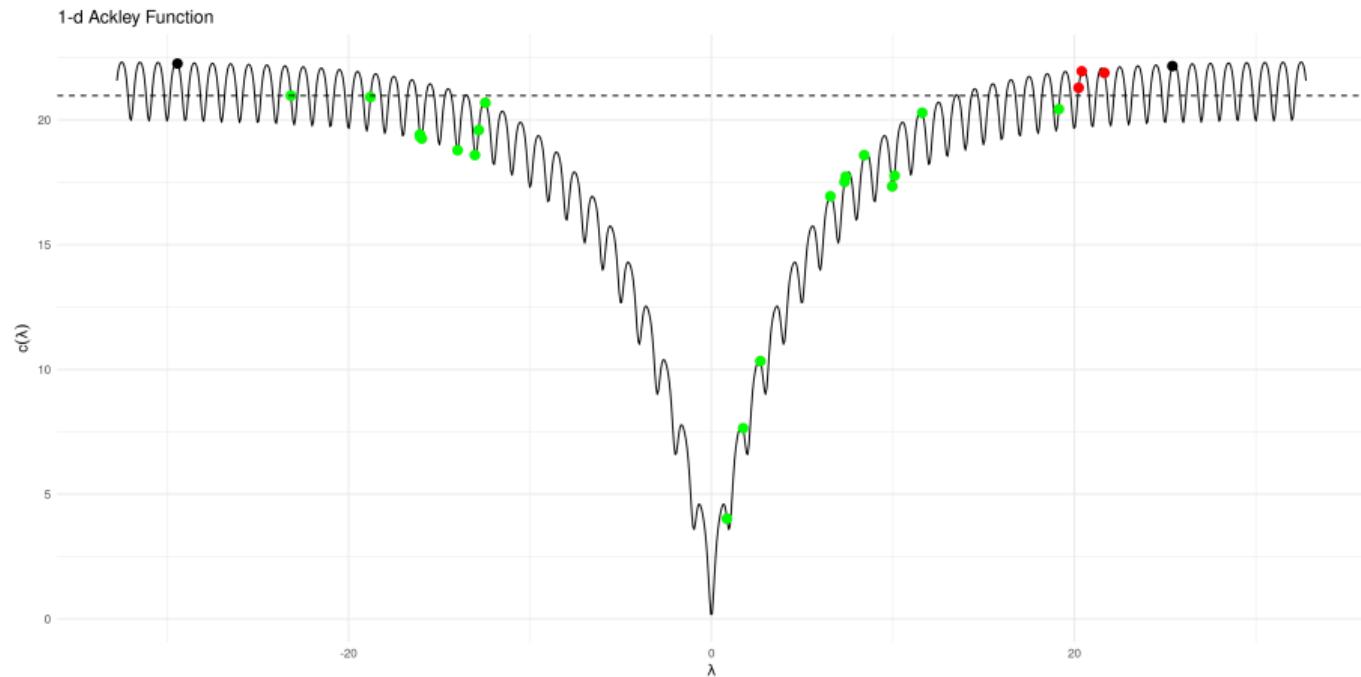
Example of an evolutionary algorithm IV

We use a Gauss mutation with $\sigma = 2$ and do not apply a recombination.



Example of an evolutionary algorithm V

We use a $(\mu + \lambda)$ selection. The selected individuals are green.



Evolutionary Algorithms

Advantages

- Conceptually simple, yet powerful enough to solve complex problems (including HPO)
- All parameter types possible in general
- Highly parallelizable (depends on λ)
- Allows customization via specific variation operators

Disadvantages

- Less theory available (for realistic, complex EAs)
- Can be hard to get balance between exploration and exploitation right
- Can have quite a few control parameters, hard to set them correctly
- Customization necessary for complex problems
- Not perfectly suited for expensive problems like HPO, as quite a higher number of evaluations is usually needed for appropriate convergence / progress

AutoML: Hyperparameter Optimization

Example and Practical Hints

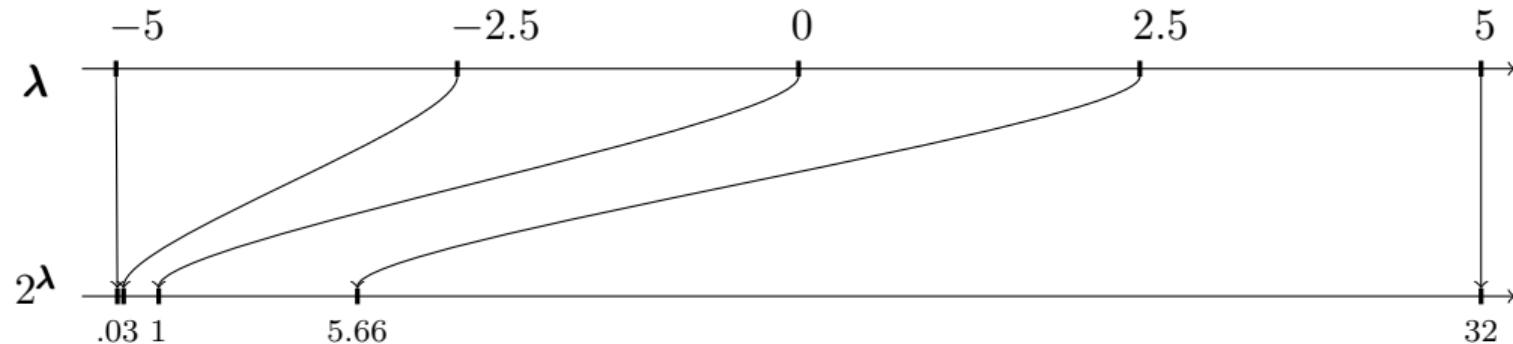
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Scaling and Ranges I

- Knowledge about hyperparameters can help to guide the optimization
- E.g., it can be beneficial to optimize hyperparameters on a non-uniform scale.

Example: regularization hyperparameter on log-scale

- Many ML algorithms have non-neg. hyperparameters (e.g. regularization constant), for which it can make sense to try out very small and very large values during tuning
- Usual trick: put on log-scale: C of SVM: $[2^{-15}, 2^{15}] = [0.00003, 32768]$



Scaling and Ranges II

- Similar to the scale, e.g., linear or logarithmic, upper and lower bounds for hyperparameters have to be specified as many optimizers require them
- Setting these correctly usually requires deeper knowledge about the inner workings of the ML method and a lot of practical experience
- Furthermore, if $\hat{\lambda}$ is close to the border of Λ the ranges should be increased (or a different scale should be selected), but many algorithms do not do this automatically
- Meta-Learning can help to decide which hyperparameters should be tuned in which ranges

Default Heuristics

- Instead of using static defaults, we sometimes can compute hyperparameter defaults heuristically, based on simple data characteristics.
- A lot faster than tuning and sometimes can work well, although not many guarantees exist and it is often unclear how well this works across many different data scenarios
- Well-known example: Number of random features to consider for splitting in random forest: $mtry = \sqrt{p}$, where p is total number of features.
- For the RBF-SVM a data dependent default for the γ parameter (inverse kernel width) can be computed by using the (inverse of the) median of the pairwise distances $\|\mathbf{x} - \tilde{\mathbf{x}}\|$ between data points (or a smaller random subset for efficiency)

Tuning Example - Setup

We want to train a *spam detector* on the popular Spam dataset¹.

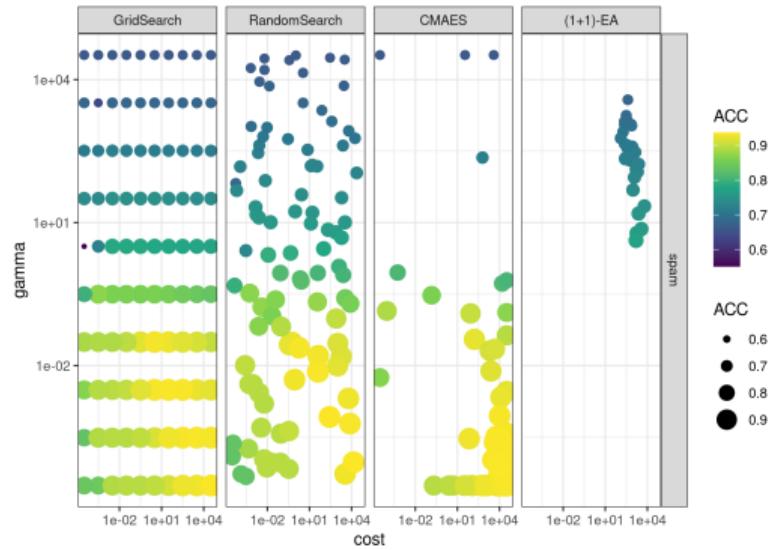
- The learning algorithm is a support vector machine (SVM) with RBF kernel.
- The hyperparameters we optimize are
 - ▶ Cost parameter $\text{cost} \in [2^{-15}, 2^{15}]$.
 - ▶ Kernel parameter $\gamma \in [2^{-15}, 2^{15}]$.
- We compare four different optimizers
 - ▶ Random search
 - ▶ Grid search
 - ▶ A $(1 + 1)$ -selection EA and Gauss mutation with $\sigma = 1$.
 - ▶ *CMAES* - efficient EA that generates offspring from a multivariate Gaussian
- We use a 5-fold CV for tuning on the inside, to optimize accuracy (ACC) and 10-fold on the outside for nested CV
- All methods are allowed a budget of 100 evaluations

¹<https://archive.ics.uci.edu/ml/datasets/spambase>

Tuning Example

We notice here:

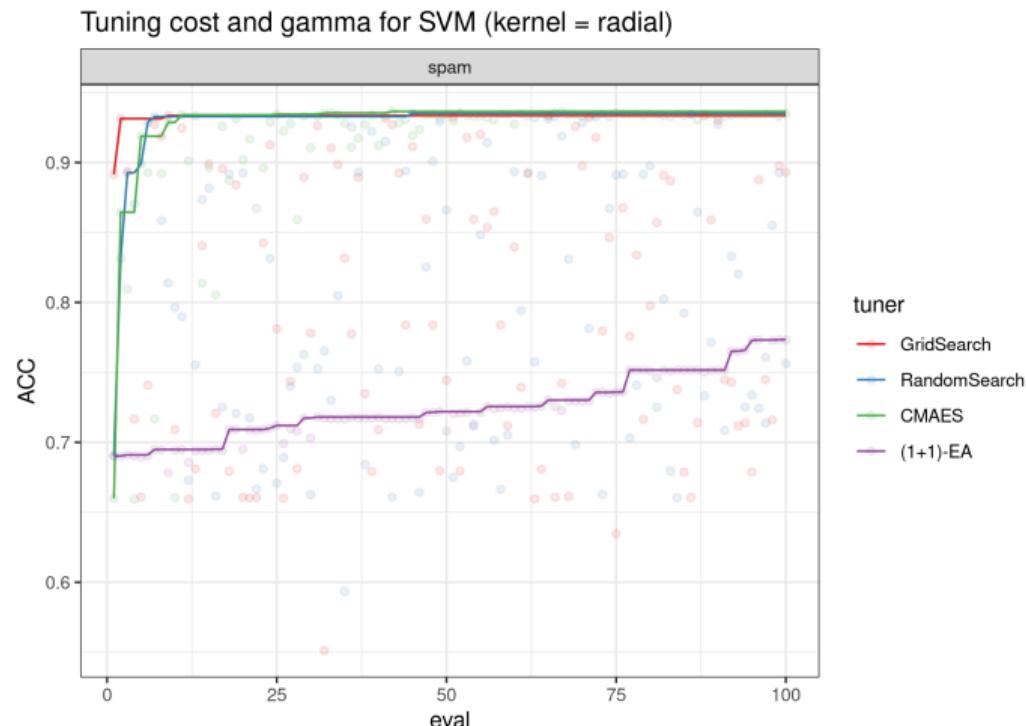
- Both *Grid search* and *random search* have many evaluations in regions with bad performance ($\gamma > 1$)
- *CMAES* only explores a small region
- *(1+1)-EA* does not converge, we probably set its control parameters in a suboptimal manner
- May we should increase ranges?
- Such a visual analysis is a lot harder for more than 2 hyperparameters



Tuning Example cont.

The *optimization trace* shows the best obtained performance until a given time point.

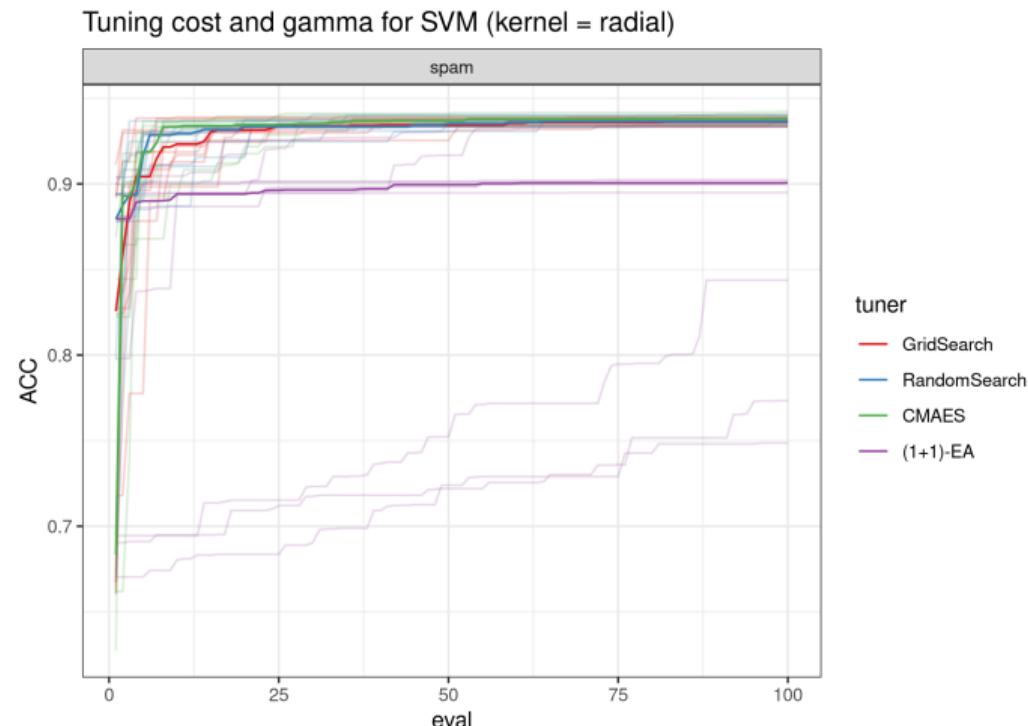
- For *random search* and *grid search* the chronological order of point evaluation is somewhat arbitrary
- The curve shows the performance on the tuning validation (*inner resampling*) on a single fold



Tuning Example cont.

The *optimization trace* shows the best obtained performance until a given time point.

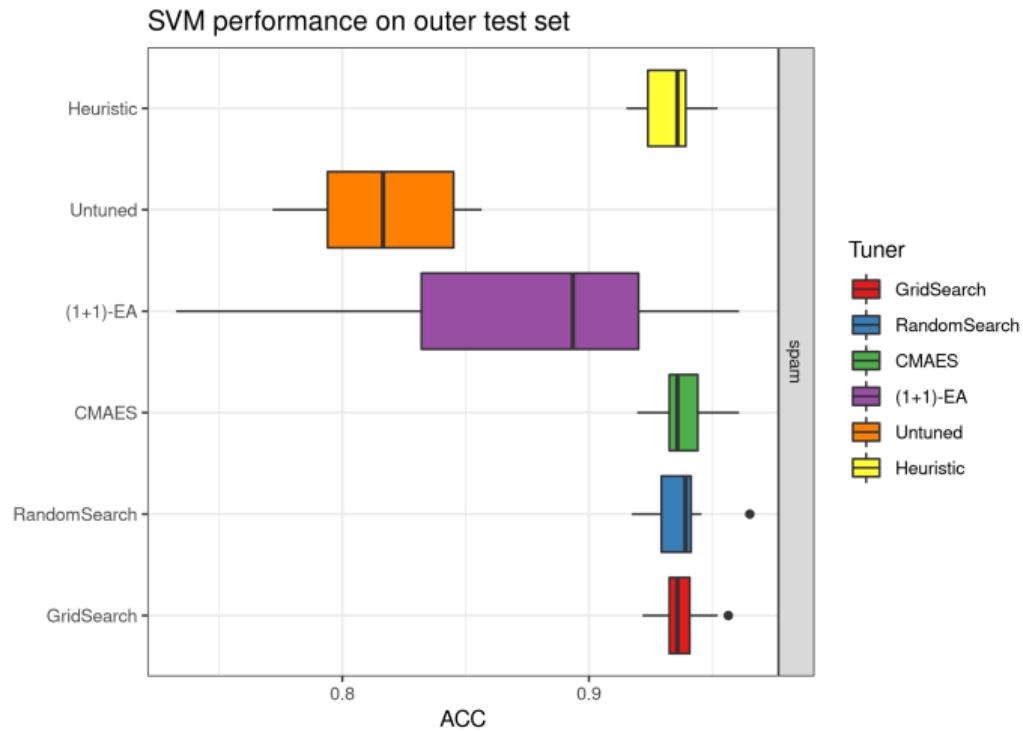
- The outer 10-fold CV gives us 10 optimization curves.
- The median at each time point gives us an estimate of the average optimization progress.



Tuning Example: Validation

Distribution of the ACC-values on *outer test sets* with a 10-fold CV.

- We compare with SVM in (unreasonable) defaults ($\text{cost} = 1, \gamma = 1$) and the previously discussed heuristic with $\text{cost} = 1$
- We abstained from proper statistical testing here
- The performance is somewhat lower than indicated by the results on the inner resampling



Challenges and Final Comments I

- ① Getting it right which hyperparameters to tune, in what ranges, and where defaults and where heuristics might be ok.
- ② Choosing and balancing out budget for tuning and inner and outer resampling.
- ③ Dealing with multi criteria-situations, where multiple performance metrics are of interest
- ④ Dealing with parallelization and time-heterogeneity
- ⑤ Ensuring the computational stability of the tuning process and dealing with crashes

Challenges and Final Comments II

- ⑥ Post-Hoc analysis of all obtained tuning results
- ⑦ Exploiting the multi-fidelity property of ML training (suppress bad configurations early without investing too much time)
- ⑧ Including preprocessing and full pipelining into the tuning process, and dealing with complex hierarchical spaces

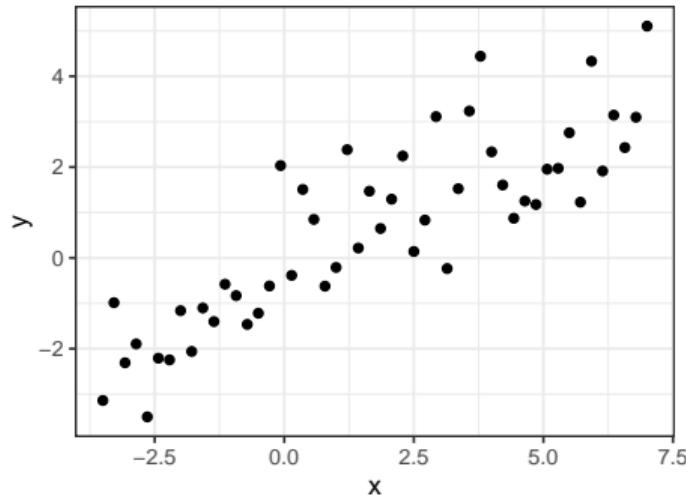
AutoML: Gaussian Processes

The Bayesian Linear Model

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Review: The Bayesian Linear Model I

Let $\mathcal{D}_{\text{train}} = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})\}$ be a training set of i.i.d. observations from some unknown distribution.



Let $\mathbf{y} = (y^{(1)}, \dots, y^{(n)})^\top$ and $\mathbf{X} \in \mathbb{R}^{n \times p}$ be the design matrix where the i-th row contains vector $\mathbf{x}^{(i)}$.

Review: The Bayesian Linear Model II

The linear regression model is defined as

$$y = f(\mathbf{x}) + \epsilon = \boldsymbol{\theta}^\top \mathbf{x} + \epsilon$$

or on the data:

$$y^{(i)} = f(\mathbf{x}^{(i)}) + \epsilon^{(i)} = \boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \epsilon^{(i)}, \text{ for all } i \in \{1, \dots, n\}.$$

We now assume (from a Bayesian perspective) that also our parameter vector $\boldsymbol{\theta}$ is stochastic and follows a distribution.

The observed values $y^{(i)}$ differ from the function values $f(\mathbf{x}^{(i)})$ by some additive noise, which is assumed to be i.i.d. Gaussian

$$\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$$

and independent of \mathbf{x} and $\boldsymbol{\theta}$.

Review: The Bayesian Linear Model III

- Let us assume we have **prior beliefs** about the parameter θ that are represented in a prior distribution $\theta \sim \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I}_p)$.
- Whenever data points are observed, we update the parameters' prior distribution according to Bayes' rule

$$\underbrace{p(\theta | \mathbf{X}, \mathbf{y})}_{\text{posterior}} = \frac{\overbrace{p(\mathbf{y} | \mathbf{X}, \theta)}^{\text{likelihood}} \overbrace{q(\theta)}^{\text{prior}}}{\underbrace{p(\mathbf{y} | \mathbf{X})}_{\text{marginal}}}.$$

Review: The Bayesian Linear Model IV

The posterior distribution of the parameter θ is again normal distributed (the Gaussian family is self-conjugate):

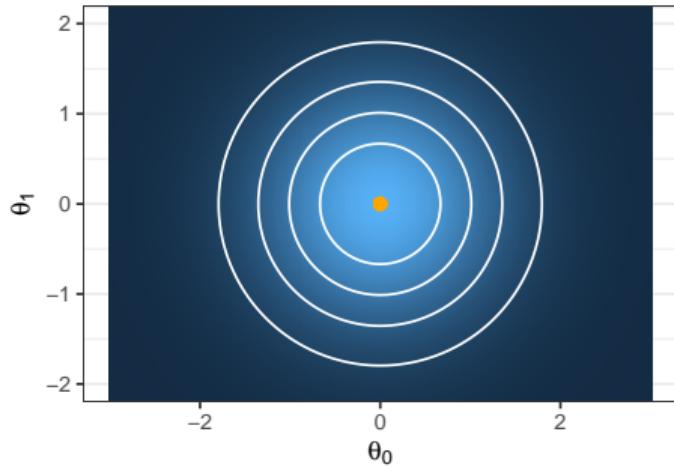
$$\theta | \mathbf{X}, \mathbf{y} \sim \mathcal{N}(\sigma^{-2} \mathbf{A}^{-1} \mathbf{X}^\top \mathbf{y}, \mathbf{A}^{-1}), \text{ where } \mathbf{A} := \sigma^{-2} \mathbf{X}^\top \mathbf{X} + \frac{1}{\tau^2} \mathbf{I}_p.$$

Note: If the posterior distributions $p(\theta | \mathbf{X}, \mathbf{y})$ are in the same probability distribution family as the prior $q(\theta)$, the prior and posterior are then called **conjugate distributions**, and the prior is called a **conjugate prior** for the likelihood function $p(\mathbf{y} | \mathbf{X}, \theta)$.

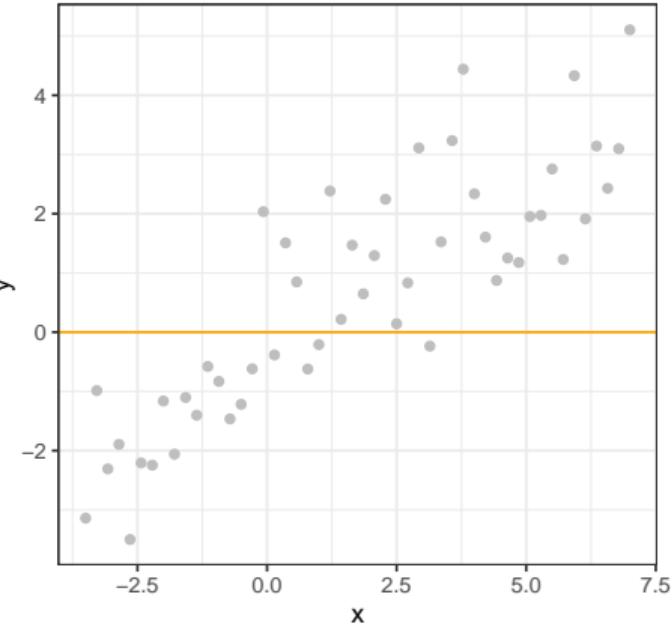
Note: The Gaussian family is **self-conjugate** with respect to a Gaussian likelihood function: choosing a Gaussian prior for a Gaussian likelihood ensures that the posterior is also Gaussian.

Review: The Bayesian Linear Model V

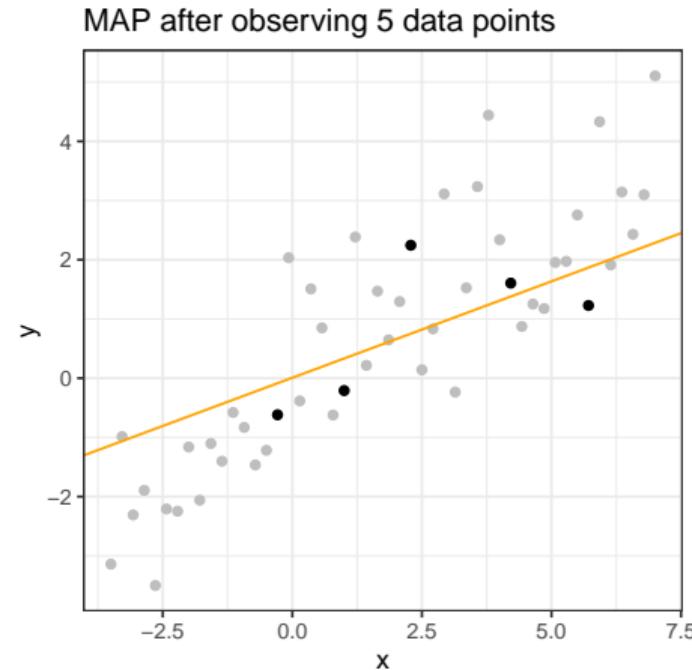
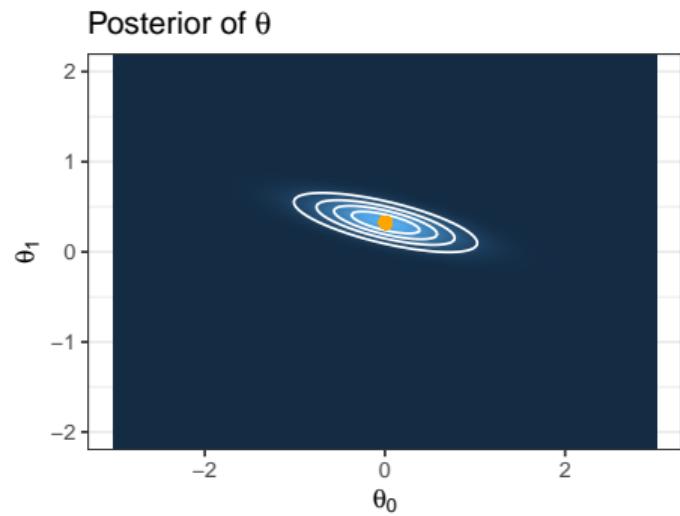
Prior $\theta \sim N(0, 1)$



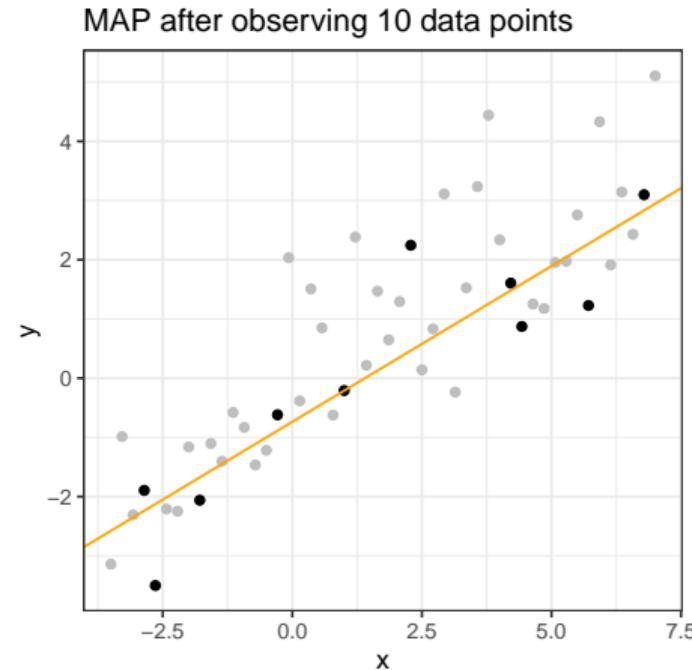
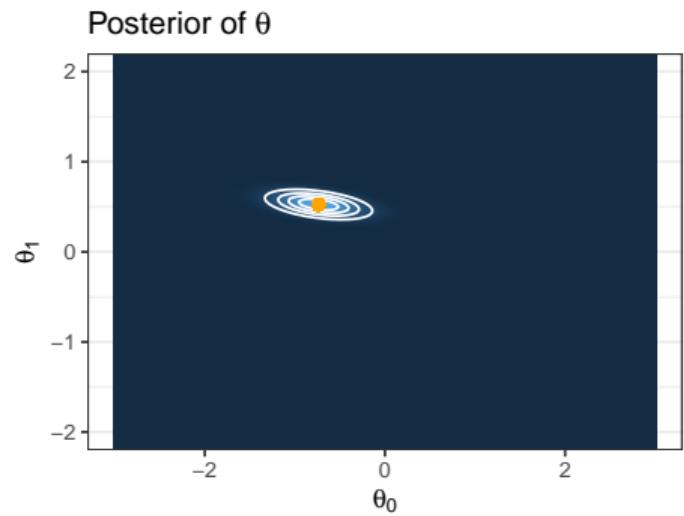
No data points observed



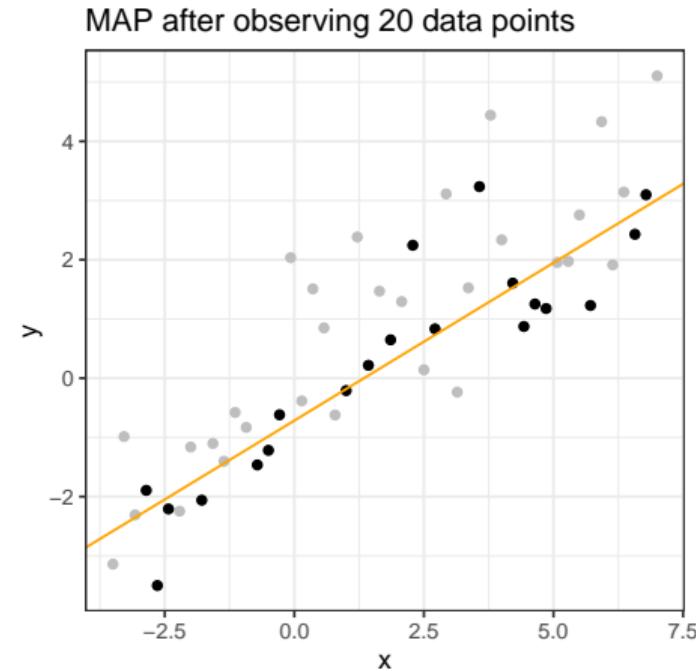
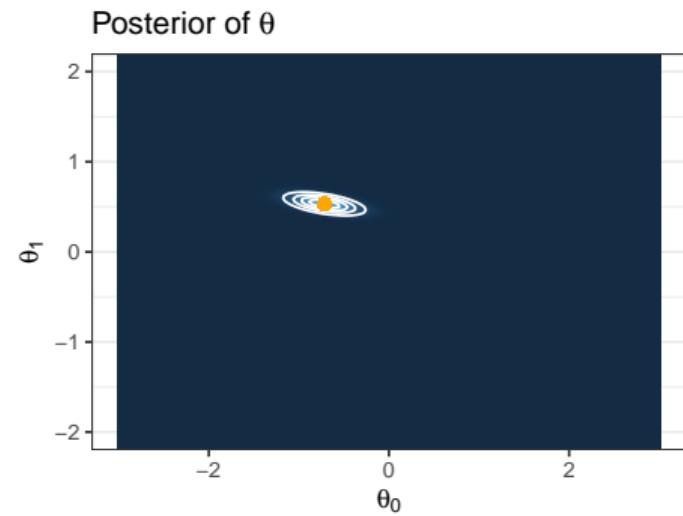
Review: The Bayesian Linear Model VI



Review: The Bayesian Linear Model VII



Review: The Bayesian Linear Model VIII



Review: The Bayesian Linear Model IX

Theorem:

- For a Gaussian prior on $\theta \sim \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I}_p)$ and a Gaussian likelihood $y | \mathbf{X}, \theta \sim \mathcal{N}(\mathbf{X}^\top \theta, \sigma^2 \mathbf{I}_n)$, the resulting posterior is Gaussian: $\mathcal{N}(\sigma^{-2} \mathbf{A}^{-1} \mathbf{X}^\top \mathbf{y}, \mathbf{A}^{-1})$, with $\mathbf{A} := \sigma^{-2} \mathbf{X}^\top \mathbf{X} + \frac{1}{\tau^2} \mathbf{I}_p$.

Proof:

Plugging in Bayes' rule and multiplying out yields

$$\begin{aligned} p(\theta | \mathbf{X}, \mathbf{y}) &\propto p(\mathbf{y} | \mathbf{X}, \theta) q(\theta) \propto \exp \left[-\frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{X}\theta)^\top (\mathbf{y} - \mathbf{X}\theta) - \frac{1}{2\tau^2} \theta^\top \theta \right] \\ &= \exp \left[-\frac{1}{2} \left(\underbrace{\sigma^{-2} \mathbf{y}^\top \mathbf{y}}_{\text{doesn't depend on } \theta} - 2\sigma^{-2} \mathbf{y}^\top \mathbf{X}\theta + \sigma^{-2} \theta^\top \mathbf{X}^\top \mathbf{X}\theta + \tau^{-2} \theta^\top \theta \right) \right] \\ &\propto \exp \left[-\frac{1}{2} \left(\sigma^{-2} \theta^\top \mathbf{X}^\top \mathbf{X}\theta + \tau^{-2} \theta^\top \theta - 2\sigma^{-2} \mathbf{y}^\top \mathbf{X}\theta \right) \right] \\ &= \exp \left[-\frac{1}{2} \theta^\top \underbrace{\left(\sigma^{-2} \mathbf{X}^\top \mathbf{X} + \tau^{-2} \mathbf{I}_p \right)}_{:= \mathbf{A}} \theta + \color{red}{\sigma^{-2} \mathbf{y}^\top \mathbf{X}\theta} \right] \end{aligned}$$

This expression resembles a normal density - except for the term in red!

Review: The Bayesian Linear Model X

Note: We need not worry about the normalizing constant since its mere role is to convert probability functions to density functions with a total probability of one.

We subtract a (not yet defined) constant c while compensating for this change by adding the respective terms ("adding 0"), emphasized in green:

$$\begin{aligned} p(\boldsymbol{\theta} | \mathbf{X}, \mathbf{y}) &\propto \exp \left[-\frac{1}{2} (\boldsymbol{\theta} - \mathbf{c})^\top \mathbf{A} (\boldsymbol{\theta} - \mathbf{c}) - \mathbf{c}^\top \mathbf{A} \boldsymbol{\theta} + \underbrace{\frac{1}{2} \mathbf{c}^\top \mathbf{A} \mathbf{c}}_{\text{doesn't depend on } \boldsymbol{\theta}} + \sigma^{-2} \mathbf{y}^\top \mathbf{X} \boldsymbol{\theta} \right] \\ &\propto \exp \left[-\frac{1}{2} (\boldsymbol{\theta} - \mathbf{c})^\top \mathbf{A} (\boldsymbol{\theta} - \mathbf{c}) - \mathbf{c}^\top \mathbf{A} \boldsymbol{\theta} + \sigma^{-2} \mathbf{y}^\top \mathbf{X} \boldsymbol{\theta} \right] \end{aligned}$$

If we choose c such that $-\mathbf{c}^\top \mathbf{A} \boldsymbol{\theta} + \sigma^{-2} \mathbf{y}^\top \mathbf{X} \boldsymbol{\theta} = 0$, the posterior is normal with mean c and covariance matrix \mathbf{A}^{-1} . Taking into account that \mathbf{A} is symmetric, this is if we choose

$$\begin{aligned} \sigma^{-2} \mathbf{y}^\top \mathbf{X} &= \mathbf{c}^\top \mathbf{A} \\ \Leftrightarrow \sigma^{-2} \mathbf{y}^\top \mathbf{X} \mathbf{A}^{-1} &= \mathbf{c}^\top \\ \Leftrightarrow c &= \sigma^{-2} \mathbf{A}^{-1} \mathbf{X}^\top \mathbf{y} \end{aligned}$$

as claimed.

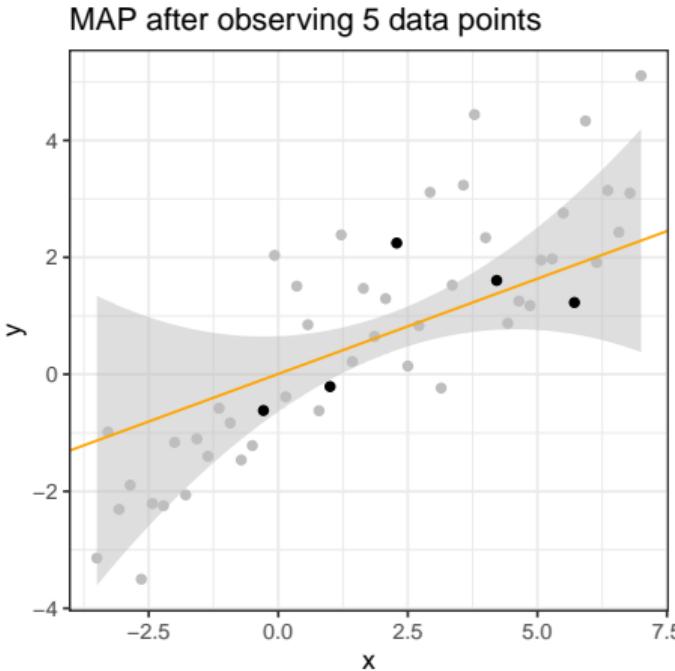
Review: The Bayesian Linear Model XI

- Based on the posterior distribution, $\theta | \mathbf{X}, \mathbf{y} \sim \mathcal{N}(\sigma^{-2} \mathbf{A}^{-1} \mathbf{X}^\top \mathbf{y}, \mathbf{A}^{-1})$, we can derive the predictive distribution for a new observation \mathbf{x}_* .
- The predictive distribution for the Bayesian linear model, i.e. the distribution of $\theta^\top \mathbf{x}_*$, is

$$y_* | \mathbf{X}, \mathbf{y}, \mathbf{x}_* \sim \mathcal{N}(\sigma^{-2} \mathbf{y}^\top \mathbf{X} \mathbf{A}^{-1} \mathbf{x}_*, \mathbf{x}_*^\top \mathbf{A}^{-1} \mathbf{x}_*).$$

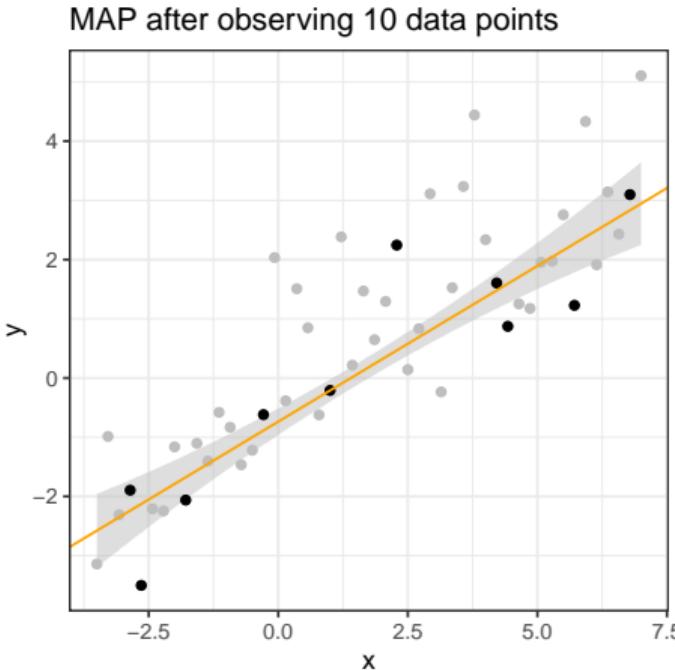
Note: This can be obtained by applying the rules for linear transformations of Gaussians.

Review: The Bayesian Linear Model XII



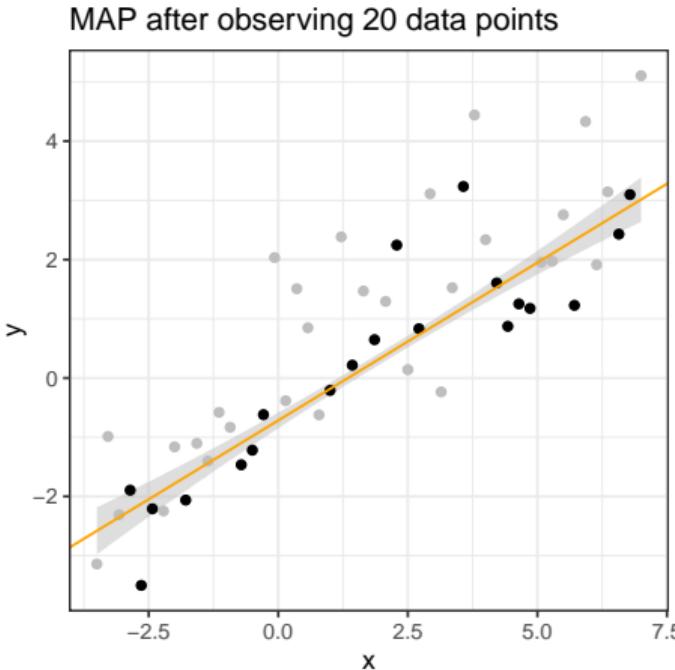
For every test input x_* , we get a distribution over the prediction y_* . In particular, we get a posterior mean (orange) and a posterior variance (the grey region, which equals \pm two times the standard deviation).

Review: The Bayesian Linear Model XIII



For every test input x_* , we get a distribution over the prediction y_* . In particular, we get a posterior mean (orange) and a posterior variance (the grey region, which equals \pm two times the standard deviation).

Review: The Bayesian Linear Model XIV



For every test input x_* , we get a distribution over the prediction y_* . In particular, we get a posterior mean (orange) and a posterior variance (the grey region, which equals \pm two times the standard deviation).

Summary: The Bayesian Linear Model

- By switching to a Bayesian perspective, we have not only point estimation for the parameter θ but also whole **distributions**.
- From the posterior distribution of θ , we can derive a predictive distribution for $y_* = \theta^\top \mathbf{x}_*$.
- We can perform online updates: the **posterior distribution** of θ can be updated whenever new datapoints are observed.
- In the next step, we would like go beyond the linear functions and develop a theory for functions with general shapes.

AutoML: Gaussian Processes

Gaussian Processes

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Weight-Space View

- So far, we have considered a hypothesis space \mathcal{H} of parameterized functions $f(\mathbf{x} | \boldsymbol{\theta})$ (in particular, the space of linear functions).
- Using Bayesian inference, we derived distributions for $\boldsymbol{\theta}$ after having observed data $\mathcal{D}_{\text{train}}$.
- Prior beliefs about the parameter are expressed via a prior distribution $q(\boldsymbol{\theta})$, which is updated according to Bayes' rule

$$\underbrace{p(\boldsymbol{\theta} | \mathbf{X}, \mathbf{y})}_{\text{posterior}} = \frac{\overbrace{p(\mathbf{y} | \mathbf{X}, \boldsymbol{\theta})}^{\text{likelihood}} \overbrace{q(\boldsymbol{\theta})}^{\text{prior}}}{\underbrace{p(\mathbf{y} | \mathbf{X})}_{\text{marginal}}}.$$

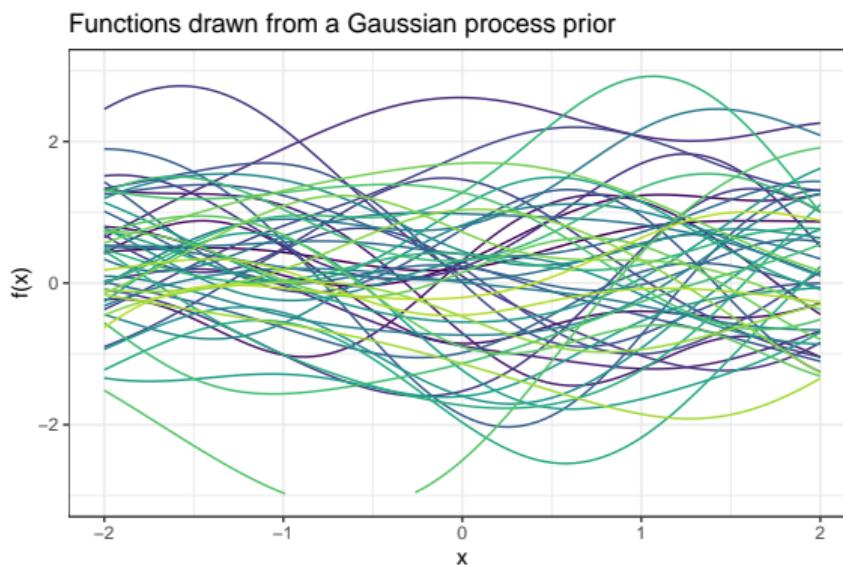
Function-Space View I

Let us change our point of view:

- Instead of “searching” for a parameter θ in the parameter space, we directly search in a space of “allowed” functions \mathcal{H} .
- We will still use Bayesian inference, but instead of specifying a prior distribution over a parameter, we will specify a prior distribution **over functions** and will update it according to the data points that we observe.

Function-Space View II

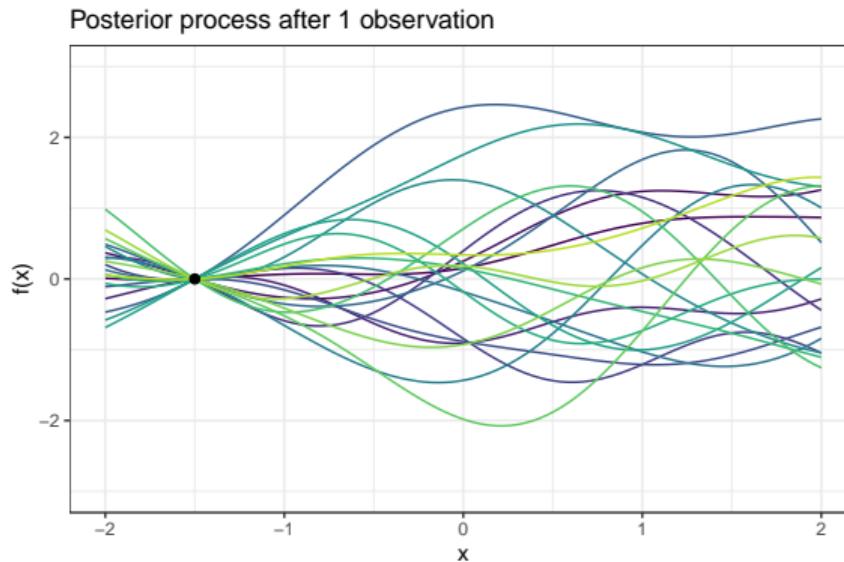
Intuitively, imagine we could draw a huge number of functions from some prior distribution over functions (*).



(*) We will see in a minute how distributions over functions can be specified.

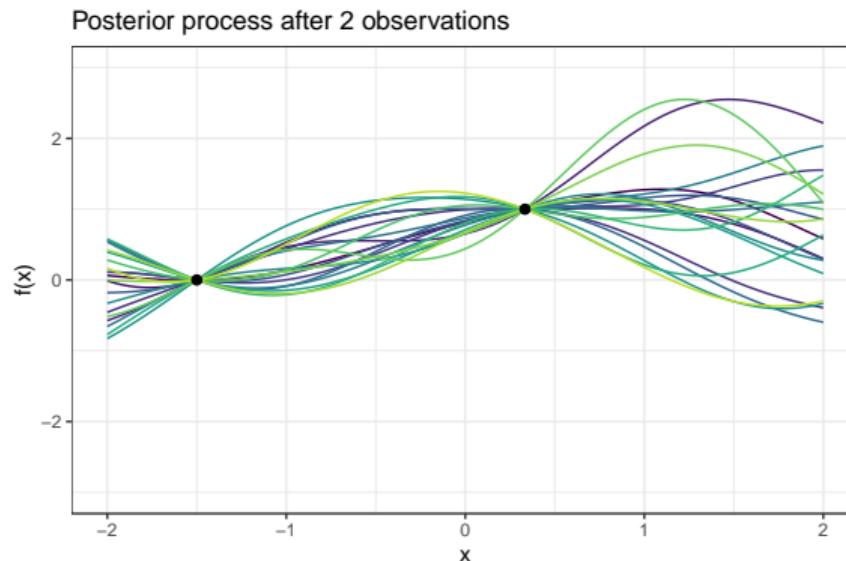
Function-Space View III

After observing some data points, we are allowed to sample only those functions that are consistent with the data.



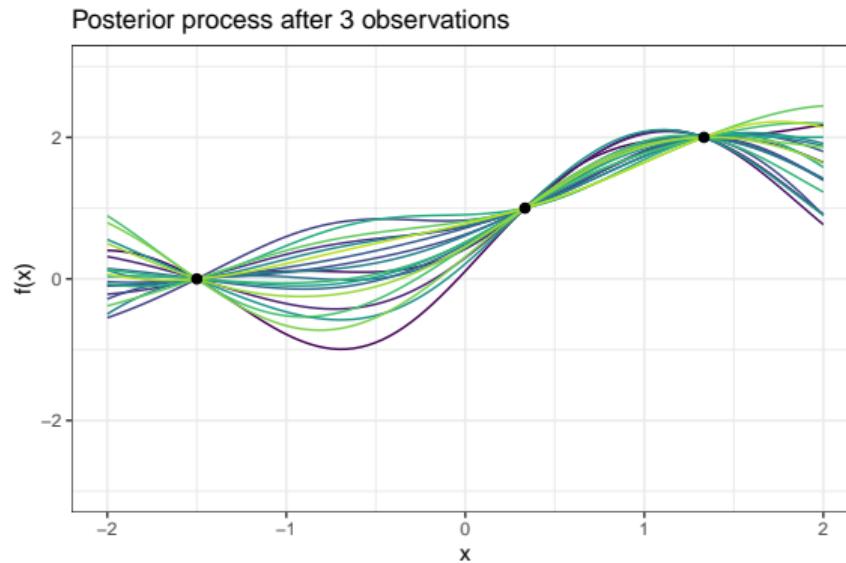
Function-Space View IV

After observing some data points, we are allowed to sample only those functions that are consistent with the data.



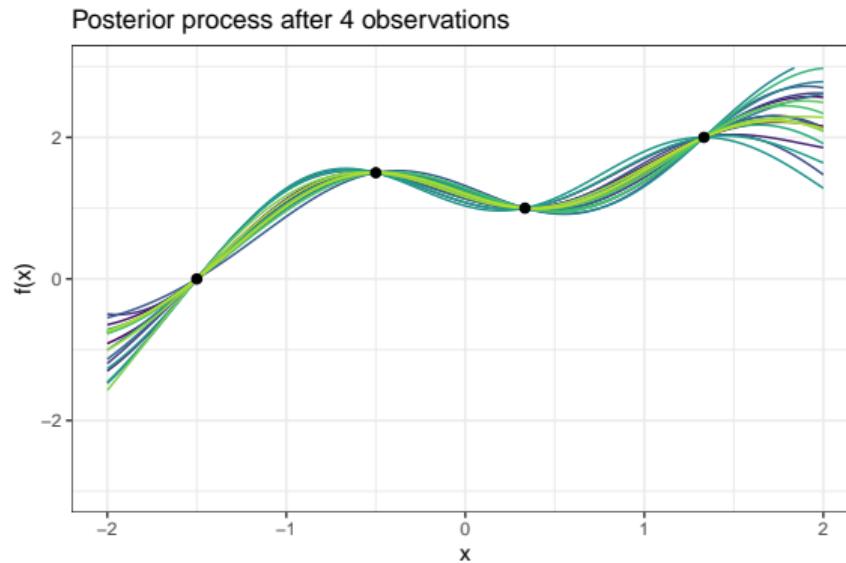
Function-Space View V

After observing some data points, we are allowed to sample only those functions that are consistent with the data.



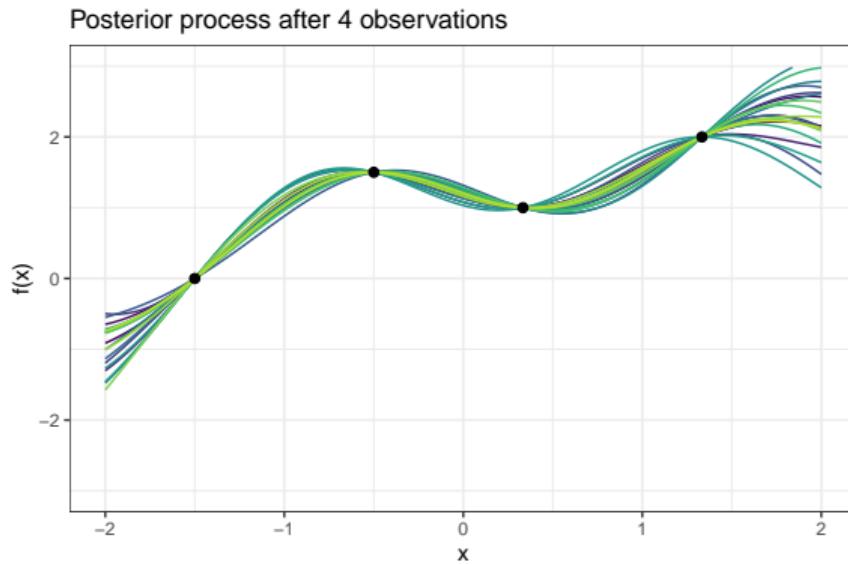
Function-Space View VI

As we observe more and more data points, the number of functions that consistent with the data shrinks.



Function-Space View VII

Intuitively, there is something like the “mean” and “variance” of a distribution over functions.



Weight-Space View vs. Function-Space View

Weight-Space View

Parameterize functions

$$\text{Example: } f(\mathbf{x} \mid \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \mathbf{x}$$

Define distributions on $\boldsymbol{\theta}$

Inference in parameter space Θ

Function-Space View

Define distributions on f

Inference in function space \mathcal{H}

Next, we will see how we can define distributions over functions mathematically.

Distributions on Functions

Discrete Functions I

For simplicity, we will firstly consider functions with finite domains.

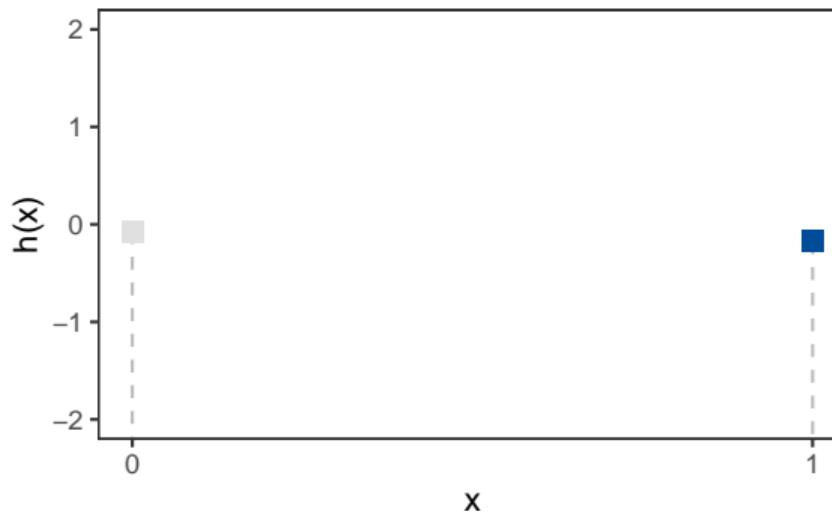
- Let $\mathcal{X} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$ be a finite set of elements and \mathcal{H} the set of all functions $h : \mathcal{X} \rightarrow \mathbb{R}$.
- Since the domain of any $h(\cdot) \in \mathcal{H}$ has only n elements, we can represent the function $h(\cdot)$ compactly as a n -dimensional vector

$$\mathbf{h} = [h(\mathbf{x}^{(1)}), \dots, h(\mathbf{x}^{(n)})].$$

Discrete Functions II

Example 1: Consider function $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **two** points $\mathcal{X} = \{0, 1\}$.

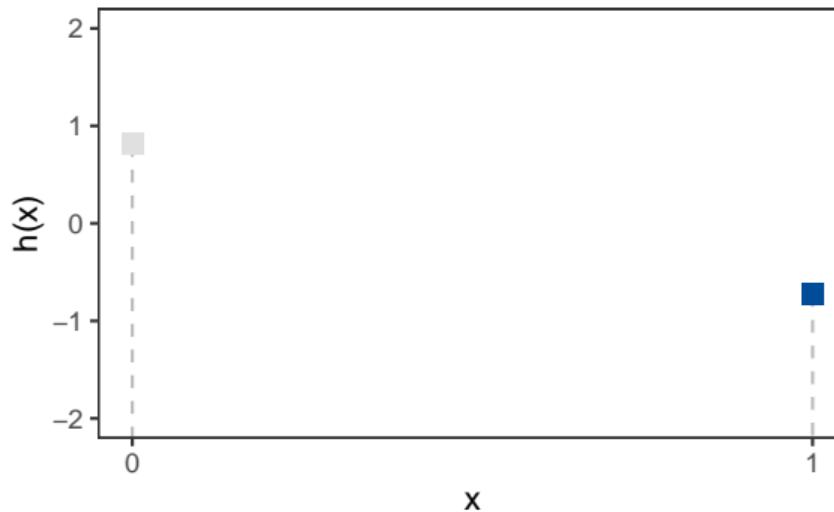
Examples for functions that live in this space:



Discrete Functions III

Example 1: Consider function $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **two** points $\mathcal{X} = \{0, 1\}$.

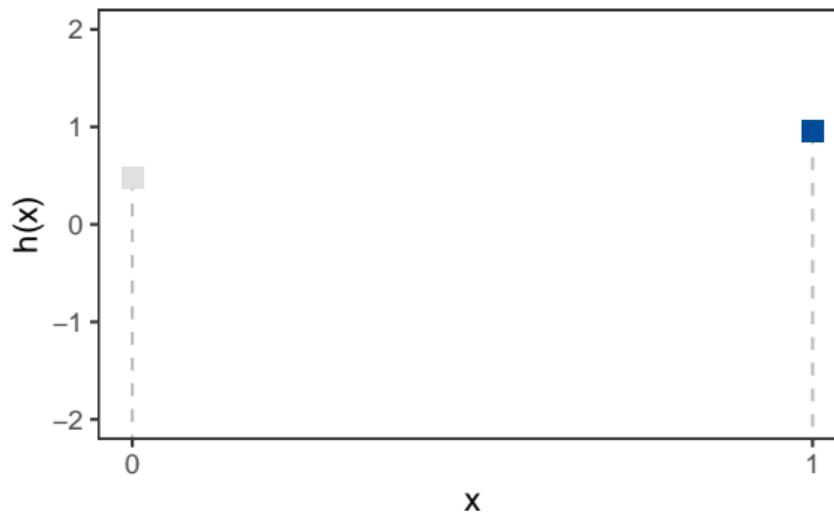
Examples for functions that live in this space:



Discrete Functions IV

Example 1: Consider function $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **two** points $\mathcal{X} = \{0, 1\}$.

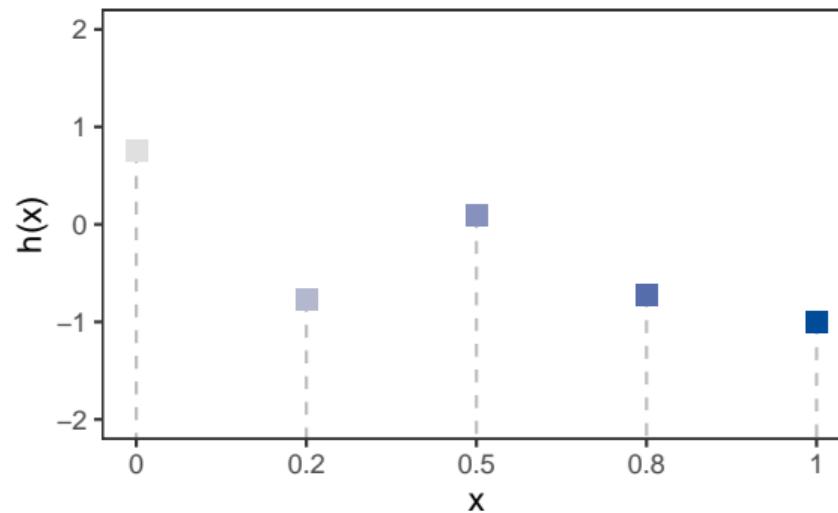
Examples for functions that live in this space:



Discrete Functions V

Example 2: Consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **five** points
 $\mathcal{X} = \{0, 0.25, 0.5, 0.75, 1\}$.

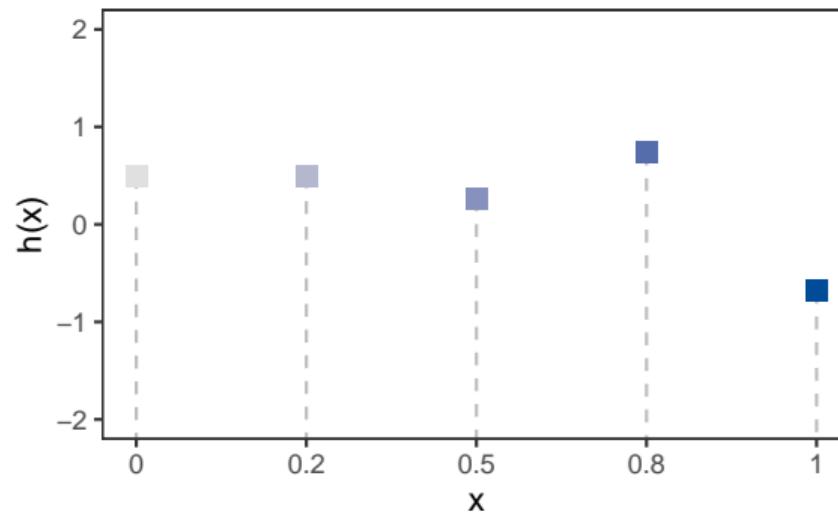
Examples for functions that live in this space:



Discrete Functions VI

Example 2: Consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **five** points
 $\mathcal{X} = \{0, 0.25, 0.5, 0.75, 1\}$.

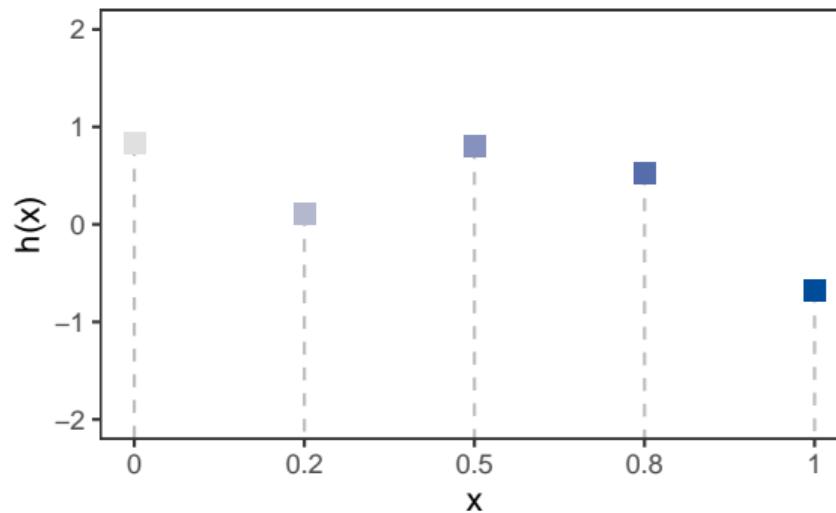
Examples for functions that live in this space:



Discrete Functions VII

Example 2: Consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **five** points
 $\mathcal{X} = \{0, 0.25, 0.5, 0.75, 1\}$.

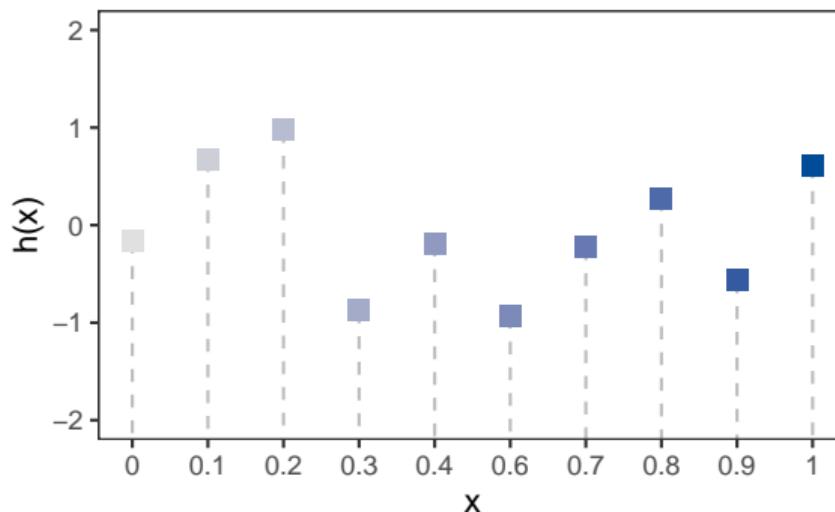
Examples for functions that live in this space:



Discrete Functions VIII

Example 3: Consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **ten** points.

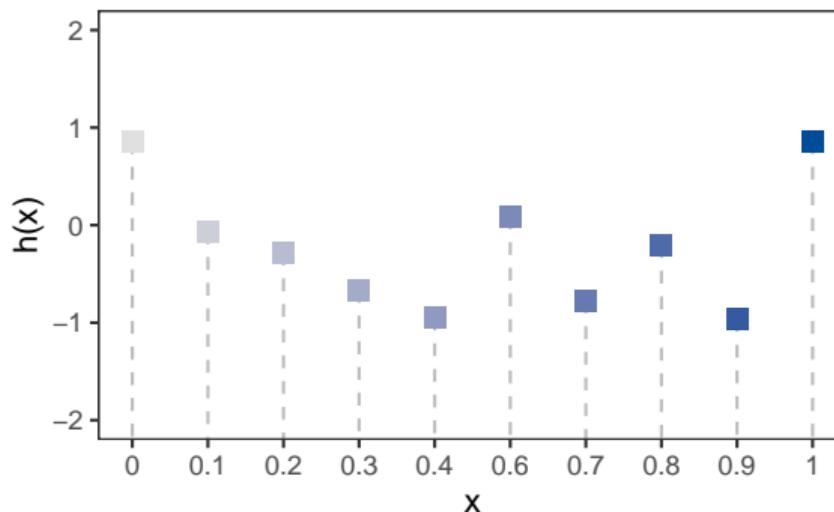
Examples for functions that live in this space:



Discrete Functions IX

Example 3: Consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **ten** points.

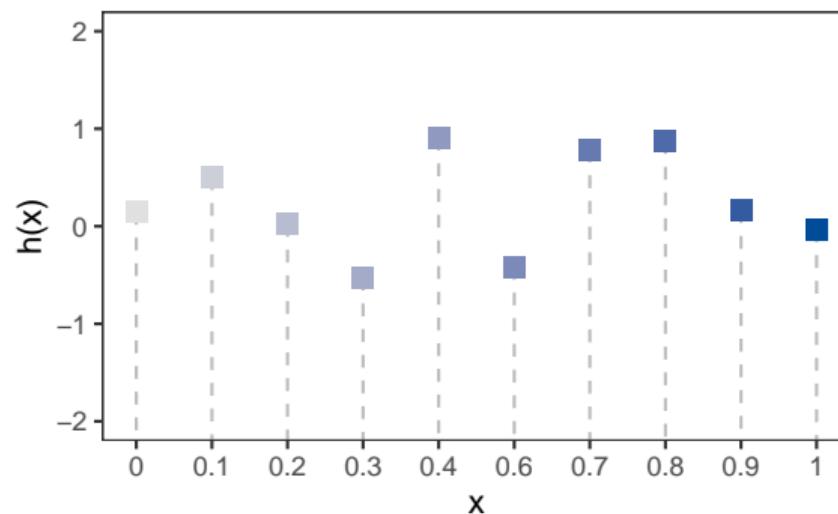
Examples for functions that live in this space:



Discrete Functions X

Example 3: Consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **ten** points.

Examples for functions that live in this space:



Distributions on Discrete Functions I

- One natural way to specify a probability distribution on a discrete function $h \in \mathcal{H}$ is to use the vector representation of the function:

$$\mathbf{h} = [h(\mathbf{x}^{(1)}), h(\mathbf{x}^{(2)}), \dots, h(\mathbf{x}^{(n)})].$$

- Let us consider \mathbf{h} as a n -dimensional random variable. We will further assume the following normal distribution:

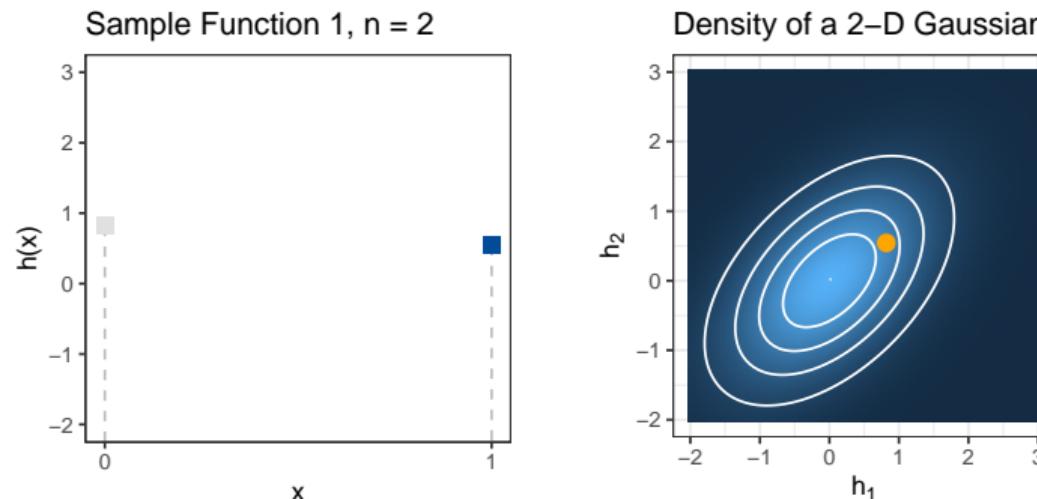
$$\mathbf{h} \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$

Note: For now, we set $\mathbf{m} = \mathbf{0}$ and take the covariance matrix \mathbf{K} as given. We will see later how they are chosen / estimated.

Distributions on Discrete Functions II

Example 1 (continued): Let $h : \mathcal{X} \rightarrow \mathcal{Y}$ be a function that is defined on **two** points \mathcal{X} . We sample functions by sampling from a two-dimensional normal variable

$$\mathbf{h} = [h(1), h(2)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$

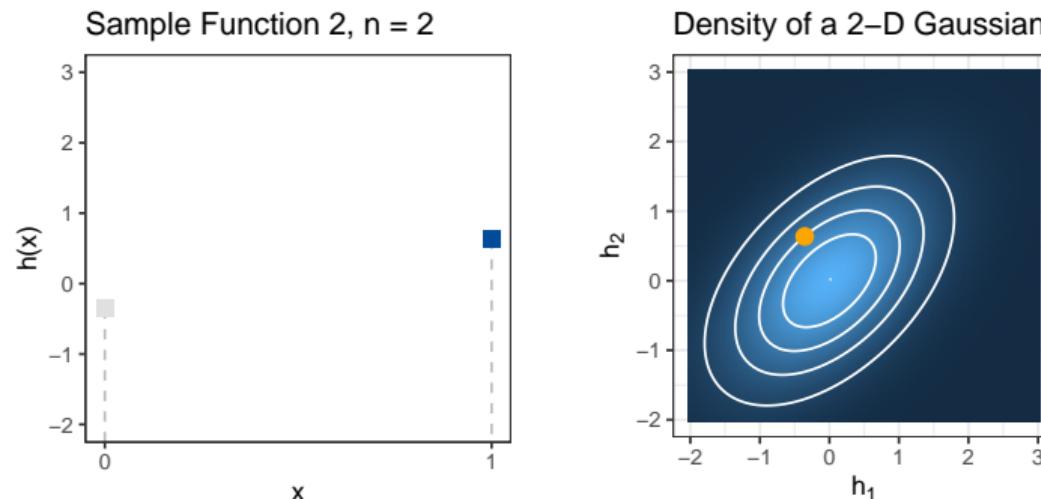


In this example, $m = (0, 0)$ and $K = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$.

Distributions on Discrete Functions III

Example 1 (continued): Let $h : \mathcal{X} \rightarrow \mathcal{Y}$ be a function that is defined on **two** points \mathcal{X} . We sample functions by sampling from a two-dimensional normal variable

$$\mathbf{h} = [h(1), h(2)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$

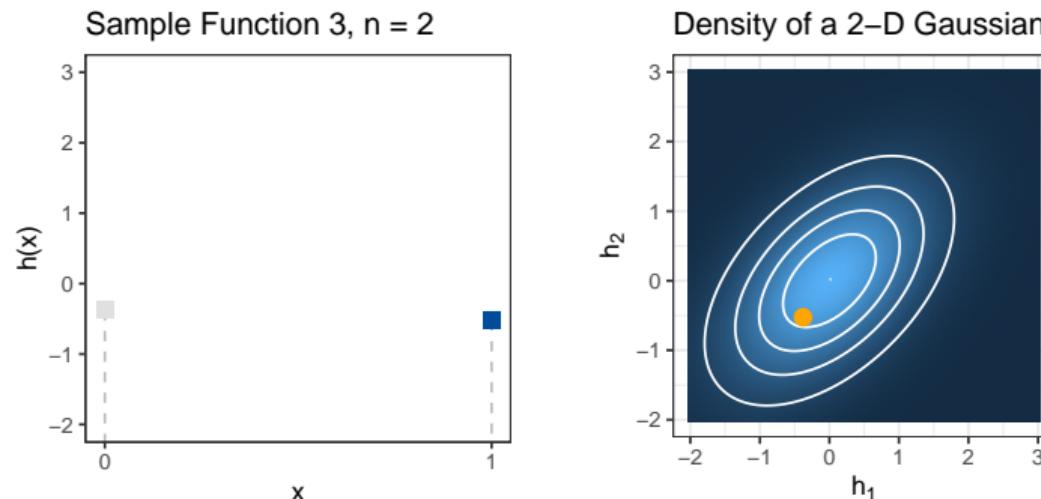


In this example, $m = (0, 0)$ and $K = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$.

Distributions on Discrete Functions IV

Example 1 (continued): Let $h : \mathcal{X} \rightarrow \mathcal{Y}$ be a function that is defined on **two** points \mathcal{X} . We sample functions by sampling from a two-dimensional normal variable

$$\mathbf{h} = [h(1), h(2)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$

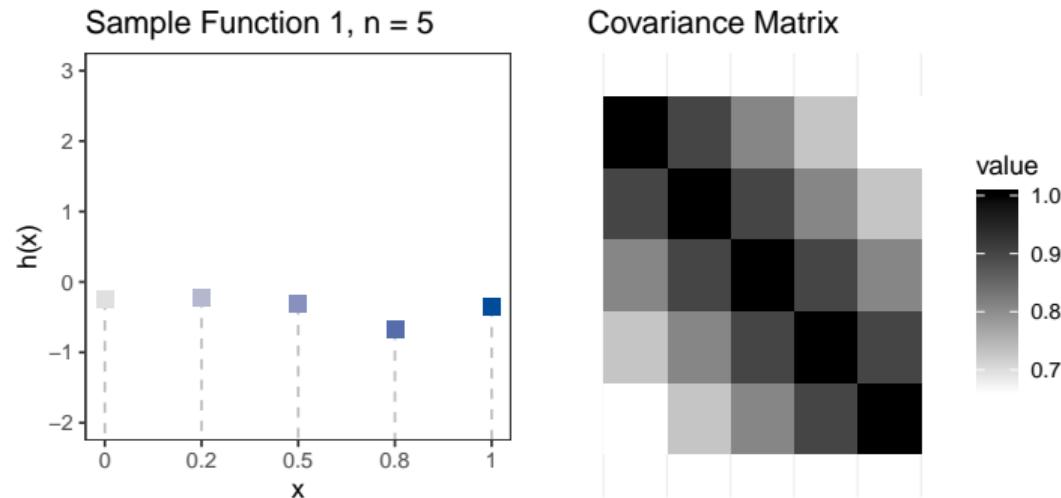


In this example, $m = (0, 0)$ and $K = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$.

Distributions on Discrete Functions V

Example 2 (continued): Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **five** points. We sample functions by sampling from a five-dimensional normal variable

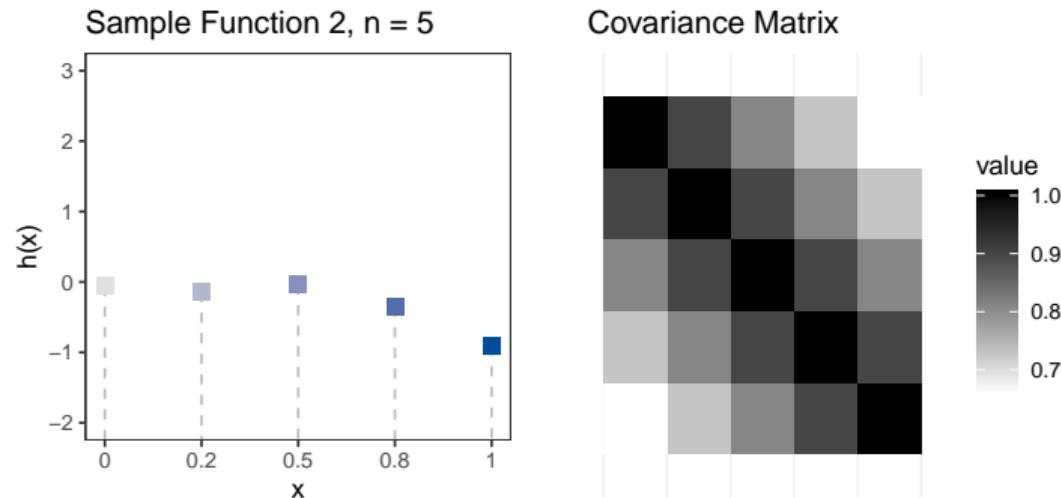
$$\mathbf{h} = [h(1), h(2), h(3), h(4), h(5)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$



Distributions on Discrete Functions VI

Example 2 (continued): Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **five** points. We sample functions by sampling from a five-dimensional normal variable

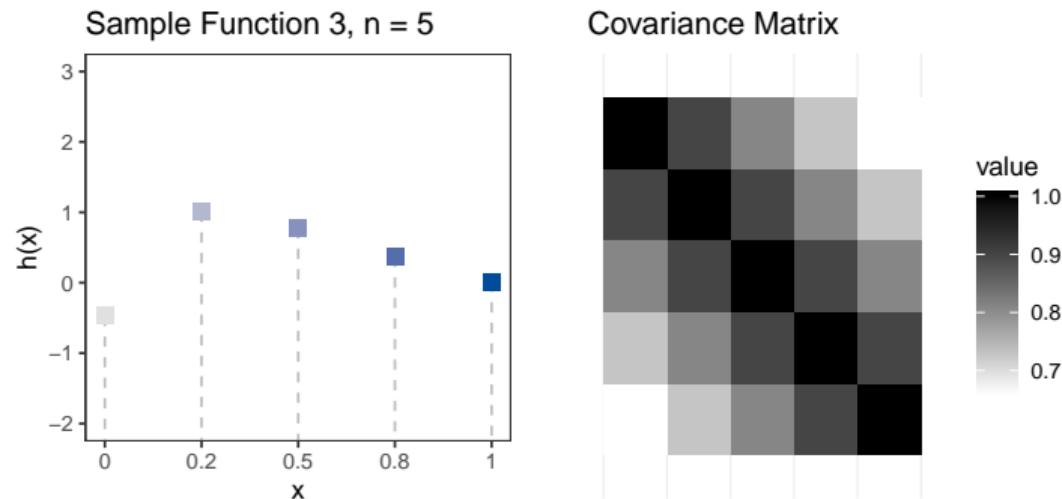
$$\mathbf{h} = [h(1), h(2), h(3), h(4), h(5)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$



Distributions on Discrete Functions VII

Example 2 (continued): Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **five** points. We sample functions by sampling from a five-dimensional normal variable

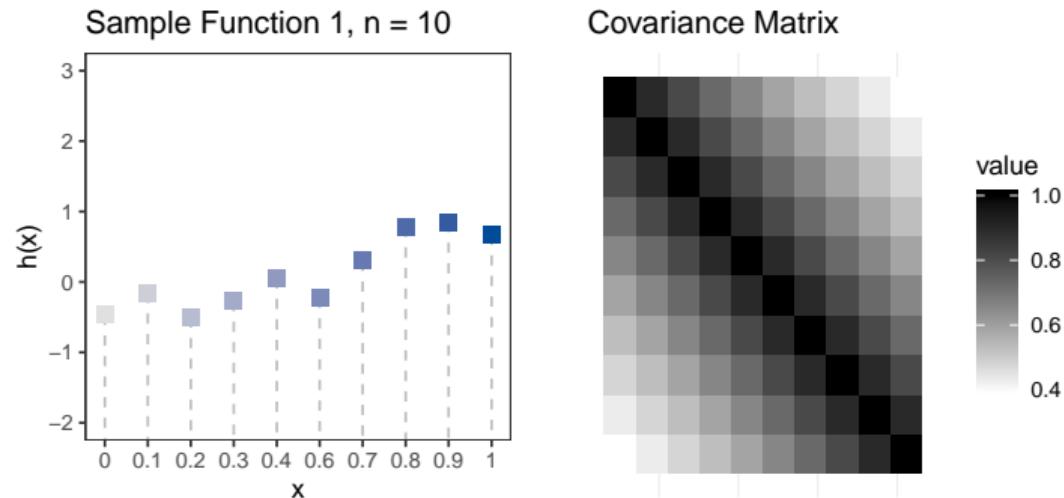
$$\mathbf{h} = [h(1), h(2), h(3), h(4), h(5)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$



Distributions on Discrete Functions VIII

Example 3 (continued): Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **ten** points. We sample functions by sampling from a ten-dimensional normal variable

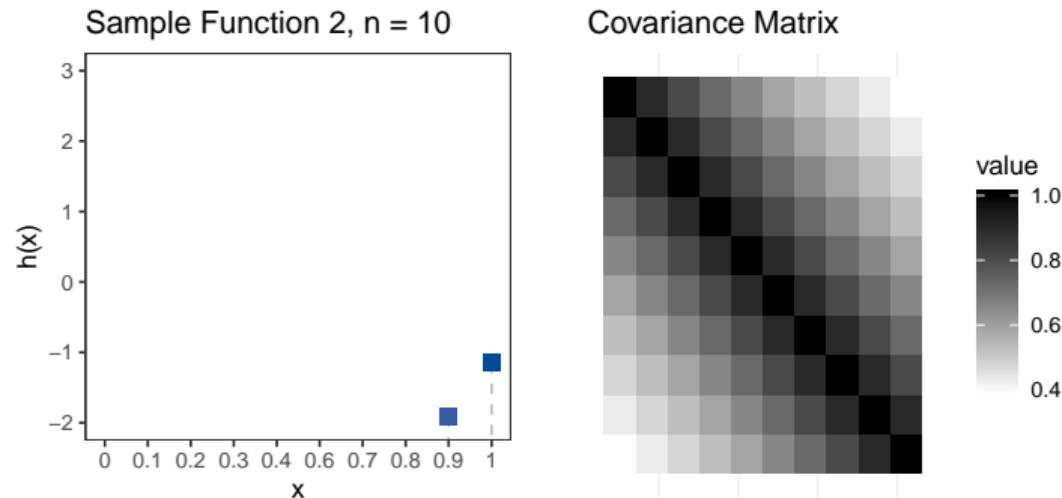
$$\mathbf{h} = [h(1), h(2), \dots, h(10)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$



Distributions on Discrete Functions IX

Example 3 (continued): Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **ten** points. We sample functions by sampling from a ten-dimensional normal variable

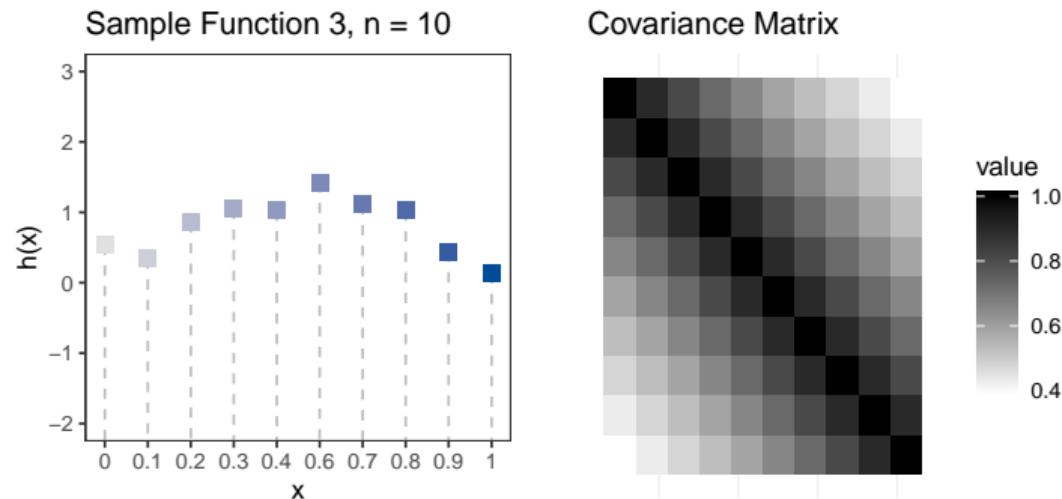
$$\mathbf{h} = [h(1), h(2), \dots, h(10)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$



Distributions on Discrete Functions X

Example 3 (continued): Let us consider $h : \mathcal{X} \rightarrow \mathcal{Y}$ where the input space consists of **ten** points. We sample functions by sampling from a ten-dimensional normal variable

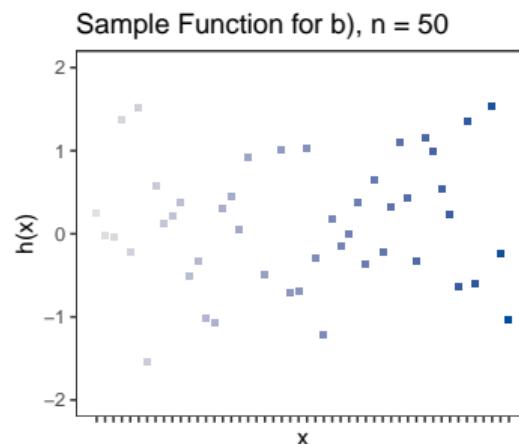
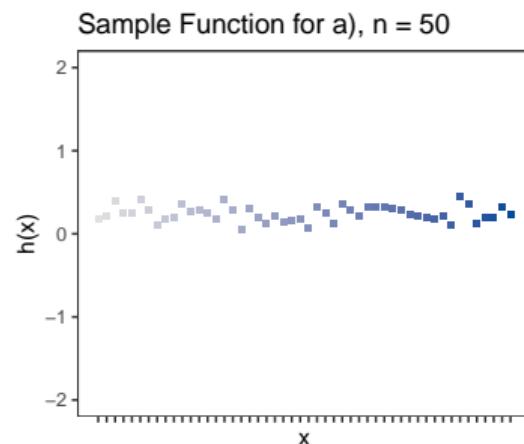
$$\mathbf{h} = [h(1), h(2), \dots, h(10)] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$



The Role of Covariance Function I

The covariance controls the “shape” of drawn functions. Consider two extreme cases where function values are:

- a) strongly correlated: $\mathbf{K} = \begin{pmatrix} 1 & 0.99 & \dots & 0.99 \\ 0.99 & 1 & \dots & 0.99 \\ 0.99 & 0.99 & \ddots & 0.99 \\ 0.99 & \dots & 0.99 & 1 \end{pmatrix}$
- b) uncorrelated: $\mathbf{K} = \mathbf{I}$.



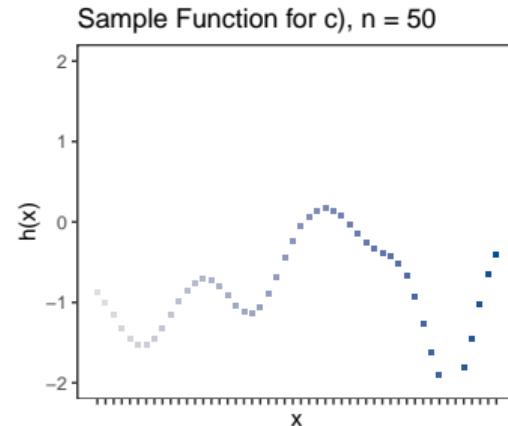
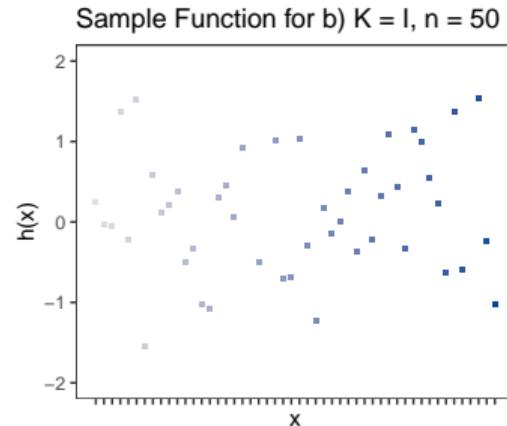
The Role of Covariance Function II

- On a numeric space \mathcal{X} , “meaningful” functions may be characterized by the following spatial property:
If $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are close in the \mathcal{X} -space, their function values $f(\mathbf{x}^{(i)})$ and $f(\mathbf{x}^{(j)})$ should be close in \mathcal{Y} -space.
- 💡 In other words, if two data points are close in \mathcal{X} -space, their corresponding values should be **correlated!**
- 💡 We can enforce this condition by choosing a covariance function for which, K_{ij} is high, if $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are close.

The Role of Covariance Function III

We can compute the entries of the covariance matrix by a function that is based on the distance between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$. For example:

c) spatial correlation: $K_{ij} = k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp\left(-\frac{1}{2} \left|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right|^2\right)$



Note: $k(\cdot, \cdot)$ is known as the **covariance function** or **kernel**. It will be studied in more detail later on.

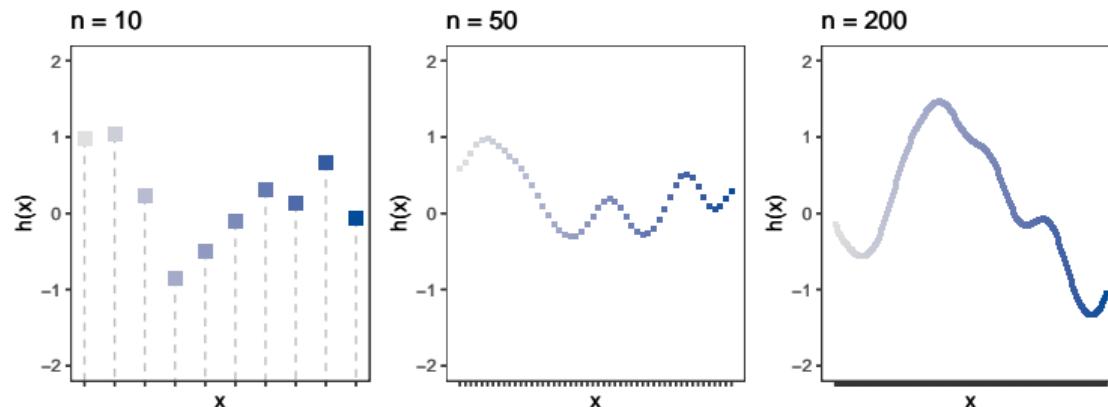
Gaussian Processes

From Discrete to Continuous Functions

- We have already considered distributions on functions with discrete domain. We did so, by defining Gaussian distributions on the vector of the respective function values

$$\mathbf{h} = [h(\mathbf{x}^{(1)}), h(\mathbf{x}^{(2)}), \dots, h(\mathbf{x}^{(n)})] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$

- We can generalize this idea for $n \rightarrow \infty$.



Gaussian Processes: Intuition I

- No matter how large n is, we consider functions with discrete domains.
- But, how can we extend our definition to functions with **continuous** domains $\mathcal{X} \subset \mathbb{R}$?
- Intuitively, a function f drawn from a **Gaussian process** can be understood as an “infinite” long Gaussian random vector.
- It is unclear how to handle an “infinite” long Gaussian random vector!

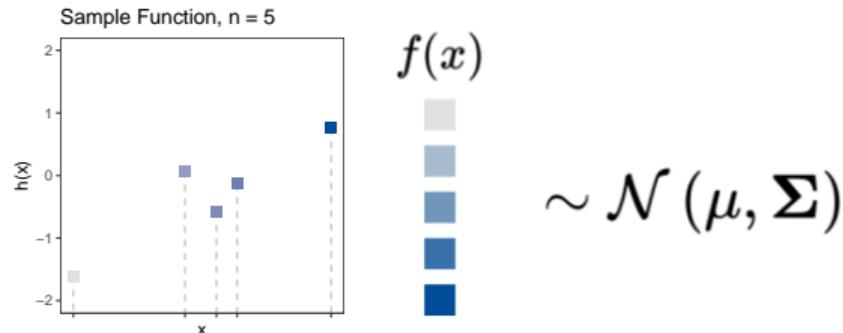


Gaussian Processes: Intuition II

- Thus, it is required that for **any finite set** of inputs $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\} \subset \mathcal{X}$, the vector \mathbf{f} has a Gaussian distribution with \mathbf{m} and \mathbf{K} being calculated by a mean function $m(\cdot)$ and a covariance function $k(\cdot, \cdot)$:

$$\mathbf{f} = \left[f\left(\mathbf{x}^{(1)}\right), \dots, f\left(\mathbf{x}^{(n)}\right) \right] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$

- This property is called the **Marginalization Property**.

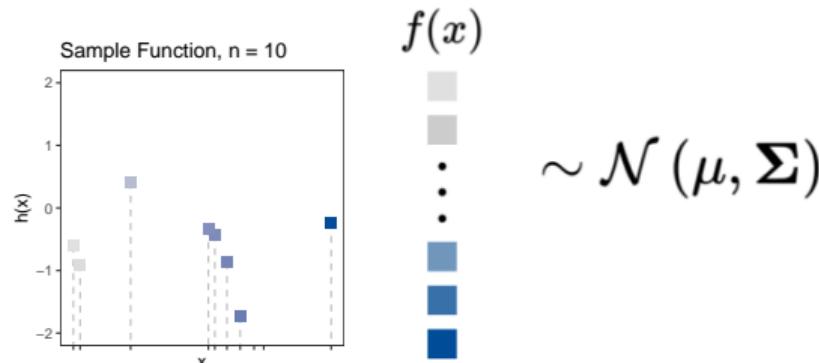


Gaussian Processes: Intuition III

- Thus, it is required that for **any finite set** of inputs $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\} \subset \mathcal{X}$, the vector \mathbf{f} has a Gaussian distribution with \mathbf{m} and \mathbf{K} being calculated by a mean function $m(\cdot)$ and a covariance function $k(\cdot, \cdot)$:

$$\mathbf{f} = \left[f\left(\mathbf{x}^{(1)}\right), \dots, f\left(\mathbf{x}^{(n)}\right) \right] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$

- This property is called the **Marginalization Property**.

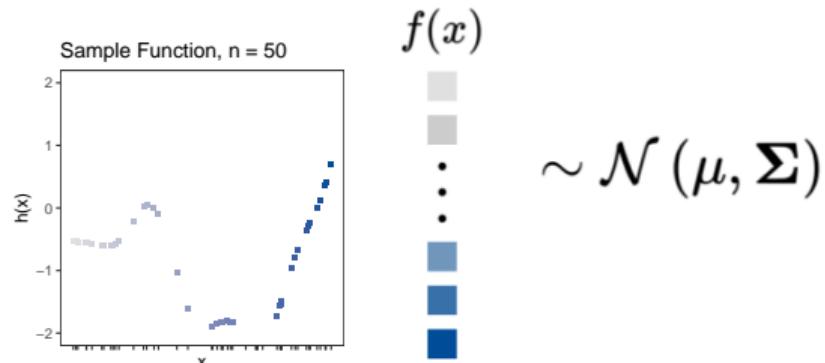


Gaussian Processes: Intuition IV

- Thus, it is required that for **any finite set** of inputs $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\} \subset \mathcal{X}$, the vector \mathbf{f} has a Gaussian distribution with \mathbf{m} and \mathbf{K} being calculated by a mean function $m(\cdot)$ and a covariance function $k(\cdot, \cdot)$:

$$\mathbf{f} = \left[f\left(\mathbf{x}^{(1)}\right), \dots, f\left(\mathbf{x}^{(n)}\right) \right] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$

- This property is called the **Marginalization Property**.



Gaussian Processes: Formal Definitions I

- The above intuitive explanation is formally defined as follows.

A function $f(\mathbf{x})$ is generated by a Gaussian process \mathcal{G} if for **any finite set of inputs** $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$, the associated vector of function values has a Gaussian distribution:

$$\mathbf{f} = \left(f(\mathbf{x}^{(1)}), \dots, f(\mathbf{x}^{(n)}) \right) \sim \mathcal{N}\left(\mathbf{m}, \mathbf{K}\right),$$

with

$$\mathbf{m} := \left(m\left(\mathbf{x}^{(i)}\right) \right)_i, \quad \mathbf{K} := \left(k\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) \right)_{i,j},$$

where $m(\mathbf{x})$ is called mean function and $k(\mathbf{x}, \mathbf{x}')$ is called covariance function.

Gaussian Processes: Formal Definitions II

- A GP is **completely specified** by its mean and covariance functions.
- The mean function $m(\mathbf{x})$ and the covariance function $k(\mathbf{x}, \mathbf{x}')$ of a real process $f(\mathbf{x})$ are defined as:

$$\begin{aligned}m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})] \\k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}\left[(f(\mathbf{x}) - \mathbb{E}[f(\mathbf{x})])(f(\mathbf{x}') - \mathbb{E}[f(\mathbf{x}')])\right]\end{aligned}$$

- We denote a GP by

$$f(\mathbf{x}) \sim \mathcal{G}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$$

Note: For now, we assume $m(\mathbf{x}) \equiv 0$. This is not a drastic limitation. In fact, it is common to consider GPs with a zero mean function.

Sampling from a Gaussian Process Prior I

- We can draw functions from a Gaussian process prior. To do so, consider $f(\mathbf{x}) \sim \mathcal{G}(0, k(\mathbf{x}, \mathbf{x}'))$ with the squared exponential covariance function (*)

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2\ell^2}\|\mathbf{x} - \mathbf{x}'\|^2\right), \quad \ell = 1.$$

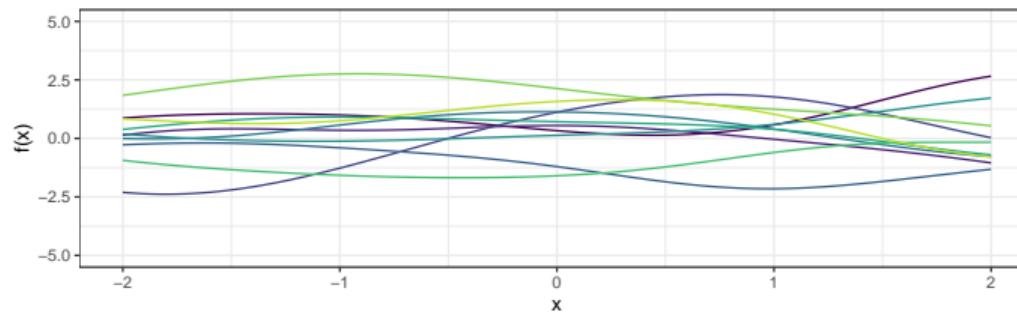
- This covariance function specifies the Gaussian process completely.

(*) We will talk later about different choices of covariance functions.

Sampling from a Gaussian Process Prior II

To visualize a sample function, we

- choose a large number of equidistant points: $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$,
- compute their corresponding covariance matrix by plugging in all pairs of $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ in $\mathbf{K} = (k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}))_{i,j}$,
- sample from a Gaussian $f \sim \mathcal{N}(\mathbf{0}, \mathbf{K})$.



We draw 10 times from the Gaussian, to get 10 different samples. Since we specified the mean function to be zero, the drawn functions have a zero mean.

Gaussian Processes as an Indexed Family

Gaussian Processes as an Indexed Family

- A Gaussian process is a special case of a **stochastic process** which is defined as a collection of random variables indexed by some index set (also called an **indexed family**).
- What does it mean?
- An **indexed family** is a mathematical function (or “rule”) that maps indices $t \in T$ to objects in \mathcal{S} .

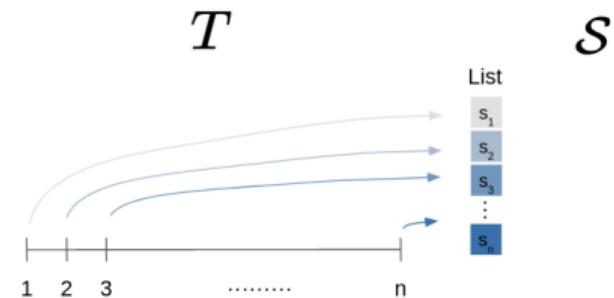
Definition: *an **index family** (or a family of elements in \mathcal{S} indexed by T) is a surjective function that is defined as follows:*

$$\begin{aligned}s : T &\rightarrow \mathcal{S} \\ t &\mapsto s_t = s(t)\end{aligned}$$

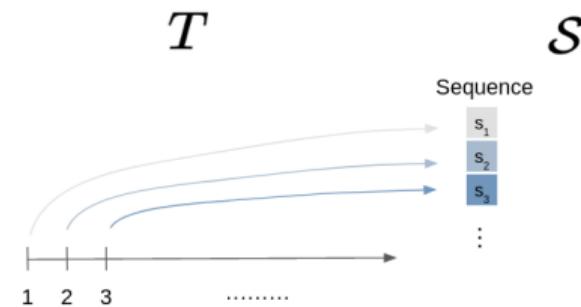
Index Family I

Some simple examples for indexed families are:

- Finite sequences (lists): $T = \{1, 2, \dots, n\}$ and $(s_t)_{t \in T} \in \mathbb{R}$



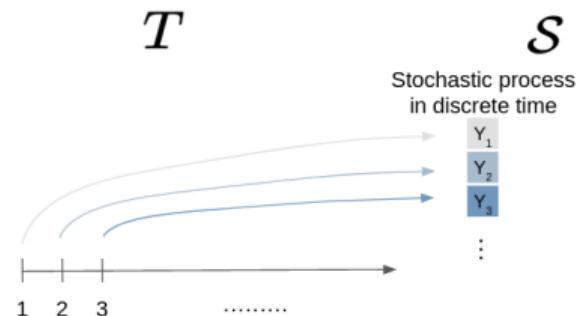
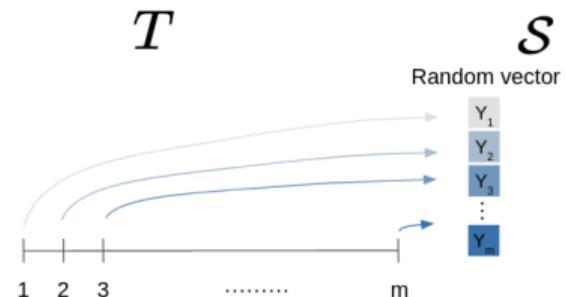
- Infinite sequences: $T = \mathbb{N}$ and $(s_t)_{t \in T} \in \mathbb{R}$



Index Family II

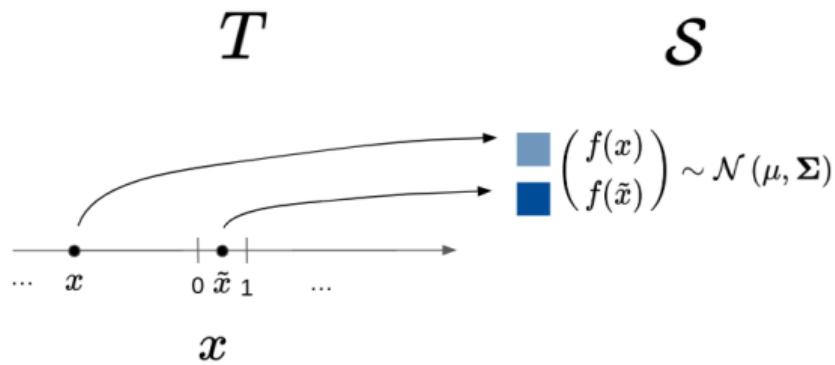
But the indexed set \mathcal{S} can be something more complicated, for example functions or **random variables** (RV):

- $T = \{1, \dots, m\}$, Y_t 's are RVs: Indexed family is a random vector.
- $T = \{1, \dots, m\}$, Y_t 's are RVs: Indexed family is a stochastic process in discrete time.
- $T = \mathbb{Z}^2$, Y_t 's are RVs: Indexed family is a 2D-random walk.



Index Family III

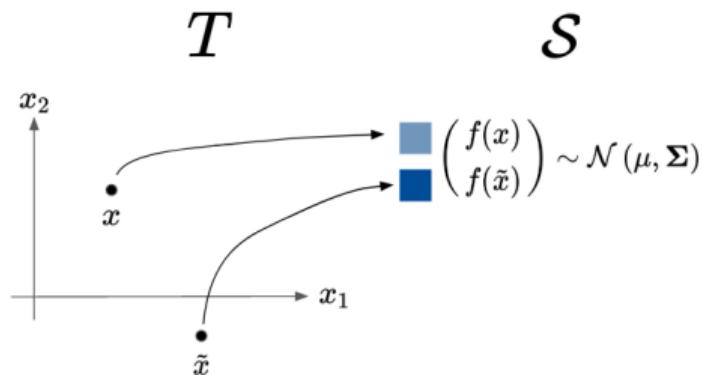
- A Gaussian process is also an indexed family, where the random variables $f(\mathbf{x})$ are indexed by the input values $\mathbf{x} \in \mathcal{X}$.
- Importantly, any indexed (finite) random vector has a multivariate Gaussian distribution (which comes with all the nice properties of Gaussianity!).



Visualization for a one-dimensional \mathcal{X} .

Index Family IV

- A Gaussian process is also an indexed family, where the random variables $f(\mathbf{x})$ are indexed by the input values $\mathbf{x} \in \mathcal{X}$.
- Importantly, any indexed (finite) random vector has a multivariate Gaussian distribution (which comes with all the nice properties of Gaussianity!).



Visualization for a two-dimensional \mathcal{X} .

AutoML: Gaussian Processes

Covariance Functions for GPs

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Covariance function of a GP I

The marginalization property of the Gaussian process implies that for any finite set of input values, the corresponding vector of function values is Gaussian:

$$\mathbf{f} = \left[f(\mathbf{x}^{(1)}), \dots, f(\mathbf{x}^{(n)}) \right] \sim \mathcal{N}(\mathbf{m}, \mathbf{K}).$$

- The covariance matrix \mathbf{K} is constructed according to the chosen inputs $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$.
- Each entry K_{ij} is computed by $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$.
- Technically, to be a valid covariance matrix, \mathbf{K} needs to be positive semi-definite for **every** choice of inputs $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$.
- A function $k(\cdot, \cdot)$ that satisfies this condition is called **positive definite**.

Covariance function of a GP II

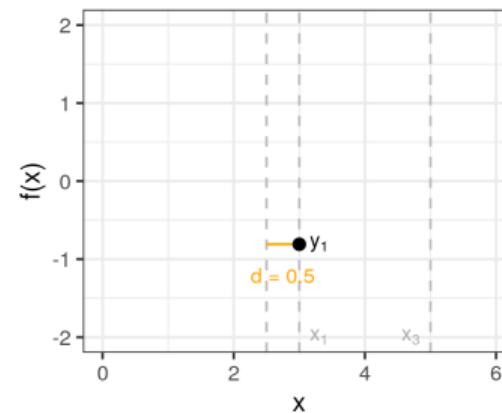
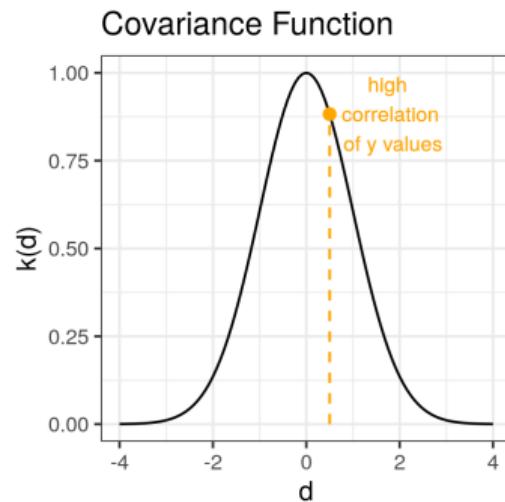
- Recall that the purpose of the covariance function is to control to which degree the following condition is fulfilled:

If $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ are close in the \mathcal{X} -space, their function values $f(\mathbf{x}^{(i)})$ and $f(\mathbf{x}^{(j)})$ should be close in \mathcal{Y} -space.

- 💡 Closeness of $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ in the input space \mathcal{X} is measured by $\mathbf{d} = \mathbf{x}^{(i)} - \mathbf{x}^{(j)}$.

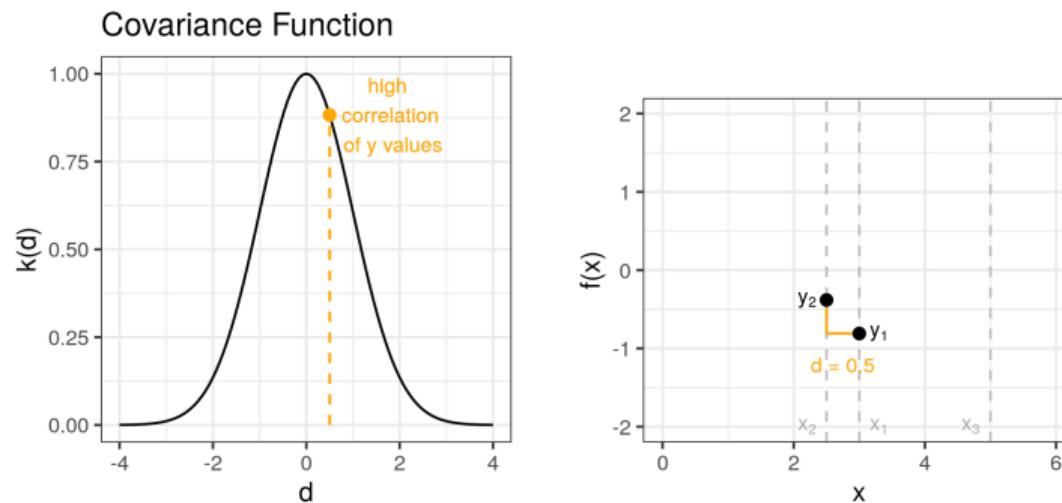
Covariance function of a GP: Example I

- Let $f(\mathbf{x})$ be a GP with $k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2}\|\mathbf{d}\|^2)$ where $\mathbf{d} = \mathbf{x} - \mathbf{x}'$.
- Consider two points $\mathbf{x}^{(1)} = 3$ and $\mathbf{x}^{(2)} = 2.5$. To investigate how correlated their function values are, compute their correlation!



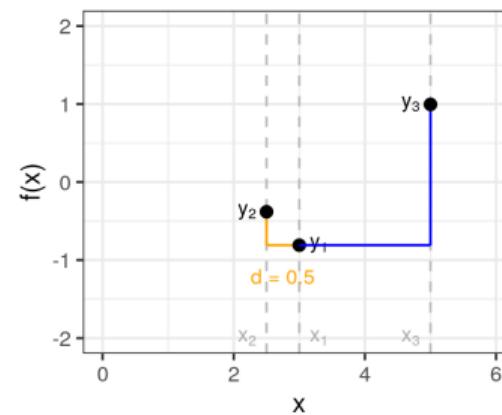
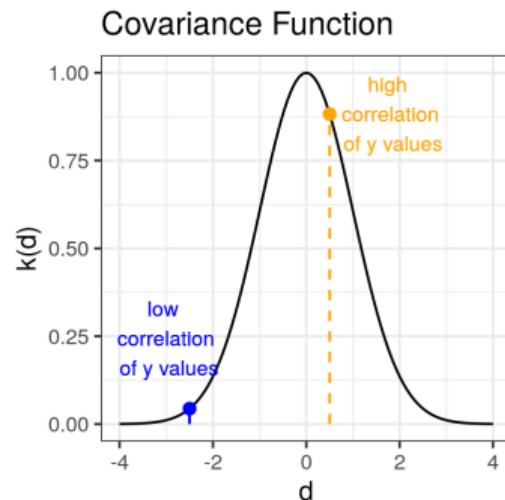
Covariance function of a GP: Example II

- Assume that we observe a value of $y^{(1)} = -0.8$. Under the said assumption for the Gaussian process, the value of $y^{(2)}$ should be close to $y^{(1)}$.



Covariance function of a GP: Example III

- Now, let us take a new point $\mathbf{x}^{(3)}$ which is not too close to $\mathbf{x}^{(1)}$.
- Their function values should not be so correlated. That is, $y^{(1)}$ and $y^{(3)}$ are probably far away from each other.



Covariance Functions

Three types of properties are commonly used in covariance functions:

- k is **stationary** if it depends only on $\mathbf{d} = \mathbf{x} - \mathbf{x}'$ and is denoted by $k(\mathbf{d})$.
- k is **isotropic** if it depends only on $r = \|\mathbf{x} - \mathbf{x}'\|$ and is denoted by $k(r)$.
- k is a **dot product** if it depends only on $\mathbf{x}^T \mathbf{x}'$.

💡 Isotropy implies stationarity.

💡 Isotropic functions are rotationally invariant.

💡 Stationary functions are translationally invariant:

$$k(\mathbf{x}, \mathbf{x} + \mathbf{d}) = k(\mathbf{0}, \mathbf{d}) = k(\mathbf{d})$$

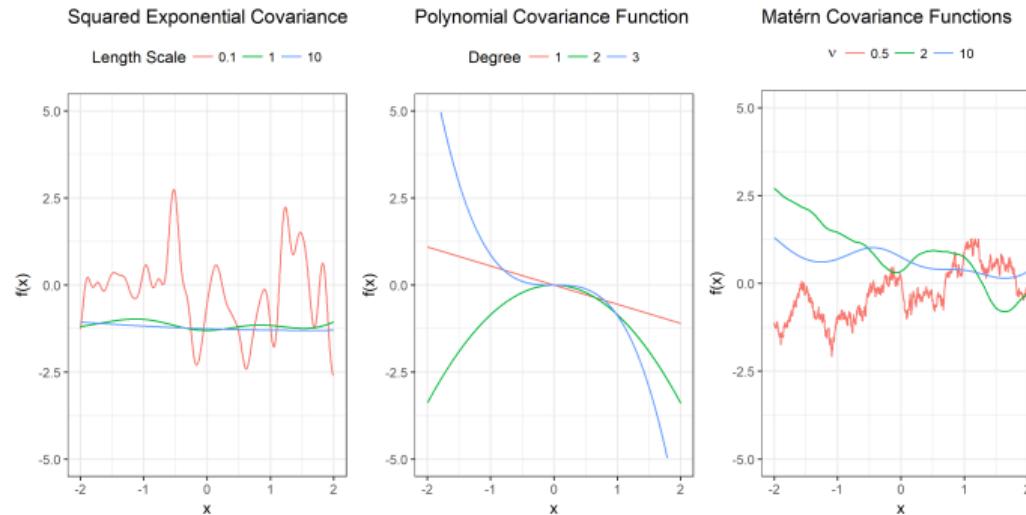
Commonly Used Covariance Functions I

Name	$k(\mathbf{x}, \mathbf{x}')$
constant	σ_0^2
linear	$\sigma_0^2 + \mathbf{x}^T \mathbf{x}'$
polynomial	$(\sigma_0^2 + \mathbf{x}^T \mathbf{x}')^p$
squared exponential	$\exp(-\frac{\ \mathbf{x}-\mathbf{x}'\ ^2}{2\ell^2})$
Matérn	$\frac{1}{2^\nu \Gamma(\nu)} \left(\frac{\sqrt{2\nu}}{\ell} \ \mathbf{x} - \mathbf{x}'\ \right)^\nu K_\nu \left(\frac{\sqrt{2\nu}}{\ell} \ \mathbf{x} - \mathbf{x}'\ \right)$
exponential	$\exp \left(-\frac{\ \mathbf{x}-\mathbf{x}'\ }{\ell} \right)$

$K_\nu(\cdot)$ is the modified Bessel function of the second kind.

Commonly Used Covariance Functions II

- Some random functions drawn from Gaussian processes with a Squared Exponential Kernel (left), Polynomial Kernel (middle), and a Matérn Kernel (right, $\ell = 1$).
- The length-scale hyperparameter determines the “wiggliness” of the function.
- For Matérn, the ν parameter determines how differentiable the process is.



Squared Exponential Covariance Function

The squared exponential function is one of the most commonly used covariance functions.

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right).$$

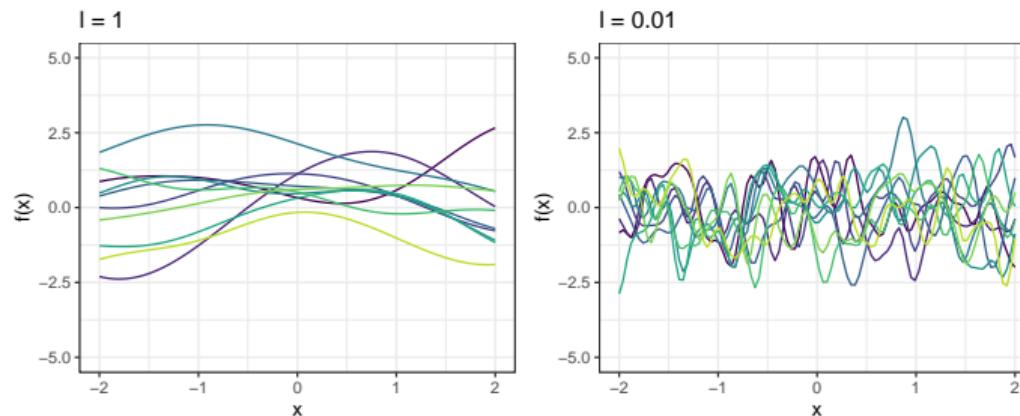
Properties:

- 💡 It depends merely on the distance $r = \|\mathbf{x} - \mathbf{x}'\| \rightarrow$ isotropic and stationary.
- 💡 Infinitely differentiable \rightarrow the corresponding GP is too smooth.
- 💡 It utilizes strong smoothness assumptions \rightarrow unrealistic for modeling most of the physical processes.

Characteristic Length-Scale I

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2\ell^2}\|\mathbf{x} - \mathbf{x}'\|^2\right)$$

ℓ is called **characteristic length-scale**. Loosely speaking, the characteristic length-scale describes how far you need to move in input space for the function values to become uncorrelated. Higher ℓ induces smoother functions, lower ℓ induces more wiggly functions.



Characteristic Length-Scale II

For more than $p = 2$ dimensions, the squared exponential can be parameterized as follows:

$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sigma_f^2 \exp\left(-\frac{1}{2} \left(\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right)^\top \mathbf{M} \left(\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right)\right)$$

Possible choices for the matrix \mathbf{M} include

$$\mathbf{M}_1 = \ell^{-2} \mathbf{I} \quad \mathbf{M}_2 = \text{diag}(\ell)^{-2} \quad \mathbf{M}_3 = \Gamma \Gamma^\top + \text{diag}(\ell)^{-2}$$

where ℓ is a p -vector of positive values and Γ is a $p \times k$ matrix.

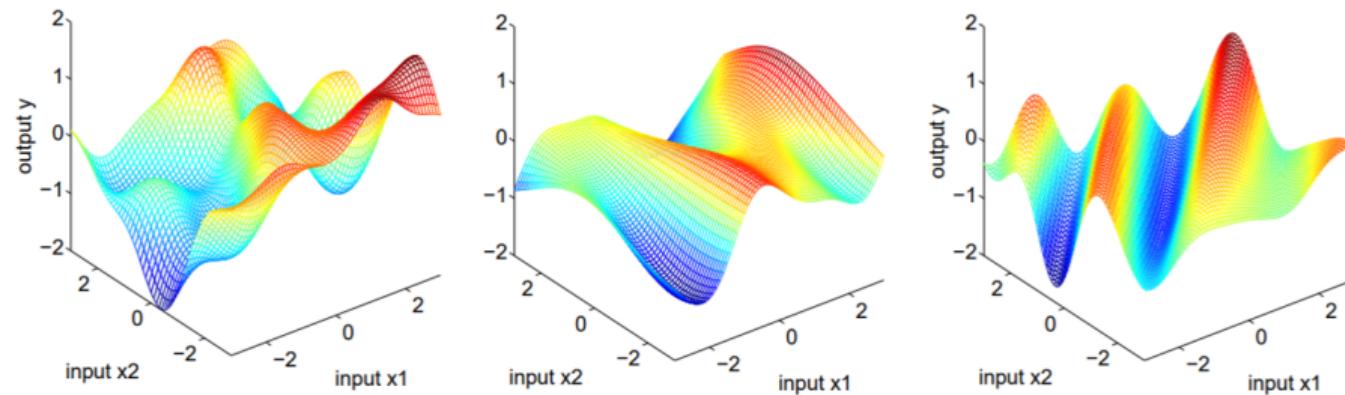
Here again, $\ell = (\ell_1, \dots, \ell_p)$ are characteristic length-scales for each dimension.

Characteristic Length-Scale III

What is the benefit of having an individual hyperparameter ℓ_i for each dimension?

- The ℓ_1, \dots, ℓ_p hyperparameters play the role of **characteristic length-scales**.
- Loosely speaking, ℓ_i describes how far you need to move along axis i in input space for the function values to be uncorrelated.
- Such a covariance function implements **automatic relevance determination** (ARD), since the inverse of the length-scale ℓ_i determines the relevancy of input feature i to the regression.
- If ℓ_i is very large, the covariance will become almost independent of that input, effectively removing it from inference.
- If the features are on different scales, the data can be automatically **rescaled** by estimating ℓ_1, \dots, ℓ_p .

Characteristic Length-Scale IV



For the first plot, we have chosen $\mathbf{M} = \mathbf{I}$: the function varies the same in all directions. The second plot is for $\mathbf{M} = \text{diag}(\ell)^{-2}$ and $\ell = (1, 3)$: The function varies less rapidly as a function of x_2 than x_1 as the length-scale for x_1 is less. In the third plot $\mathbf{M} = \Gamma\Gamma^T + \text{diag}(\ell)^{-2}$ for $\Gamma = (1, -1)^\top$ and $\ell = (6, 6)^\top$. Here Γ gives the direction of the most rapid variation.
[Rasmussen and Williams. 2006]

AutoML: Gaussian Processes

Gaussian Process Prediction

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Motivation

- So far, we have learned how to **sample** from a Gaussian process prior.
- However, most of the time, we are not interested in drawing random functions from the prior. Instead, we usually like to use the knowledge provided by the training data to predict values of f at a new test point \mathbf{x}_* .
- In what follows, we will investigate how to update the Gaussian process prior (\rightarrow posterior process) and how to make predictions.

Gaussian Posterior Process and Prediction

Posterior Process I

- Let us distinguish between **observed training** inputs (also denoted by a design matrix \mathbf{X}), their corresponding values

$$\mathbf{f} = \left[f\left(\mathbf{x}^{(1)}\right), \dots, f\left(\mathbf{x}^{(n)}\right) \right],$$

and one single **unobserved test point** \mathbf{x}_* with $f_* = f(\mathbf{x}_*)$.

- We now want to infer the distribution of $f_* | \mathbf{x}_*, \mathbf{X}, \mathbf{f}$.
- Assuming a zero-mean GP prior $\mathcal{G}(\mathbf{0}, k(\mathbf{x}, \mathbf{x}'))$, we can assert that

$$\begin{bmatrix} \mathbf{f} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k}_* \\ \mathbf{k}_*^T & \mathbf{k}_{**} \end{bmatrix}\right),$$

where, $\mathbf{K} = (k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}))_{i,j}$, $\mathbf{k}_* = [k(\mathbf{x}_*, \mathbf{x}^{(1)}), \dots, k(\mathbf{x}_*, \mathbf{x}^{(n)})]$ and $\mathbf{k}_{**} = k(\mathbf{x}_*, \mathbf{x}_*)$.

Posterior Process II

(*) A General Rule of Conditioning for Gaussian Random Variables

If the m -dimensional Gaussian vector $\mathbf{z} \sim \mathcal{N}(\mu, \Sigma)$ can be partitioned with $\mathbf{z} = (\mathbf{z}_1, \mathbf{z}_2)$ where \mathbf{z}_1 is m_1 -dimensional and \mathbf{z}_2 is m_2 -dimensional, and:

$$(\mu_1, \mu_2), \quad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix},$$

then the conditional distribution $\mathbf{a} = \mathbf{z}_2 \mid \mathbf{z}_1$ will be a multivariate normal distribution:

$$\mathcal{N}(\mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(\mathbf{a} - \mu_1), \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12})$$

Posterior Process III

- Given that f is observed, we can exploit the general rule (*) to obtain the following formula:

$$f_* \mid \mathbf{x}_*, \mathbf{X}, f \sim \mathcal{N}(\mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{f}, \mathbf{k}_{**} - \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{k}_*).$$

- As the posterior is Gaussian, the maximum a-posteriori estimate (i.e., the mode of the posterior distribution) is:

$$\mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{f}.$$

GP Prediction: Two Points I

To visualize the above idea, assume that we have observed a single training point $\mathbf{x} = -0.5$. Based on this point, we intend to make a prediction at the test point $\mathbf{x}_* = 0.5$.

- Under a zero-mean \mathcal{G} with $k(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^2)$, we compute the cov-matrix:

$$\begin{bmatrix} f \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} 1 & 0.61 \\ 0.61 & 1 \end{bmatrix}\right).$$

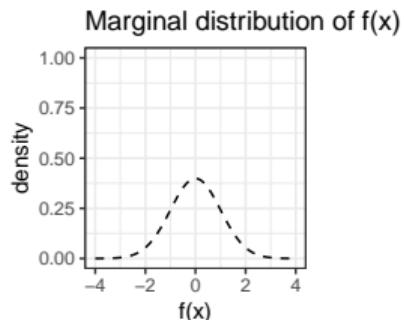
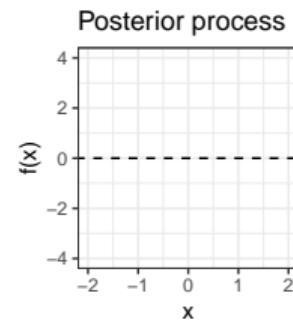
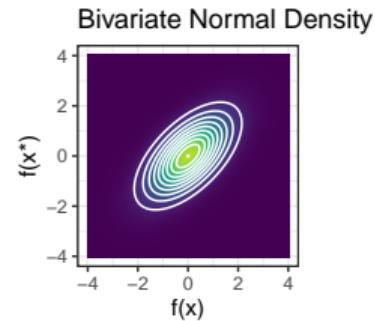
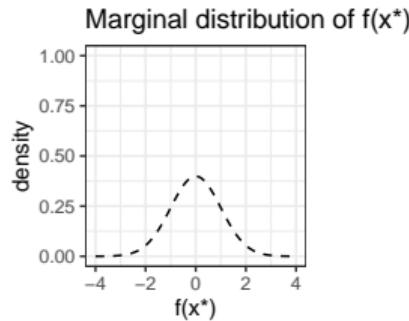
- Let us assume that we observe the point $f(\mathbf{x}) = 1$. We can compute the posterior distribution:

$$\begin{aligned} f_* \mid \mathbf{x}_*, \mathbf{x}, f &\sim \mathcal{N}(\mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{f}, k_{**} - \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{k}_*) \\ &\sim \mathcal{N}(0.61 \cdot 1 \cdot 1, 1 - 0.61 \cdot 1 \cdot 0.61) \\ &\sim \mathcal{N}(0.61, 0.6279) \end{aligned}$$

- The MAP-estimate for \mathbf{x}_* is $f(\mathbf{x}_*) = 0.61$, and the uncertainty estimate is 0.6279.

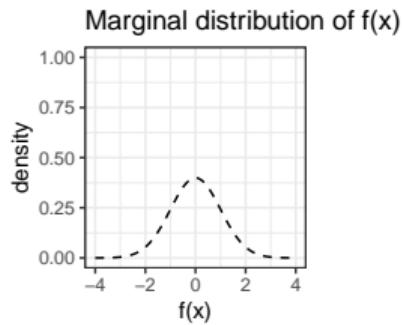
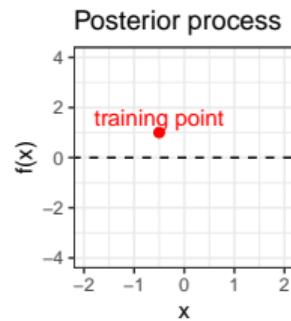
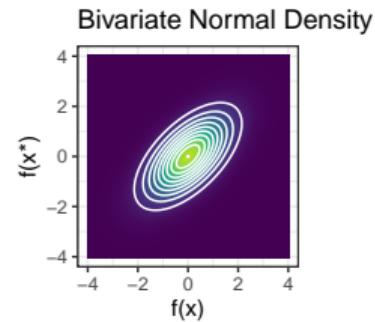
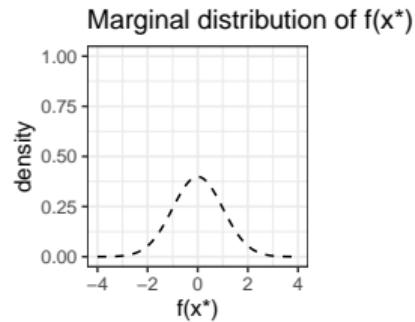
GP Prediction: Two Points II

The figures show the bivariate normal density as well as the corresponding marginals.



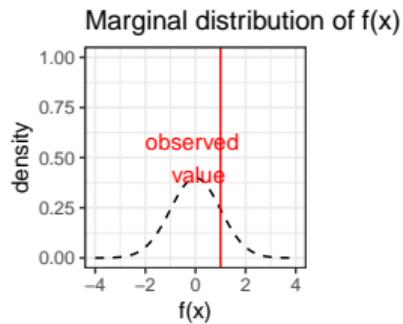
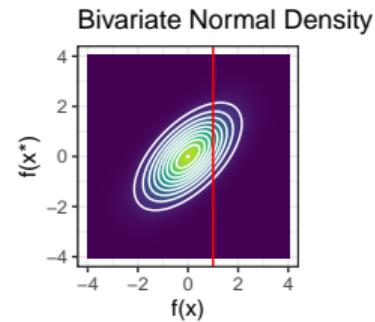
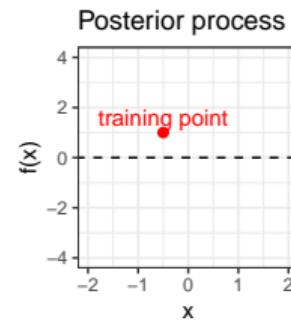
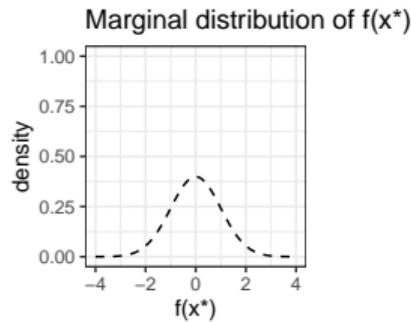
GP Prediction: Two Points III

We observe $f(x) = 1$ at training point $x = -0.5$.



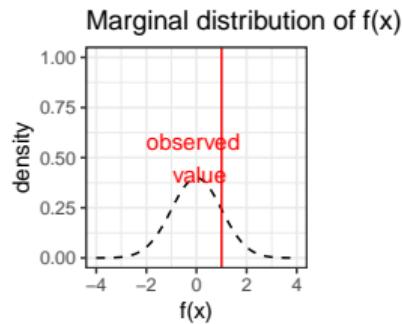
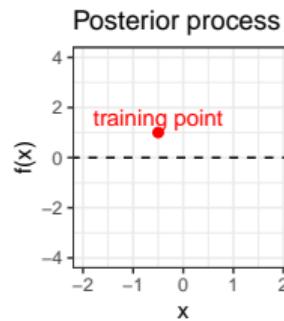
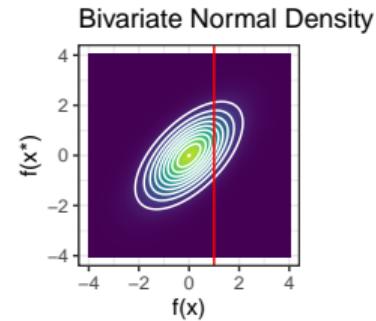
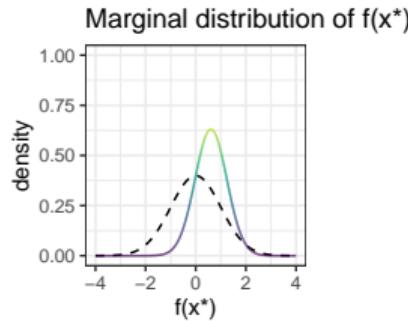
GP Prediction: Two Points IV

We condition the Gaussian on $f(\mathbf{x}) = 1$.



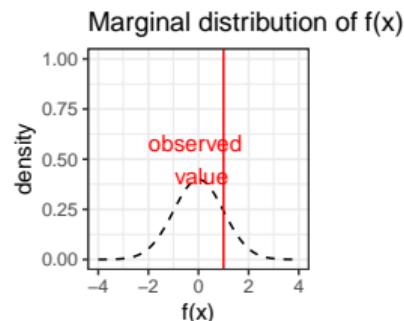
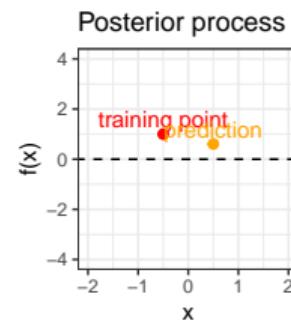
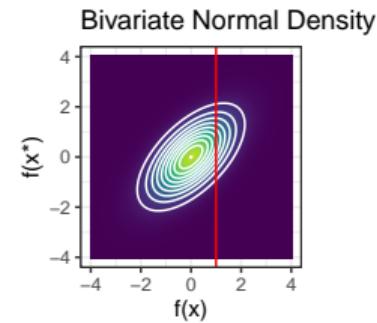
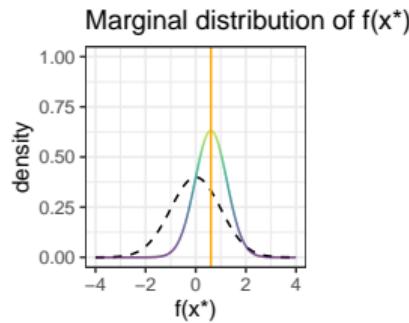
GP Prediction: Two Points V

We then compute the posterior distribution of $f(\mathbf{x}_*)$ given that $f(\mathbf{x}) = 1$.



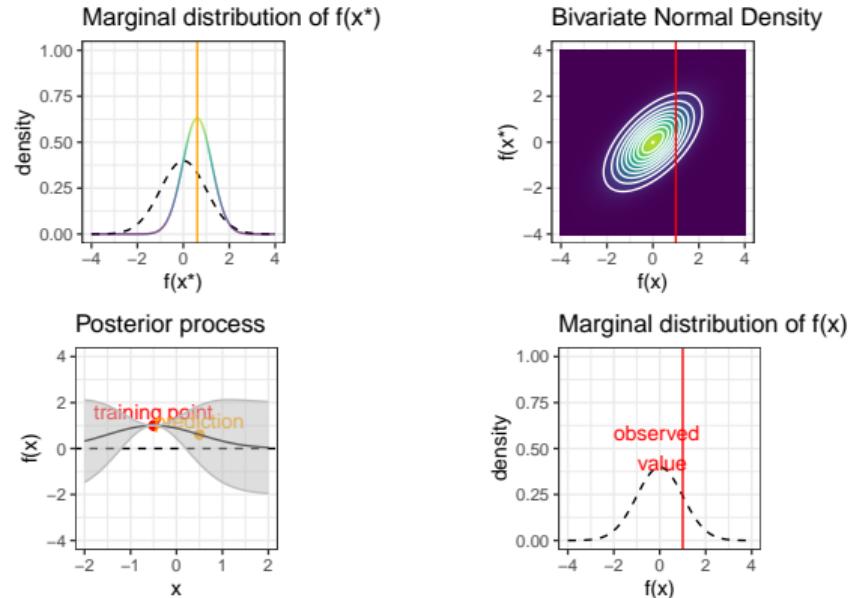
GP Prediction: Two Points VI

A possible predictor for f at x_* is the MAP of the posterior distribution.



GP Prediction: Two Points VII

We can repeat this process for different x_* and find the respective mean (grey line) and standard deviation (grey area). Note that the grey area is mean $\pm 2 \times$ standard deviation.



Posterior Process I

- The previous discussion was restricted to a single test point. However, one can generalize it to posterior processes with multiple unobserved test points:

$$\mathbf{f}_* = \left[f\left(\mathbf{x}_*^{(1)}\right), \dots, f\left(\mathbf{x}_*^{(m)}\right) \right].$$

- Under a zero-mean Gaussian process, we have:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix}\right),$$

where $\mathbf{K}_* = \left(k\left(\mathbf{x}^{(i)}, \mathbf{x}_*^{(j)}\right)\right)_{i,j}$ and $\mathbf{K}_{**} = \left(k\left(\mathbf{x}_*^{(i)}, \mathbf{x}_*^{(j)}\right)\right)_{i,j}$

Posterior Process II

- Similar to the single test point situation, to get the posterior distribution, we exploit the general rule of conditioning for Gaussians:

$$\mathbf{f}_* \mid \mathbf{X}_*, \mathbf{X}, \mathbf{f} \sim \mathcal{N}(\mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{f}, \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_*).$$

- This formula enables us to talk about correlations among different test points and sample functions from the posterior process.

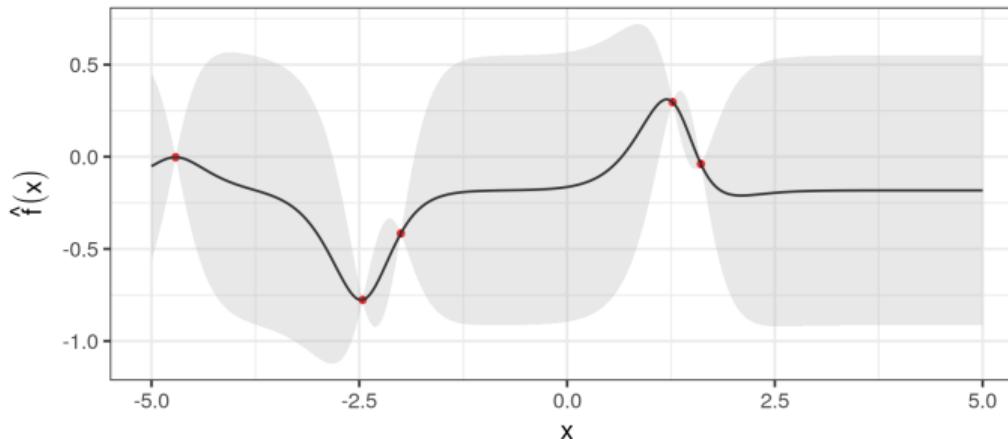
Properties of a Gaussian Process

GP as an Interpolator

- The “prediction” for a training point $\mathbf{x}^{(i)}$ is the exact function value $f(\mathbf{x}^{(i)})$. That is,

$$f \mid \mathbf{X}, f \sim \mathcal{N}(\mathbf{K}\mathbf{K}^{-1}\mathbf{f}, \mathbf{K} - \mathbf{K}^T\mathbf{K}^{-1}\mathbf{K}) = \mathcal{N}(\mathbf{f}, \mathbf{0}).$$

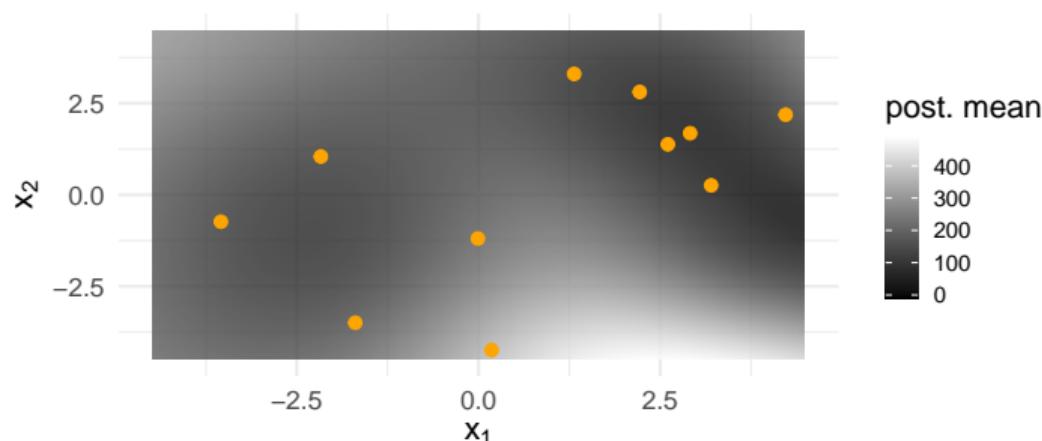
- Thus, a Gaussian process is a function **interpolator**.



After observing the training points (red), the posterior process (black) interpolates the training points.
 $(k(x, x'))$ is Matérn with $\text{nu} = 2.5$, the default for DiceKriging::km)

GP as a Spatial Model I

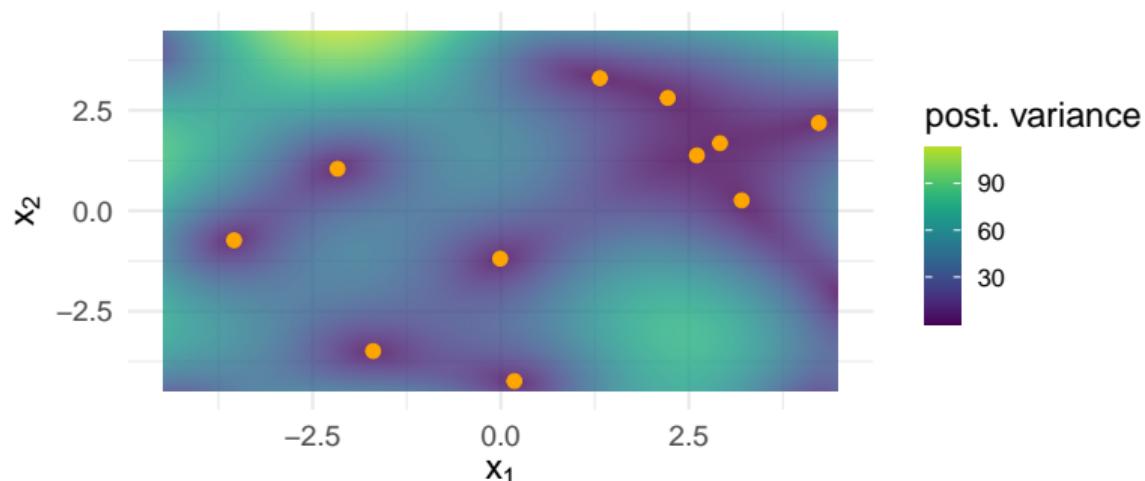
- The correlation among two outputs depends on the distance of the corresponding input points \mathbf{x} and \mathbf{x}' . For instance, the Gaussian covariance kernel is $k(\mathbf{x}, \mathbf{x}') = \exp\left(\frac{-\|\mathbf{x}-\mathbf{x}'\|^2}{2\ell^2}\right)$.
- Hence, close data points with high spatial similarity $k(\mathbf{x}, \mathbf{x}')$ enter into more strongly correlated predictions: $\mathbf{k}_*^\top \mathbf{K}^{-1} \mathbf{f}$ ($\mathbf{k}_* := (k(\mathbf{x}, \mathbf{x}^{(1)}), \dots, k(\mathbf{x}, \mathbf{x}^{(n)}))$).



Example: the posterior mean of a GP that is fitted with the Gaussian covariance kernel with $\ell = 1$.

GP as a Spatial Model II

- Posterior uncertainty increases if the new data points are far from the design points.
- The uncertainty is minimal at the design points, since the posterior variance is zero at these points.

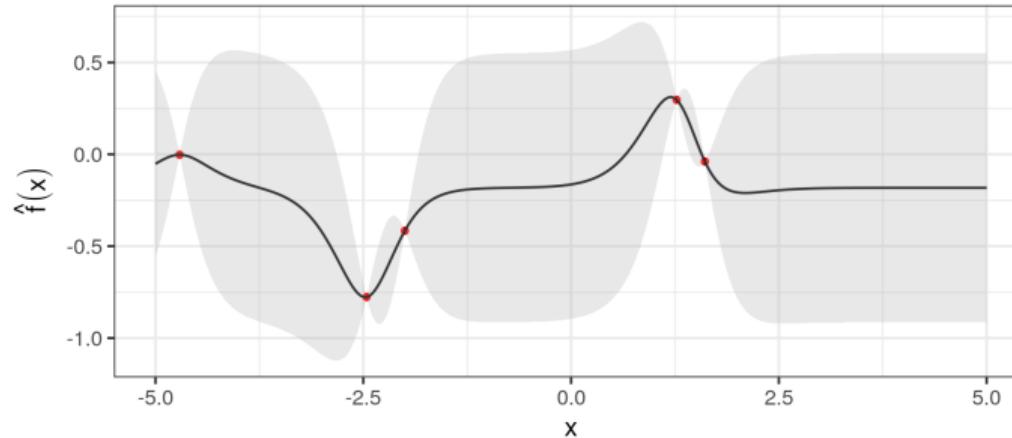


Example (continued): posterior variance

Noisy Gaussian Process

Noisy Gaussian Process I

- So far, we have implicitly assumed that we access the true function values $f(\mathbf{x})$.
- For the squared exponential kernel, for example, we had $\text{cov}(f(\mathbf{x}^{(i)}), f(\mathbf{x}^{(j)})) = 1$.
- Consequently, the posterior Gaussian process was an interpolator.



After observing the training points (red), the posterior process (black) interpolates the training points.
 $(k(x, x'))$ is Matérn with $\nu = 2.5$, the default for DiceKriging::km

Noisy Gaussian Process II

- However, in reality that is not often the case. Rather, we often only have access to a noisy version of the true function values:

$$y = f(\mathbf{x}) + \epsilon, \text{ where } \epsilon \sim \mathcal{N}(0, \sigma^2).$$

- Let us assume that $f(\mathbf{x})$ is still a Gaussian process. Then, we would have the following:

$$\begin{aligned} \text{cov}(y^{(i)}, y^{(j)}) &= \text{cov}\left(f\left(\mathbf{x}^{(i)}\right) + \epsilon^{(i)}, f\left(\mathbf{x}^{(j)}\right) + \epsilon^{(j)}\right) \\ &= \text{cov}\left(f\left(\mathbf{x}^{(i)}\right), f\left(\mathbf{x}^{(j)}\right)\right) + 2 \cdot \text{cov}\left(f\left(\mathbf{x}^{(i)}\right), \epsilon^{(j)}\right) + \text{cov}\left(\epsilon^{(i)}, \epsilon^{(j)}\right) \\ &= k\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) + \sigma^2 \delta_{ij}. \end{aligned}$$

- σ^2 is called **nugget**.

Noisy Gaussian Process III

- We can now derive the predictive distribution for the case of noisy observations.
- Assuming that f is modeled by a Gaussian process, the prior distribution of y is

$$\mathbf{y} = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{pmatrix} \sim \mathcal{N}(\mathbf{m}, \mathbf{K} + \sigma^2 I_n),$$

with

$$\mathbf{m} := \left(m(\mathbf{x}^{(i)}) \right)_i, \quad \mathbf{K} := \left(k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \right)_{i,j}.$$

Noisy Gaussian Process IV

We distinguish again between:

- Observed training points and their corresponding values, i.e, \mathbf{X} and \mathbf{y} .
- Unobserved test points and their corresponding values, i.e, \mathbf{X}_* and f_* .

and get:

$$\begin{bmatrix} \mathbf{y} \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} + \sigma^2 I_n & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{bmatrix}\right).$$

Noisy Gaussian Process V

- Similar to the noise-free case, we condition according to the rule of conditioning for Gaussians to get the posterior distribution for the test outputs f_* at \mathbf{X}_* :

$$f_* \mid \mathbf{X}_*, \mathbf{X}, \mathbf{y} \sim \mathcal{N}(\mathbf{m}_{\text{post}}, \mathbf{K}_{\text{post}}),$$

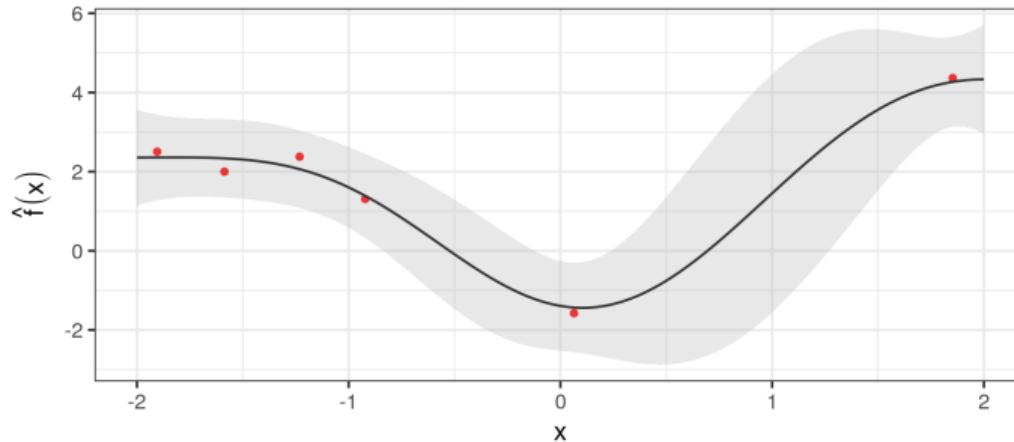
with

$$\begin{aligned}\mathbf{m}_{\text{post}} &= \mathbf{K}_*^T (\mathbf{K} + \sigma^2 \cdot \mathbf{I})^{-1} \mathbf{y} \\ \mathbf{K}_{\text{post}} &= \mathbf{K}_{**} - \mathbf{K}_*^T (\mathbf{K}^{-1} + \sigma^2 \cdot \mathbf{I}) \mathbf{K}_*.\end{aligned}$$

- This converts back to the noise-free formula if $\sigma^2 = 0$.

Noisy Gaussian Process VI

- The noisy Gaussian process is not an interpolator any more.
- A larger nugget term leads to a wider “band” around the observed training points.
- The nugget term is estimated during training.



After observing the training points (red), we have a nugget-band around the observed points.
($k(x,x')$ is the squared exponential)

AutoML: Gaussian Processes

Gaussian Process Training

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Training of a Gaussian Process

- To make predictions for a regression task by a Gaussian process, one simply needs to perform matrix computations.
- But for this to work out, we assume that the covariance functions is fully given, including all of its hyperparameters.
- A very nice property of GPs is that we can learn the numerical hyperparameters of a selected covariance function directly during GP training.

Training a GP via the Maximum Likelihood I

- Let us assume $y = f(\mathbf{x}) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, where $f(\mathbf{x}) \sim \mathcal{G}(\mathbf{0}, k(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}))$.
- Noticing that $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \mathbf{K} + \sigma^2 \mathbf{I})$, we can find the marginal log-likelihood (or evidence):

$$\begin{aligned}\log p(\mathbf{y} \mid \mathbf{X}, \boldsymbol{\theta}) &= \log \left[(2\pi)^{-n/2} |\mathbf{K}_y|^{-1/2} \exp \left(-\frac{1}{2} \mathbf{y}^\top \mathbf{K}_y^{-1} \mathbf{y} \right) \right] \\ &= -\frac{1}{2} \mathbf{y}^\top \mathbf{K}_y^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}_y| - \frac{n}{2} \log 2\pi.\end{aligned}$$

with $\mathbf{K}_y := \mathbf{K} + \sigma^2 \mathbf{I}$ and $\boldsymbol{\theta}$ denoting the parameters of the covariance function (i.e., the hyperparameters).

Training a GP via the Maximum Likelihood II

Recalling that the increase of the length-scale reduces the model flexibility, the three terms of the marginal likelihood can be interpreted as follows.

- The data fit $-\frac{1}{2}\mathbf{y}^T \mathbf{K}_y^{-1} \mathbf{y}$. The data fit tends to decrease by increasing the length-scale.
- The complexity penalty $-\frac{1}{2} \log |\mathbf{K}_y|$, which depends on the covariance function. This term decreases with the increase of the length-scale (the model gets less complex as the length-scale grows).
- The normalization constant $-\frac{n}{2} \log 2\pi$.

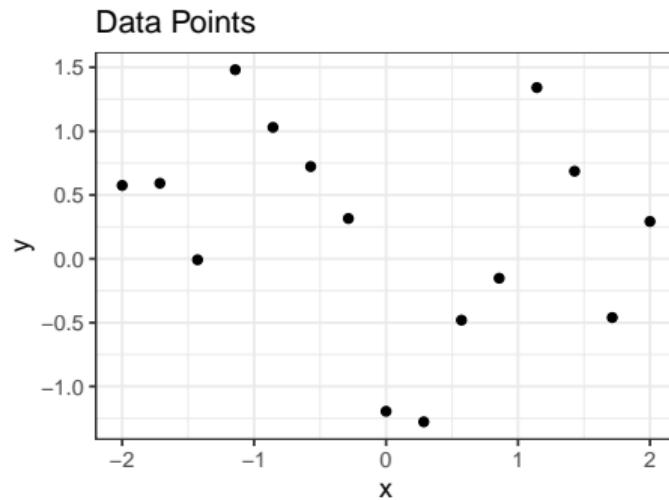
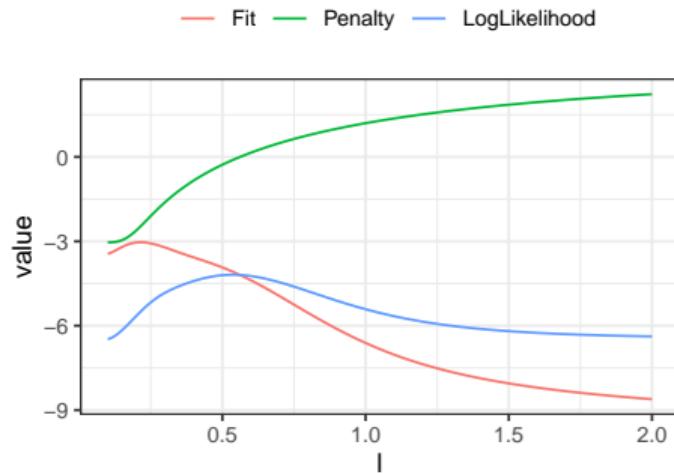
Training a GP: Example I

To visualize this, let us consider a zero-mean GP with a squared exponential kernel:

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2\ell^2}\|\mathbf{x} - \mathbf{x}'\|^2\right).$$

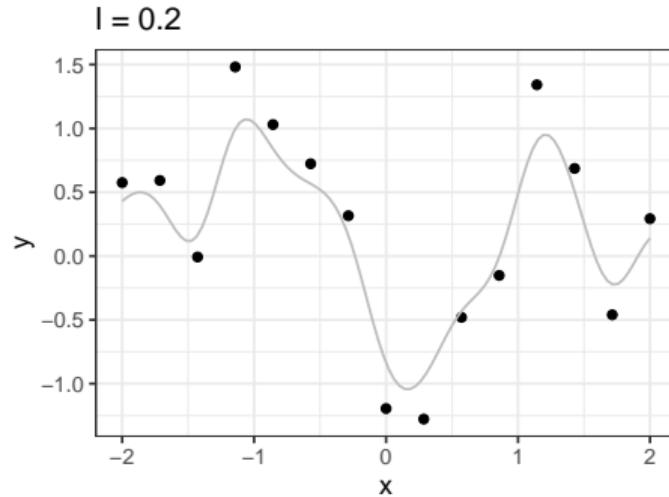
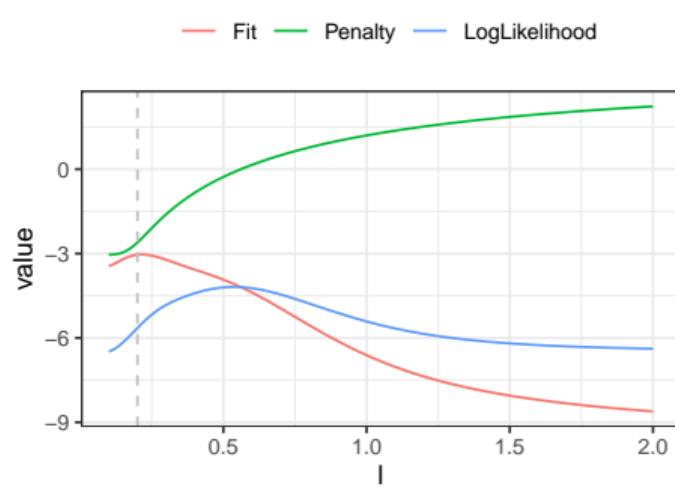
- Recall that the model becomes smoother and less complex as the length-scale ℓ increases.
- We will show how each of the following terms behaves if the value of ℓ increases:
 - ▶ the data fit $-\frac{1}{2}\mathbf{y}^\top \mathbf{K}_y^{-1} \mathbf{y}$,
 - ▶ the complexity penalty $-\frac{1}{2} \log |\mathbf{K}_y|$,
 - ▶ the overall value of the marginal likelihood $\log p(\mathbf{y} \mid \mathbf{X}, \theta)$.

Training a GP: Example II



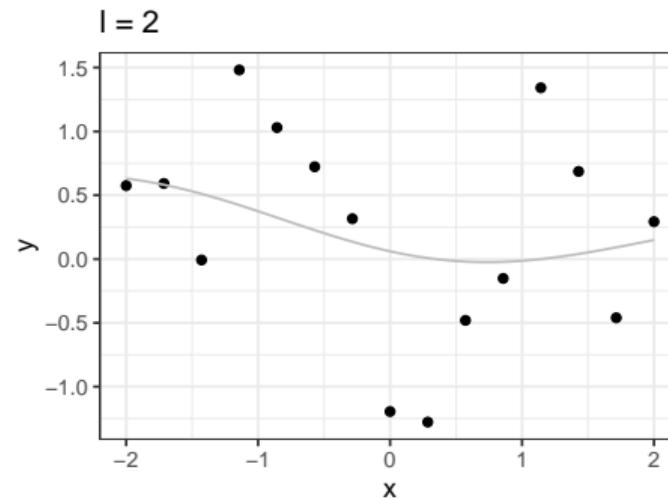
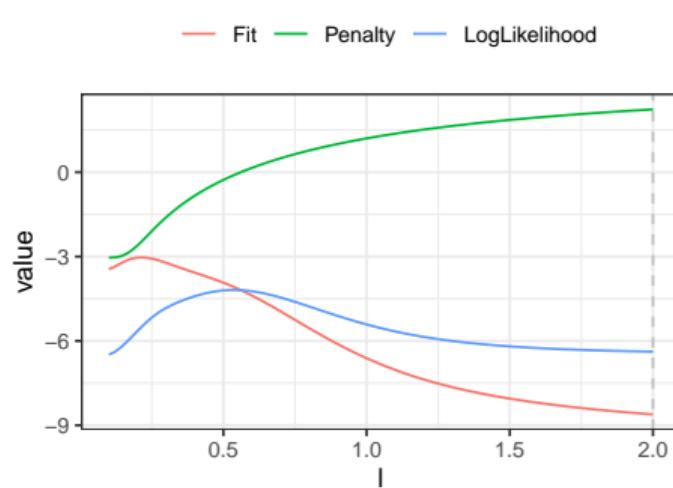
- The left plot depicts how the data fit, the complexity penalty (a higher value means less penalization), and the overall marginal likelihood behave for increasing values of the length-scale.

Training a GP: Example III



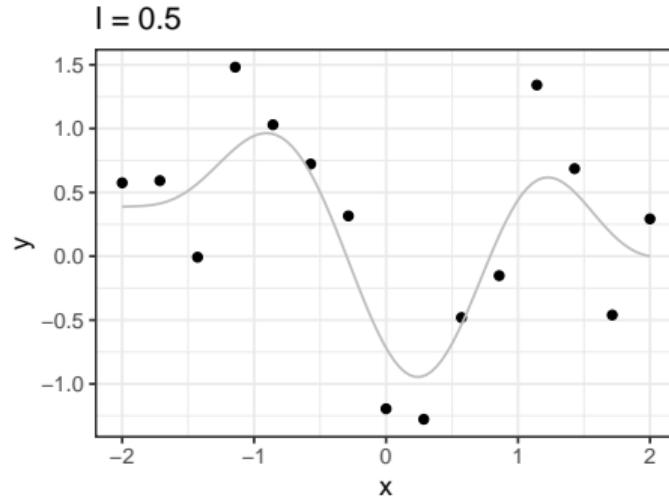
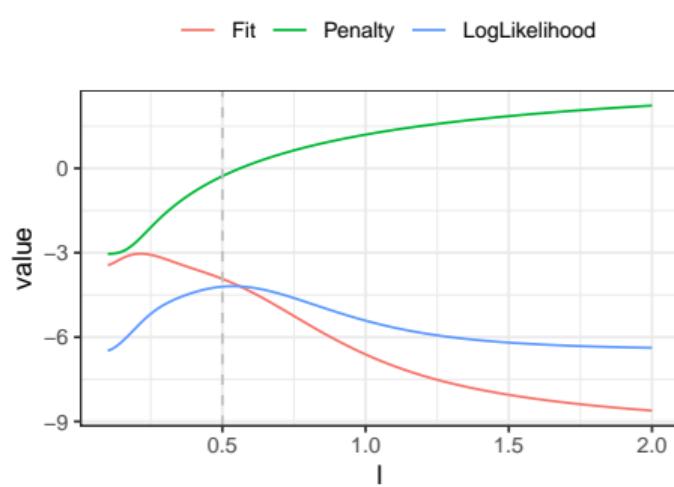
- ⓘ The left plot depicts how the data fit, the complexity penalty (a higher value means less penalization), and the overall marginal likelihood behave for increasing values of the length-scale.
- 💡 A small ℓ leads to a good fit, but, to a high complexity penalty.

Training a GP: Example IV



- 💡 The left plot depicts how the data fit, the complexity penalty (a higher value means less penalization), and the overall marginal likelihood behave for increasing values of the length-scale.
- 💡 A large ℓ results in a poor fit.

Training a GP: Example V



- i** The left plot depicts how the data fit, the complexity penalty (a higher value means less penalization), and the overall marginal likelihood behave for increasing values of the length-scale.
- 💡** The maximizer of the log-likelihood ($\ell = 0.5$) balances the complexity and data the fit.

Training a GP via the Maximum Likelihood I

To choose the hyperparameters by maximizing the marginal likelihood, we need to find the partial derivatives of the likelihood w.r.t. the hyperparameters:

$$\begin{aligned}\frac{\partial}{\partial \theta_j} \log p(\mathbf{y} | \mathbf{X}, \boldsymbol{\theta}) &= \frac{\partial}{\partial \theta_j} \left(-\frac{1}{2} \mathbf{y}^\top \mathbf{K}_y^{-1} \mathbf{y} - \frac{1}{2} \log |\mathbf{K}_y| - \frac{n}{2} \log 2\pi \right) \\ &= \frac{1}{2} \mathbf{y}^\top \mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta_j} \mathbf{K}^{-1} \mathbf{y} - \frac{1}{2} \text{tr} \left(\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \boldsymbol{\theta}} \right) \\ &= \frac{1}{2} \text{tr} \left((\mathbf{K}^{-1} \mathbf{y} \mathbf{y}^\top \mathbf{K}^{-1} - \mathbf{K}^{-1}) \frac{\partial \mathbf{K}}{\partial \theta_j} \right)\end{aligned}$$

💡 Above, we used the following identities:

$$\frac{\partial}{\partial \theta_j} \mathbf{K}^{-1} = -\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \theta_j} \mathbf{K}^{-1} \quad \text{and} \quad \frac{\partial}{\partial \boldsymbol{\theta}} \log |\mathbf{K}| = \text{tr} \left(\mathbf{K}^{-1} \frac{\partial \mathbf{K}}{\partial \boldsymbol{\theta}} \right)$$

Training a GP via the Maximum Likelihood II

- The complexity and the runtime of training a Gaussian process is dominated by the computational task of inverting \mathbf{K} - or let's rather say for decomposing it.
 - Standard methods require $\mathcal{O}(n^3)$ time (!) for this.
 - Once \mathbf{K}^{-1} - or rather the decomposition -is known, the computation of the partial derivatives requires only $\mathcal{O}(n^2)$ time per hyperparameter.
- 💡 Thus, the computational overhead of computing derivatives is small, and using a gradient based optimizer is advantageous.

Training a GP via the Maximum Likelihood III

Workarounds to make GP estimation feasible for big data include:

- Using kernels that yield sparse \mathbf{K} : cheaper to invert.
- Subsampling the data to estimate θ ; $\mathcal{O}(m^3)$ for subset of size m .
- Combining estimates on different subsets of size m : **Bayesian committee**; $\mathcal{O}(nm^2)$.
- Exploiting low-rank approximations of \mathbf{K} by using only a representative subset (inducing points) of m training data \mathbf{X}_m : **Nyström approximation** $\mathbf{K} \approx \mathbf{K}_{nm}\mathbf{K}_{mm}^{-}\mathbf{K}_{mn}$, with $\mathcal{O}(nmk + m^3)$ for a rank-k-approximate inverse of \mathbf{K}_{mm} .
- Utilizing structure in \mathbf{K} induced by the kernel: exact solutions but complicated maths, not applicable for all kernels.

... this is still an active area of research.

AutoML: Gaussian Processes

Mean Functions for Gaussian Processes

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

The Role of Mean Functions I

- It is common but by no means necessary to consider GPs with zero-mean functions:

$$m(\mathbf{x}) \equiv 0$$

- This is not necessarily a drastic limitation, since the mean of the posterior process is not confined to be zero:

$$\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{f} \sim \mathcal{N}(\mathbf{K}_*^\top \mathbf{K}^{-1} \mathbf{f}, \mathbf{K}_{**} - \mathbf{K}_*^\top \mathbf{K}^{-1} \mathbf{K}_*).$$

- There are several reasons why one might wish to explicitly model a mean function, including interpretability, convenience of expressing prior informations, etc.

The Role of Mean Functions II

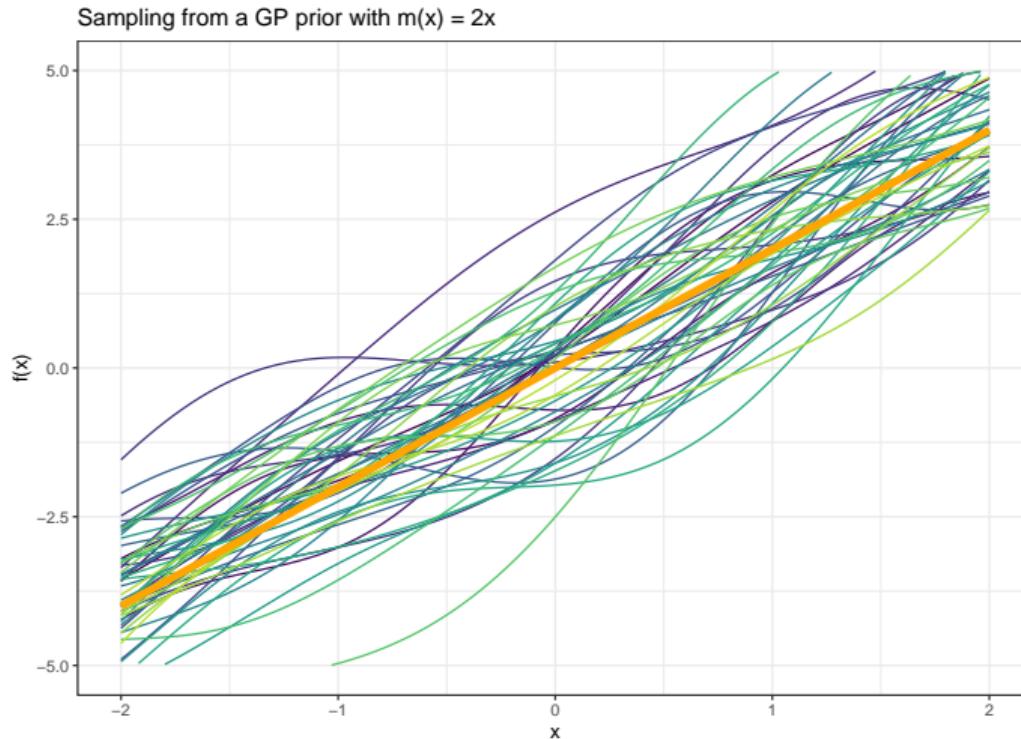
- When assuming a non-zero mean GP prior $\mathcal{G}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}'))$ with mean $m(\mathbf{x})$, the predictive mean becomes:

$$m(\mathbf{X}_*) + \mathbf{K}_* \mathbf{K}_y^{-1} (\mathbf{y} - m(\mathbf{X}))$$

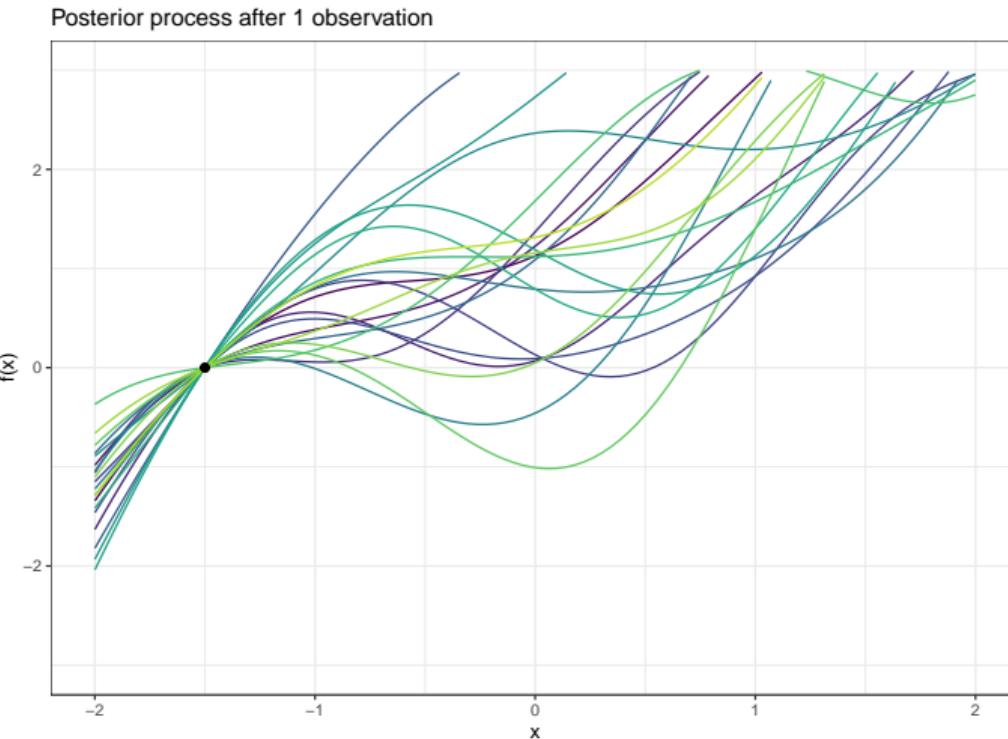
but the predictive variance remains unchanged.

- Gaussian processes with non-zero mean Gaussian process priors are also called **Gaussian processes with trend**.

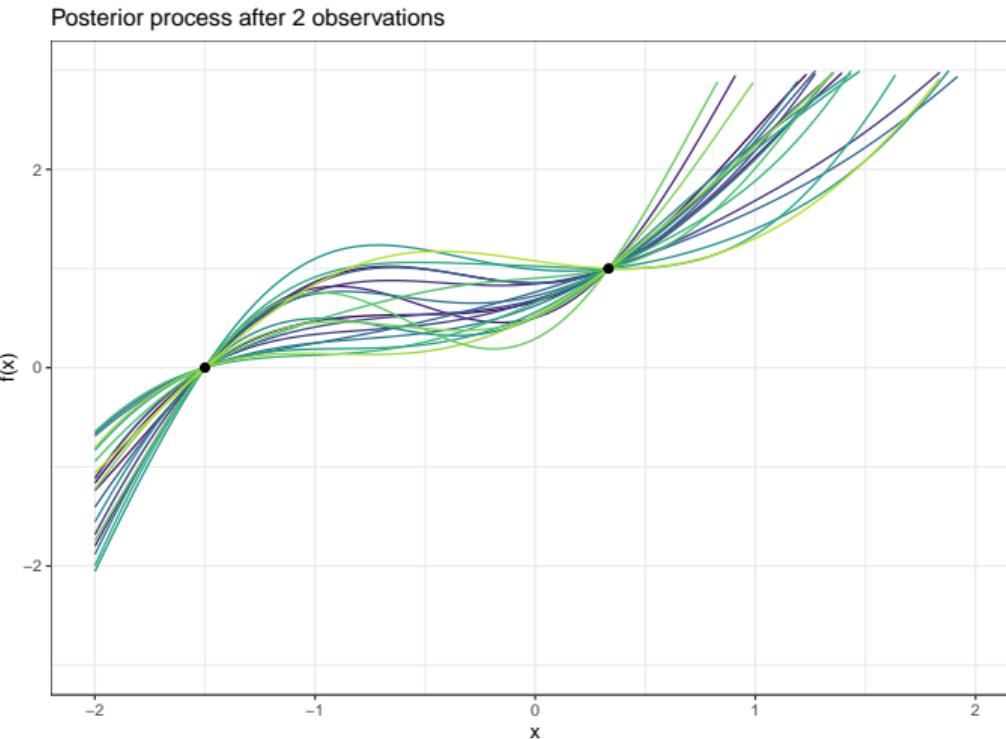
The Role of Mean Functions III



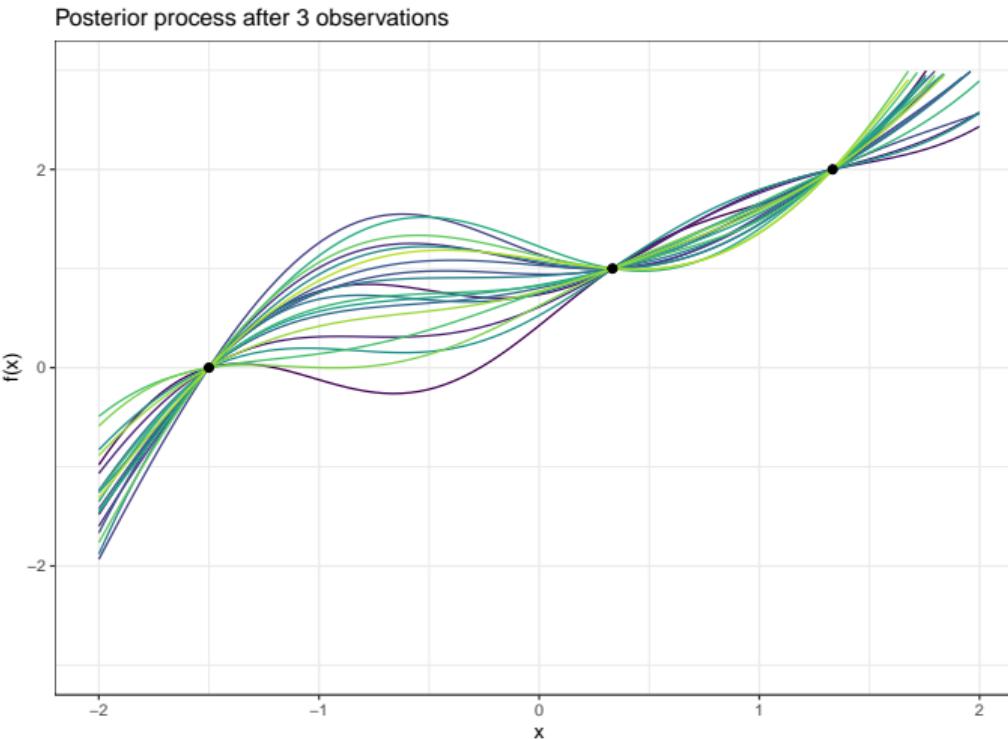
The Role of Mean Functions IV



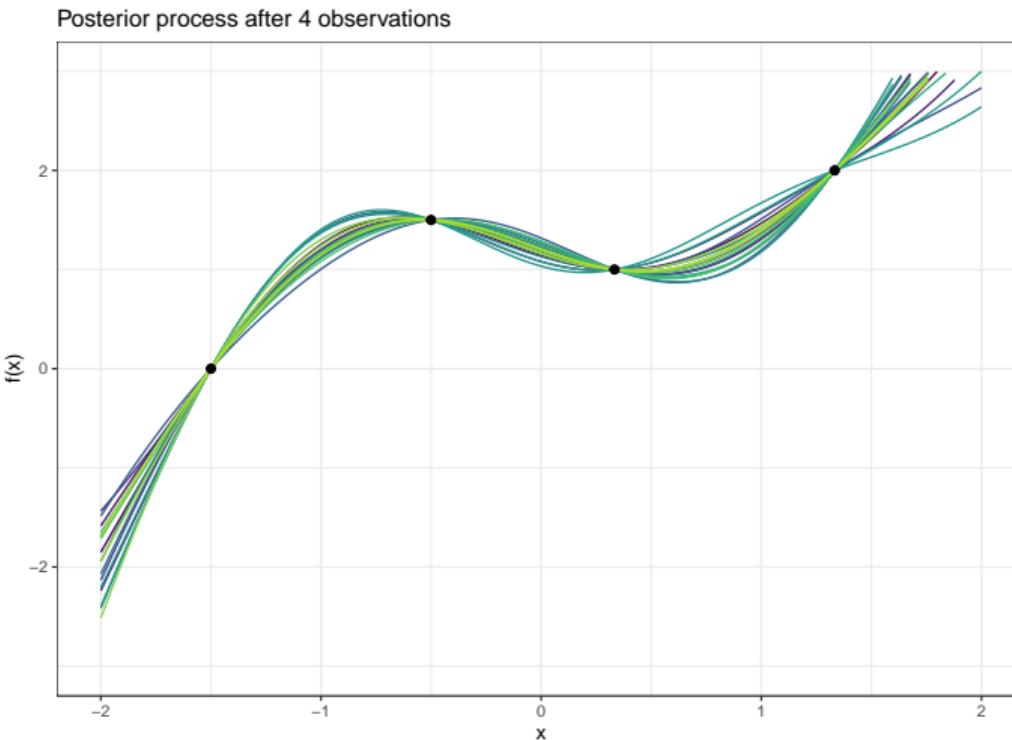
The Role of Mean Functions V



The Role of Mean Functions VI



The Role of Mean Functions VII



The Role of Mean Functions VIII

- In practice it is often difficult to specify a fixed mean function. Instead, it may be more convenient to specify a few fixed basis functions, whose coefficients, β , are to be inferred from the data.
- Consider

$$g(\mathbf{x}) = f(\mathbf{x}) + \mathbf{h}(\mathbf{x})^\top \boldsymbol{\beta}, \text{ with } f(\mathbf{x}) \sim \mathcal{G}(0, k(\mathbf{x}, \mathbf{x}'))$$

where $f(\mathbf{x})$ is a zero mean GP, $\mathbf{h}(\mathbf{x})$ are a set of fixed basis functions, and $\boldsymbol{\beta}$ are additional parameters.

- This formulation expresses that the data is close to a global linear model with the residuals being modelled by a GP.
- By taking the prior on $\boldsymbol{\beta}$ to be Gaussian, $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{b}, B)$, another GP with an added contribution in the covariance function can be obtained:

$$g(\mathbf{x}) \sim \mathcal{G}\left(\mathbf{h}(\mathbf{x})^\top \mathbf{b}, k(\mathbf{x}, \mathbf{x}') + \mathbf{h}(\mathbf{x})^\top B \mathbf{h}(\mathbf{x}')\right)$$

The Role of Mean Functions IX

- The mean and covariance functions of $g(\mathbf{x})$ will be then:

$$\bar{\mathbf{g}}(X_*) = \bar{\mathbf{f}}(X_*) + R^\top \bar{\boldsymbol{\beta}},$$

$$\text{cov}(\mathbf{g}_*) = \text{cov}(\mathbf{f}_*) + R^\top (B^{-1} + HK_y^{-1}H^\top)^{-1}R,$$

where $\bar{\boldsymbol{\beta}} = (B^{-1} + HK_y^{-1}H^\top)^{-1}(HK_y^{-1}\mathbf{y} + B^{-1}\mathbf{b})$ and $R = H_* - HK_y^{-1}K_*$.

- Notice that $\bar{\boldsymbol{\beta}}$ is the mean of the global linear model parameters and the H matrix collects the $\mathbf{h}(\mathbf{x})$ vectors for all training and test cases.
- Hence, the predictive mean is the sum of the mean linear output and what the GP model predicts from the residuals. The covariance is the usual covariance term plus a non negative contribution.

The Role of Mean Functions X

- In the limiting case where the prior on the β becomes vague ($B^{-1} \rightarrow O$), the predictive distribution becomes independent of b :

$$\bar{\mathbf{g}}(X_*) = \bar{\mathbf{f}}(X_*) + R^\top \bar{\beta},$$

$$\text{cov}(\mathbf{g}_*) = \text{cov}(\mathbf{f}_*) + R^\top (HK_y^{-1}H^\top)^{-1}R,$$

where the limiting $\bar{\beta} = (HK_y^{-1}H^\top)^{-1}HK_y^{-1}\mathbf{y}$.

- To make predictions under the limit $B^{-1} \rightarrow O$, it is important to use the above equation. Otherwise, by naively plugging the modified covariance function into the standard prediction equations, the entries of the covariance function tend to infinity making it unsuitable for numerical implementation.

The Role of Mean Functions XI

- The marginal likelihood for the model with a Gaussian prior $\beta \sim \mathcal{N}(\mathbf{b}, B)$ can be expressed as what follows (the explicit mean has been included).

$$\begin{aligned}\log p(\mathbf{y}|X, \mathbf{b}, B) = & -\frac{1}{2} \left(H^\top \mathbf{b} - \mathbf{y} \right)^\top \left(K_y + H^\top B H \right)^{-1} \left(H^\top \mathbf{b} - \mathbf{y} \right) \\ & - \frac{1}{2} \log |K_y + H^\top B H| - \frac{n}{2} \log 2\pi.\end{aligned}$$

- We are interested in exploring the limit where $B^{-1} \rightarrow O$, i.e. when the prior is vague. In this limit the mean of the prior is irrelevant, so without loss of generality we assume for now that the mean is zero, $\mathbf{b} = \mathbf{0}$.

The Role of Mean Functions XII

- This assumption gives:

$$\begin{aligned}\log p(\mathbf{y}|X, \mathbf{b} = \mathbf{0}, B) = & -\frac{1}{2}\mathbf{y}^\top K_y^{-1}\mathbf{y} + \frac{1}{2}\mathbf{y}^\top C\mathbf{y} \\ & -\frac{1}{2}\log|K_y| - \frac{1}{2}\log|B| - \frac{1}{2}\log|A| - \frac{n}{2}\log 2\pi,\end{aligned}$$

where $A = B^{-1} + HK_y^{-1}H^\top$ and $C = K_y^{-1}H^\top A^{-1}HK_y^{-1}$.

- The behaviour of the log marginal likelihood in the limiting case can be explored now. The variances of the Gaussian in the directions spanned by columns of H^\top will become infinite, and it will require special treatment.
- 💡 The log marginal likelihood consists of three terms: a quadratic form in \mathbf{y} , a log determinant term, and a term involving $\log 2\pi$.

The Role of Mean Functions XIII

- Performing an eigendecomposition of the covariance matrix, the contributions to quadratic form term from the infinite-variance directions will be zero. However, the log determinant term will tend to minus infinity.
- The standard solution in this case is to project \mathbf{y} onto the directions orthogonal to the span of H^\top and compute the marginal likelihood in this subspace.
- By taking m to denote the rank of H^\top , we can discard the terms $-\frac{1}{2} \log |B| - \frac{m}{2} \log 2\pi$ from the previous equation to give:

$$\log p(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^\top K_y^{-1}\mathbf{y} + \frac{1}{2}\mathbf{y}^\top C\mathbf{y} - \frac{1}{2} \log |K_y| - \frac{1}{2} \log |A| - \frac{n-m}{2} \log 2\pi,$$

where $A = HK_y^{-1}H^\top$ and $C = K_y^{-1}H^\top A^{-1}HK_y^{-1}$.

AutoML: Gaussian Processes

Covariance Functions for GPs - Advanced

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

MS-Continuity and Differentiability I

We wish to describe a Gaussian process in terms of its smoothness. There are several notions of continuity for random variables. One is the continuity/differentiability in mean square (MS).

Definition

A Gaussian process $f(\mathbf{x})$ is said to be **MS continuous** at \mathbf{x}_* , if

$$\mathbb{E}[|f(\mathbf{x}^{(k)}) - f(\mathbf{x}_*)|^2] \xrightarrow{k \rightarrow \infty} 0 \text{ for all converging sequences } \mathbf{x}^{(k)} \xrightarrow{k \rightarrow \infty} \mathbf{x}_*.$$

A Gaussian process $f(\mathbf{x})$ is said to be **MS differentiable** along the i direction, if the following limit exists, with $\mathbf{e}_i = (0, \dots, 0, 1, 0, \dots, 0)^\top$ being the unit vector along the i -th axis.

$$\lim_{h \rightarrow 0} \mathbb{E}\left[\left|\frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}\right|\right]$$

MS-Continuity and Differentiability II

- The MS continuity/differentiability do not necessarily lead to the continuity/differentiability of sampled functions!
- The MS continuity/differentiability of a Gaussian process can be derived from the smoothness properties of the kernel.
- The GP is continuous in MS iff the covariance function $k(\mathbf{x}, \mathbf{x}')$ is continuous.
- The MS derivative of a Gaussian process exists iff the second derivative $\frac{\partial^2 k(\mathbf{x}, \mathbf{x}')}{\partial \mathbf{x} \partial \mathbf{x}'}$ exists.

Squared Exponential Covariance Function

The squared exponential function is one of the most commonly used covariance functions.

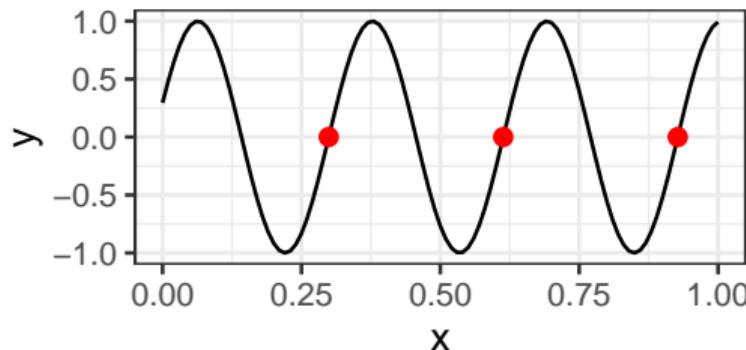
$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right).$$

Properties:

- 💡 It depends merely on the distance $r = \|\mathbf{x} - \mathbf{x}'\| \rightarrow$ isotropic and stationary.
- 💡 Infinitely differentiable \rightarrow the corresponding GP is too smooth.
- 💡 It utilizes strong smoothness assumptions \rightarrow unrealistic for modeling most of the physical processes.

Upcrossing Rate and Characteristic Length-Scale I

- Another way to describe a Gaussian process is the expected number of up-crossings at level-0 on the unit interval, which we denote by N_0 .



- For an isotropic covariance function $k(r)$, the expected number of up-crossings can be calculated explicitly:

$$\mathbb{E}[N_0] = \frac{1}{2\pi} \sqrt{\frac{-k''(0)}{k(0)}}.$$

Upcrossing Rate and Characteristic Length-Scale II

Example (squared exponential):

$$k(r) = \exp\left(-\frac{r^2}{2\ell^2}\right)$$

$$k'(r) = -k(r) \cdot \frac{r}{\ell^2}$$

$$k''(r) = k(r) \cdot \frac{r^2}{\ell^4} - k(r) \cdot \frac{1}{\ell^2}$$

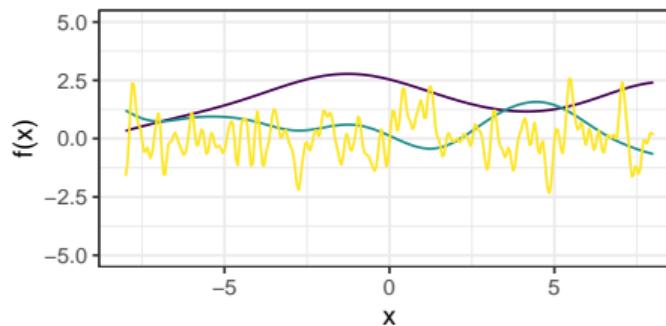
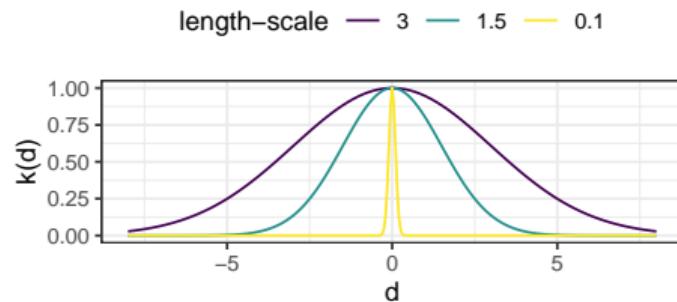
The expected number of upcrossings at level-0 is

$$\mathbb{E}[N_0] = \frac{1}{2\pi} \sqrt{\frac{-k''(0)}{k(0)}} = \frac{1}{2\pi} \sqrt{\frac{1}{\ell^2}} = (2\pi\ell)^{-1}.$$

Upcrossing Rate and Characteristic Length-Scale III

ℓ is called **characteristic length-scale**. Loosely speaking, the characteristic length-scale describes how far you need to move in input space for the function values to become uncorrelated.

- 💡 Left plot: for higher ℓ the correlation between function values (for unchanged distance of input points) is also higher
- 💡 Right plot: a higher ℓ induces a smoother function and thus fewer level-0 upcrossings



Upcrossing Rate and Characteristic Length-Scale IV

For more than $p = 2$ dimensions, the squared exponential can be parameterized as follows:

$$k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \sigma_f^2 \exp\left(-\frac{1}{2} \left(\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right)^\top \mathbf{M} \left(\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\right)\right)$$

Possible choices for the matrix \mathbf{M} include

$$\mathbf{M}_1 = \ell^{-2} \mathbf{I} \quad \mathbf{M}_2 = \text{diag}(\ell)^{-2} \quad \mathbf{M}_3 = \Gamma \Gamma^\top + \text{diag}(\ell)^{-2}$$

where ℓ is a p -vector of positive values and Γ is a $p \times k$ matrix.

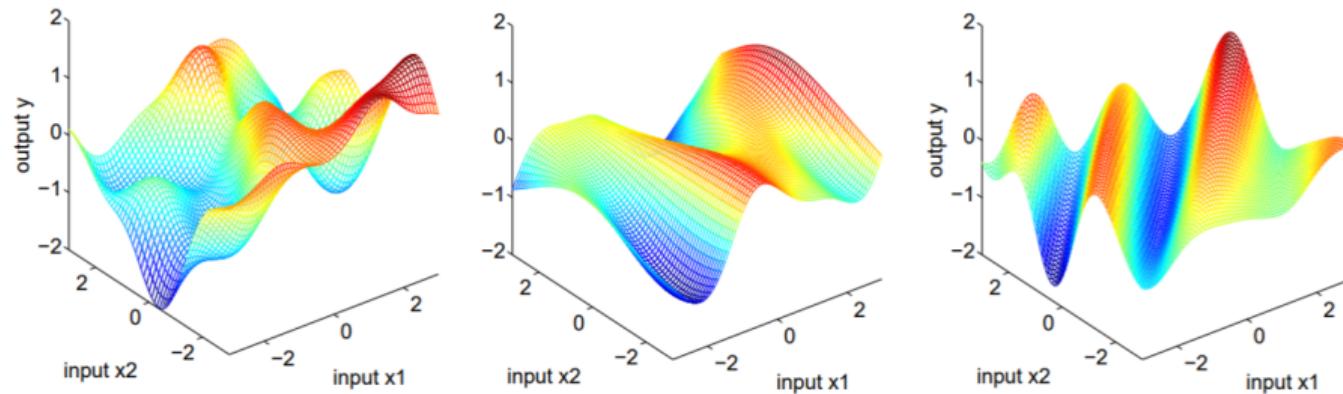
Here again, $\ell = (\ell_1, \dots, \ell_p)$ are characteristic length-scales for each dimension.

Upcrossing Rate and Characteristic Length-Scale V

What is the benefit of learning an individual hyperparameter ℓ_i for each dimension?

- The ℓ_1, \dots, ℓ_p hyperparameters play the role of **characteristic length-scales**.
- Losely speaking, ℓ_i describes how far you need to move along axis i in input space for the function values to be uncorrelated.
- Such a covariance function implements **automatic relevance determination** (ARD), since the inverse of the length-scale ℓ_i determines the relevancy of input feature i to the regression.
- If ℓ_i is very large, the covariance will become almost independent of that input, effectively removing it from inference.
- If the features are on different scales, the data can be automatically **rescaled** by estimating ℓ_1, \dots, ℓ_p

Upcrossing Rate and Characteristic Length-Scale VI



For the first plot, we have chosen $M = I$: the function varies the same in all directions. The second plot is for $M = \text{diag}(\ell)^{-2}$ and $\ell = (1, 3)$: The function varies less rapidly as a function of x_2 than x_1 as the length-scale for x_1 is less. In the third plot $M = \Gamma\Gamma^T + \text{diag}(\ell)^{-2}$ for $\Gamma = (1, -1)^\top$ and $\ell = (6, 6)^\top$. Here Γ gives the direction of the most rapid variation.
[Rasmussen and Williams. 2006]

AutoML: Gaussian Processes

Gaussian Processes: Additional Material

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Notation

In this part,

- (\mathbf{x}_*, y_*) denotes a single test observation, excluded from the training data.
- $\mathbf{X}_* \in \mathbb{R}^{n_* \times p}$ denotes a set of n_* test observations.
- $\mathbf{y}_* \in \mathbb{R}^{n_* \times p}$ denotes the corresponding outcomes, excluded from the training data.

Noisy Gaussian Processes

Noisy Gaussian Processes

- In the previous slides, we implicitly assumed that we access the true function values $f(\mathbf{x})$. However, in many practical cases, we only have a noisy version of the values:

$$y = f(\mathbf{x}) + \epsilon.$$

- By assuming an additive i.i.d. Gaussian noise, the covariance function becomes:

$$\text{cov}(y^{(i)}, y^{(j)}) = k\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right) + \sigma_n^2 \delta_{ij}, \text{ where } \delta_{ij} = 1 \text{ if } i = j.$$

- In the matrix notation, this becomes:

$$\text{cov}(\mathbf{y}) = \mathbf{K} + \sigma_n^2 \mathbf{I} =: \mathbf{K}_y, \text{ where } \sigma_n^2 \text{ is called \textbf{nugget}.}$$

GP vs. Kernelized Ridge Regression

- The predictive function is then

$$f_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y} \sim \mathcal{N}(\bar{f}_*, \text{cov}(\bar{f}_*)),$$

with $\bar{f}_* = \mathbf{K}_*^T \mathbf{K}_y^{-1} \mathbf{y}$ and $\text{cov}(\bar{f}_*) = \mathbf{K}_{**} - \mathbf{K}_*^\top \mathbf{K}_y^{-1} \mathbf{K}_*$.

- The predicted mean value at the training points $\bar{f} = \mathbf{K} \mathbf{K}_y^{-1} \mathbf{y}$ is a **linear combination** of the y values.

 Predicting the posterior mean corresponds exactly to the predictions obtained by kernelized Ridge regression. However, a GP as a Bayesian model provides us with much more information (i.e., a posterior distribution), whilst the kernelized Ridge regression does not.

Bayesian Linear Regression as a GP

Bayesian Linear Regression as a GP

- One example for a Gaussian process is the Bayesian linear regression model, and we already discuss it.
- For $\boldsymbol{\theta} \sim \mathcal{N}(\mathbf{0}, \tau^2 \mathbf{I})$, the joint distribution of any set of function values is Gaussian:

$$f(\mathbf{x}^{(i)}) = \boldsymbol{\theta}^\top \mathbf{x}^{(i)} + \epsilon.$$

- The corresponding mean function is $m(\mathbf{x}) = \mathbf{0}$, and the covariance function is

$$\begin{aligned}\text{cov}(f(\mathbf{x}), f(\mathbf{x}')) &= \mathbb{E}[f(\mathbf{x})f(\mathbf{x}')] - \underbrace{\mathbb{E}[f(\mathbf{x})]\mathbb{E}[f(\mathbf{x}')]}_{=0} \\ &= \mathbb{E}[(\boldsymbol{\theta}^\top \mathbf{x} + \epsilon)^\top (\boldsymbol{\theta}^\top \mathbf{x}' + \epsilon)] \\ &= \tau^2 \mathbf{x}^\top \mathbf{x}' + \sigma^2 =: k(\mathbf{x}, \mathbf{x}').\end{aligned}$$

Feature Spaces and the Kernel Trick I

- If one relaxes the linearity assumption by projecting the features into a higher dimensional feature space \mathcal{Z} using a basis function $\phi : \mathcal{X} \rightarrow \mathcal{Z}$, the corresponding covariance function becomes:

$$k(\mathbf{x}, \mathbf{x}') = \tau^2 \phi(\mathbf{x})^\top \phi(\mathbf{x}') + \sigma^2.$$

- To get arbitrarily complicated functions, we would have to handle high-dimensional feature vectors $\phi(\mathbf{x})$.
- Fortunately, all we need to know is the inner product $\phi(\mathbf{x})^\top \phi(\mathbf{x}')$. That is, the feature vector itself never occurs in calculations.

Feature Spaces and the Kernel Trick II

- 💡 If we can get the inner product directly and without calculating the infinite feature vectors, we can infer an infinitely complicated model with a finite amount of computation. This idea is known as **kernel trick**.
- A Gaussian process can then be defined by either:
 - ▶ deriving the covariance function from the inner products of the basis functions evaluations, or
 - ▶ choosing a positive definite kernel function (Mercer Kernel), which- according to Mercer's theorem - corresponds to taking the inner products in some (possibly infinite) feature space.

Summary: Gaussian Process Regression

- The Gaussian process regression is equivalent to the **kernelized** Bayesian linear regression.
- The covariance function describes the shape of the Gaussian process. Hence, with the right choice of covariance function, remarkably flexible models can be built.
- Naive implementations of Gaussian process models scale poorly with large datasets, as
 - ▶ the kernel matrix has to be inverted / factorized, which is $\mathcal{O}(n^3)$,
 - ▶ computing the kernel matrix uses $\mathcal{O}(n^2)$ memory - running out of memory places a hard limit on the size of problems
 - ▶ generating predictions is $\mathcal{O}(n)$ for the mean, but $\mathcal{O}(n^2)$ for the variance.
(...special tricks are needed.)

AutoML: Gaussian Processes

Gaussian Process Classification

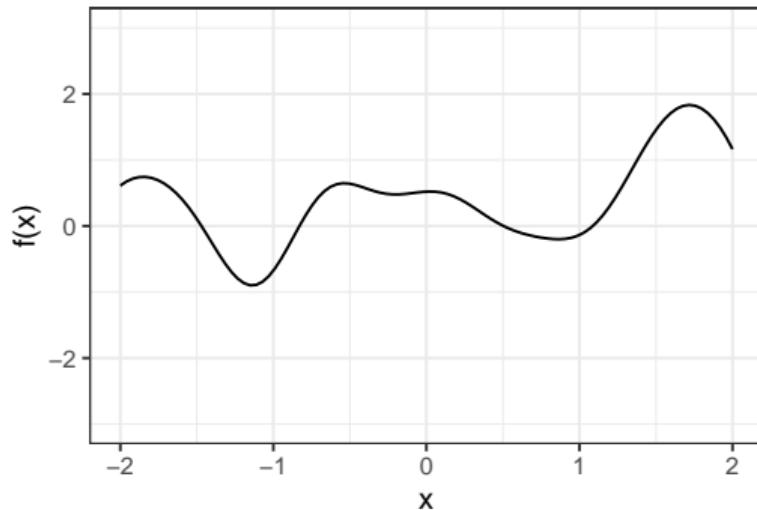
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Training a GP via the Maximum Likelihood I

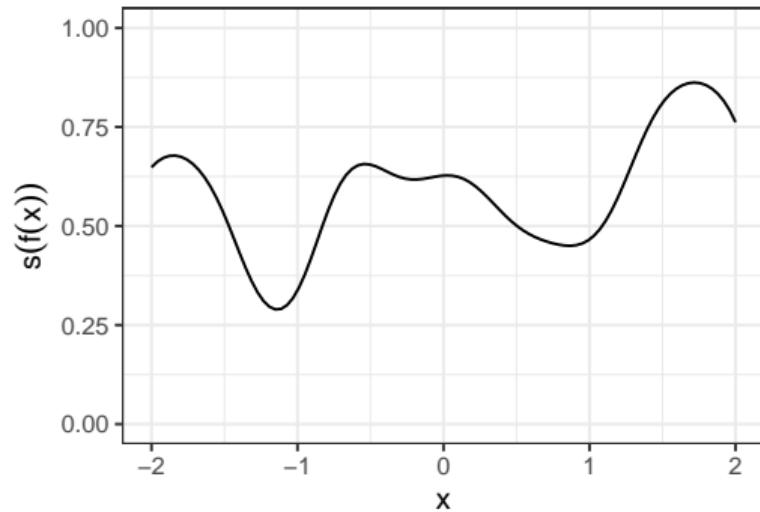
- Consider a binary classification problem, in which we want to learn $h : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{Y} = \{0, 1\}$.
- The idea behind Gaussian process classification is straightforward: a GP prior is placed over the score function $f(\mathbf{x})$ and then transformed to a class probability via a sigmoid function $s(t)$:
$$p(y = 1 \mid f(\mathbf{x})) = s(f(\mathbf{x})).$$
- Since this is a non-Gaussian likelihood, we need to use approximate inference methods, such as Laplace approximation, expectation propagation, MCMC.
- For more details see [Rasmussen and Williams. 2006: Chapter 3]

Training a GP via the Maximum Likelihood II

Function drawn from a GP prior



Function transformed into probs



Training a GP via the Maximum Likelihood III

- According to Bayes' rule, the posterior of the score function \mathbf{f} takes the following form:

$$p(\mathbf{f} \mid \mathbf{X}, \mathbf{y}) = \frac{p(\mathbf{y} \mid \mathbf{f}, \mathbf{X}) \cdot p(\mathbf{f} \mid \mathbf{X})}{p(\mathbf{y} \mid \mathbf{X})} \propto p(\mathbf{y} \mid \mathbf{f}) \cdot p(\mathbf{f} \mid \mathbf{X}),$$

where, the denominator is independent of \mathbf{f} and hence has been dropped.

- From the GP assumption, we can assert that $p(\mathbf{f} \mid \mathbf{X}) \sim \mathcal{N}(0, \mathbf{K})$. Hence, we have:

$$\log p(\mathbf{f} \mid \mathbf{X}, \mathbf{y}) \propto \log p(\mathbf{y} \mid \mathbf{f}) - \frac{1}{2} \mathbf{f}^\top \mathbf{K}^{-1} \mathbf{f} - \frac{1}{2} \log |\mathbf{K}| - \frac{n}{2} \log 2\pi.$$

Training a GP via the Maximum Likelihood IV

- If the kernel is fixed, the last two terms will be fixed. To obtain the maximum a-posteriori estimate (MAP), we should minimize:

$$\frac{1}{2} \mathbf{f}^\top \mathbf{K}^{-1} \mathbf{f} - \sum_{i=1}^n \log p(y^{(i)} | f^{(i)}) + C.$$

- Note that $-\sum_{i=1}^n \log p(y^{(i)} | f^{(i)})$ is the logistic loss.
- It can be seen that the Gaussian process classification corresponds to the **kernel Bayesian logistic regression!**

Comparison: GP vs. SVM I

- For the SVM, we have:

$$\frac{1}{2} \|\boldsymbol{\theta}\|^2 + C \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)})),$$

where $L(y, f(\mathbf{x})) = \max\{0, 1 - f(\mathbf{x}) \cdot y\}$ is the Hinge loss.

- Plugging that in, the optimization objective would be:

$$\frac{1}{2} \mathbf{f}^\top \mathbf{K}^{-1} \mathbf{f} + C \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)})).$$

- From the representer theorem: $\boldsymbol{\theta} = \sum_{i=1}^n \beta_i y^{(i)} k(\mathbf{x}^{(i)}, \cdot)$, and thus:

$$\boldsymbol{\theta}^\top \boldsymbol{\theta} = \beta^\top \mathbf{K} \beta = \mathbf{f}^\top \mathbf{K}^{-1} \mathbf{f}$$

Comparison: GP vs. SVM II

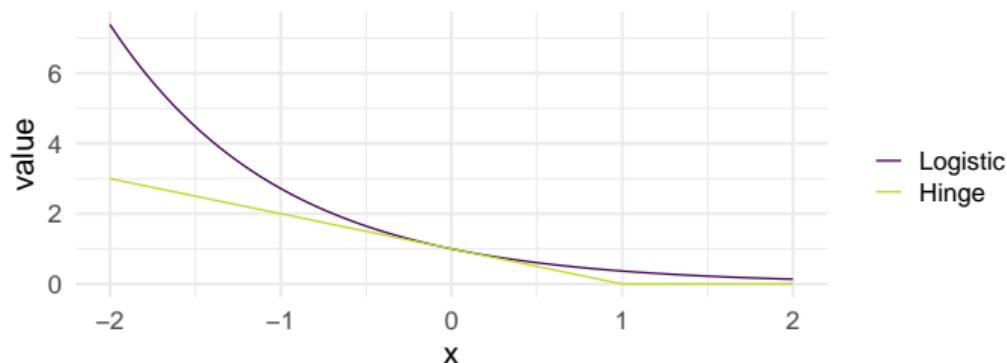
- For log-concave likelihoods $\log p(\mathbf{y} \mid \mathbf{f})$, there is a close correspondence between the MAP solution of the GP classifier and the SVM solution:

$$\arg \min_{\mathbf{f}} \frac{1}{2} \mathbf{f}^\top \mathbf{K}^{-1} \mathbf{f} - \sum_{i=1}^n \log p(y^{(i)} \mid f^{(i)}) + C \quad (\text{GP classifier})$$

$$\arg \min_{\mathbf{f}} \frac{1}{2} \mathbf{f}^\top \mathbf{K}^{-1} \mathbf{f} + C \sum_{i=1}^n L(y^{(i)}, f(\mathbf{x}^{(i)})) \quad (\text{SVM classifier})$$

Comparison: GP vs. SVM III

- Both the Hinge loss and the Bernoulli loss are monotonically decreasing with increasing margin $yf(\mathbf{x})$.
- The key difference is that the Hinge loss takes on the value 0 for $yf(\mathbf{x}) \geq 1$, while the Bernoulli loss decays slowly.
- It is this flat part of the Hinge function that gives rise to the sparsity of the SVM solution.
- The SVM classifier can be construed as a “sparse” GP classifier.



AutoML: Bayesian Optimization for HPO

Overview of Bayesian Optimization

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Global Blackbox Optimization

- Consider the global optimization problem of finding:

$$\boldsymbol{\lambda}^* \in \arg \min_{\boldsymbol{\lambda} \in \Lambda} f(\boldsymbol{\lambda})$$

Global Blackbox Optimization

- Consider the **global optimization** problem of finding:

$$\boldsymbol{\lambda}^* \in \arg \min_{\boldsymbol{\lambda} \in \Lambda} f(\boldsymbol{\lambda})$$

- In the most general form, function f is a **blackbox function**:

$$\boldsymbol{\lambda} \rightarrow \blacksquare \rightarrow f(\boldsymbol{\lambda})$$

- Only mode of interaction with f : querying f 's value at a given $\boldsymbol{\lambda}$
- Function f may not be available in closed form, not convex, not differentiable, noisy, etc.

Global Blackbox Optimization

- Consider the **global optimization** problem of finding:

$$\boldsymbol{\lambda}^* \in \arg \min_{\boldsymbol{\lambda} \in \Lambda} f(\boldsymbol{\lambda})$$

- In the most general form, function f is a **blackbox function**:

$$\boldsymbol{\lambda} \rightarrow \text{[black square]} \rightarrow f(\boldsymbol{\lambda})$$

- Only mode of interaction with f : querying f 's value at a given $\boldsymbol{\lambda}$
- Function f may not be available in closed form, not convex, not differentiable, noisy, etc.
- Today, we'll discuss a **Bayesian** approach for solving such blackbox optimization problems

Global Blackbox Optimization

- Consider the **global optimization** problem of finding:

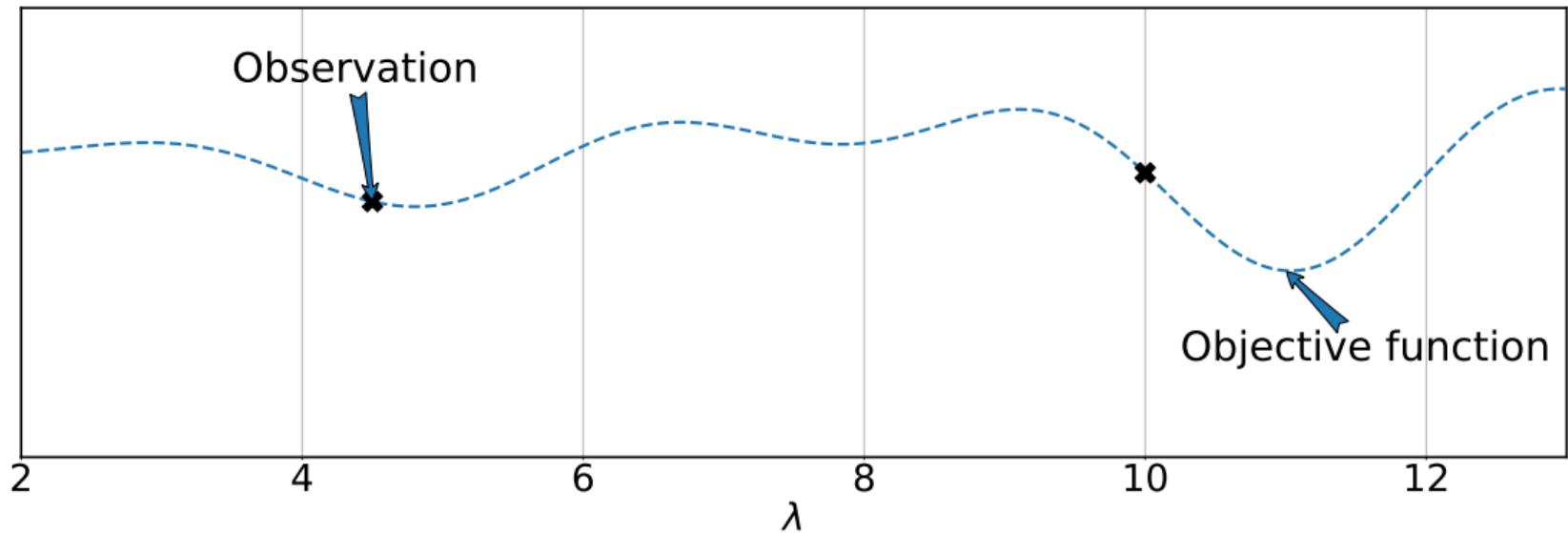
$$\boldsymbol{\lambda}^* \in \arg \min_{\boldsymbol{\lambda} \in \Lambda} f(\boldsymbol{\lambda})$$

- In the most general form, function f is a **blackbox function**:

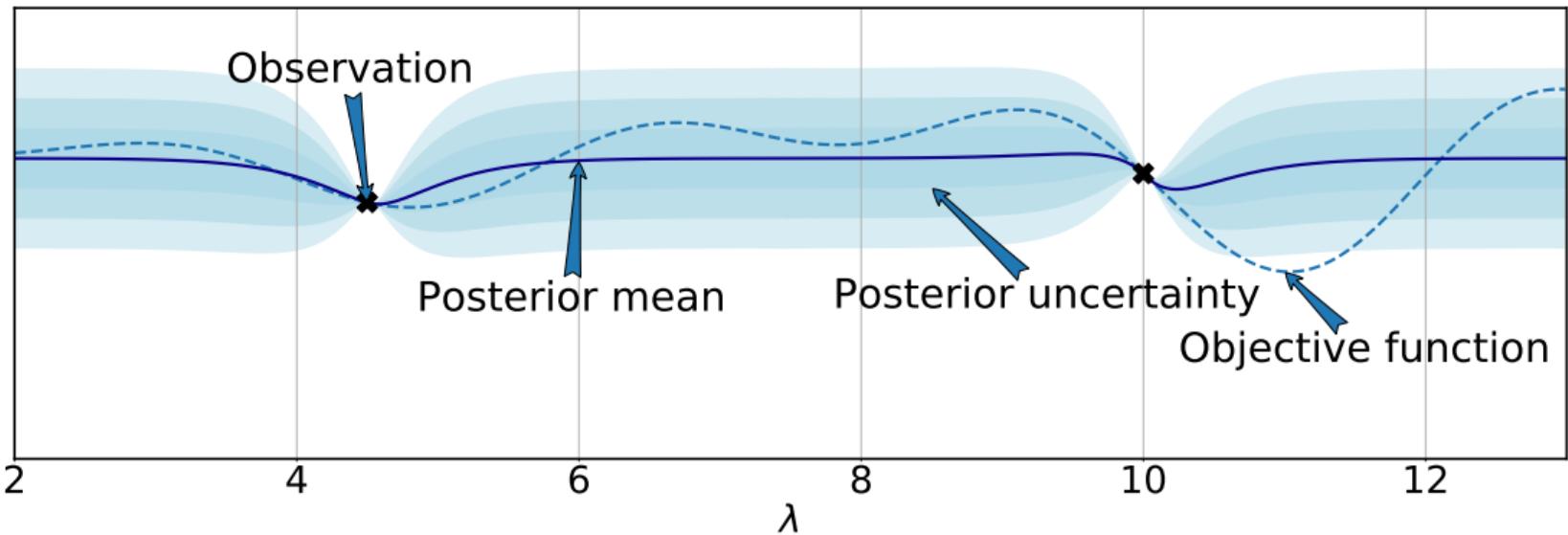


- Only mode of interaction with f : querying f 's value at a given $\boldsymbol{\lambda}$
- Function f may not be available in closed form, not convex, not differentiable, noisy, etc.
- Today, we'll discuss a **Bayesian** approach for solving such blackbox optimization problems
- Blackbox optimization can be used for hyperparameter optimization (HPO)
 - Define $f(\boldsymbol{\lambda}) := \mathcal{L}(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$
 - Note: for formulations of HPO that go beyond blackbox optimization, see next lecture

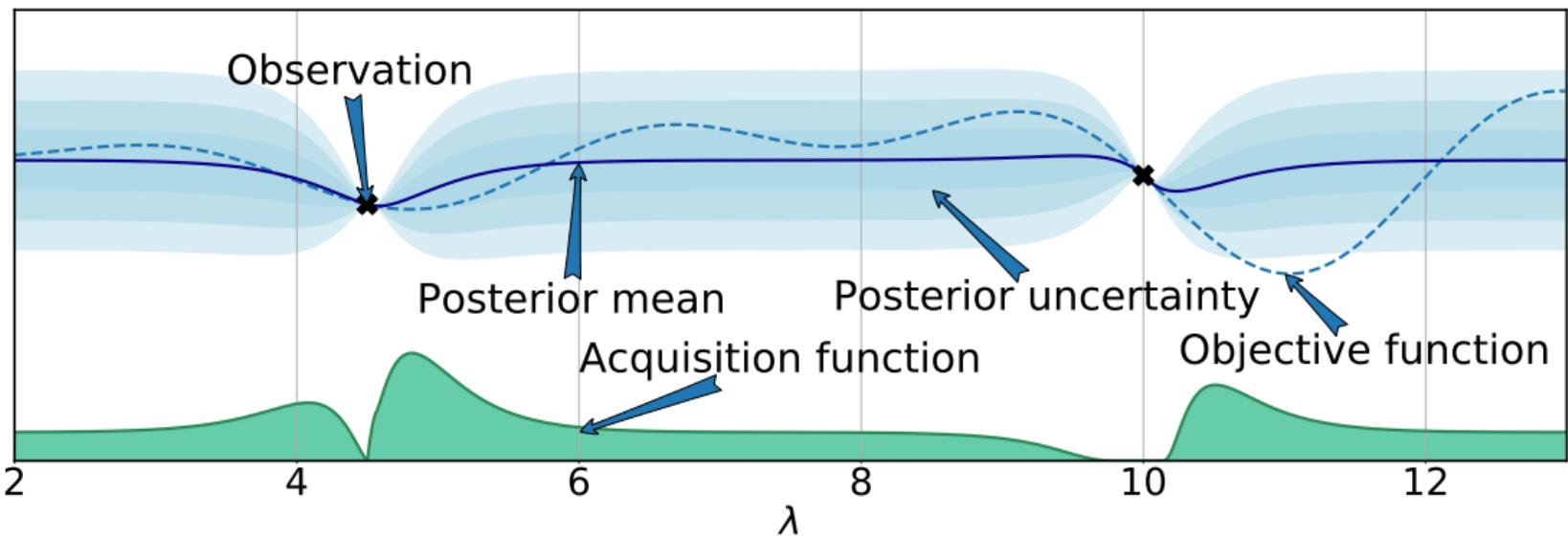
Bayesian Optimization of a blackbox function in a nutshell



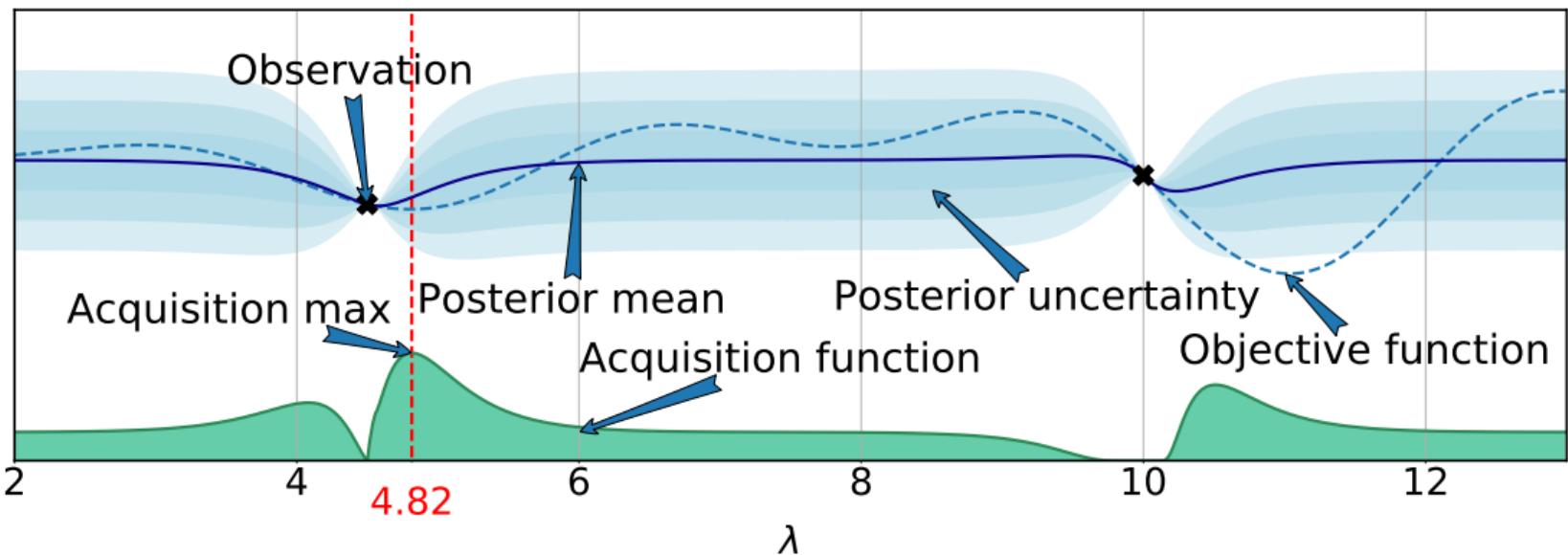
Bayesian Optimization of a blackbox function in a nutshell



Bayesian Optimization of a blackbox function in a nutshell



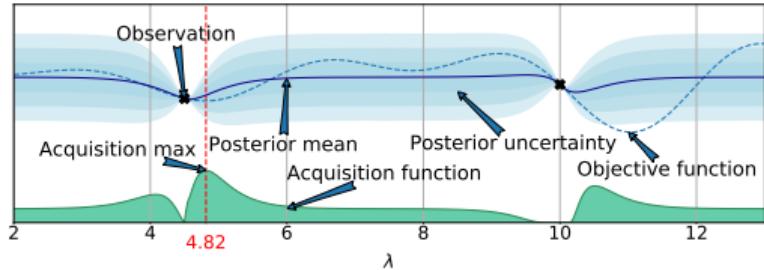
Bayesian Optimization of a blackbox function in a nutshell



Bayesian Optimization of a blackbox function in a nutshell

General approach

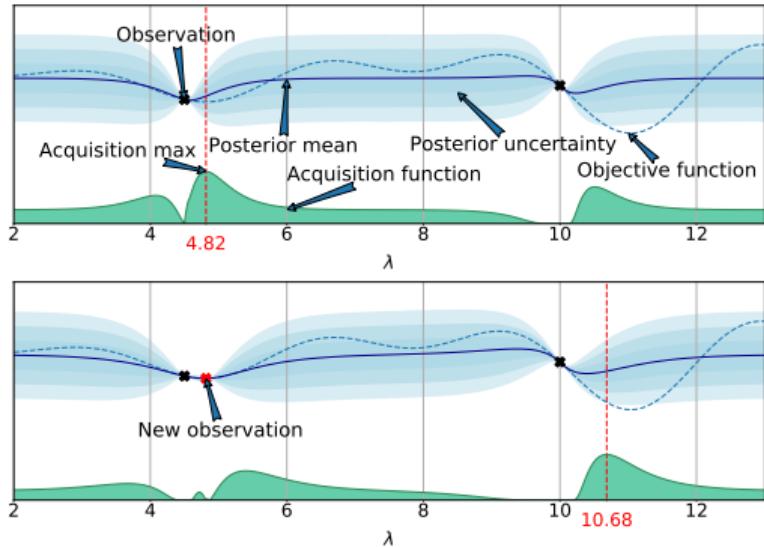
- Fit a **probabilistic model** to the collected function samples $\langle \lambda, c(\lambda) \rangle$
- Use the model to guide optimization, trading off **exploration vs exploitation**



Bayesian Optimization of a blackbox function in a nutshell

General approach

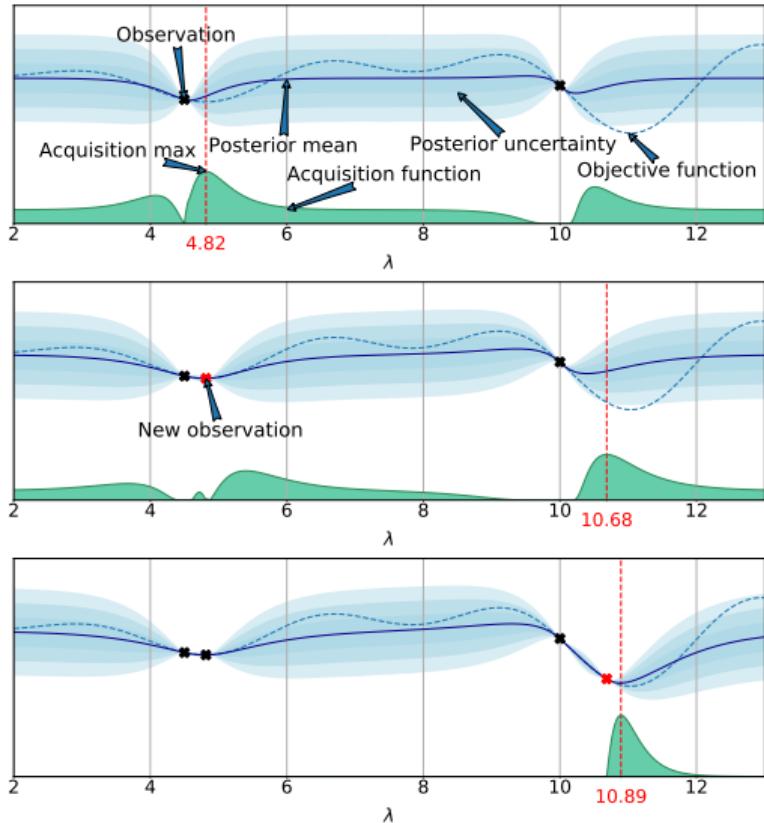
- Fit a **probabilistic model** to the collected function samples $\langle \lambda, c(\lambda) \rangle$
- Use the model to guide optimization, trading off **exploration vs exploitation**



Bayesian Optimization of a blackbox function in a nutshell

General approach

- Fit a **probabilistic model** to the collected function samples $\langle \lambda, c(\lambda) \rangle$
- Use the model to guide optimization, trading off **exploration vs exploitation**



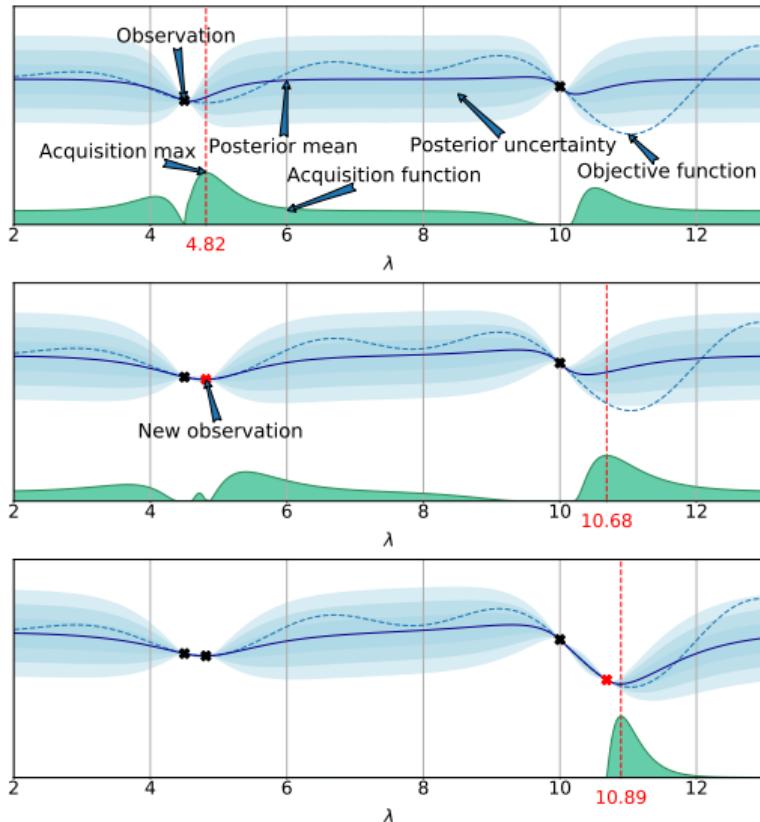
Bayesian Optimization of a blackbox function in a nutshell

General approach

- Fit a **probabilistic model** to the collected function samples $\langle \lambda, c(\lambda) \rangle$
- Use the model to guide optimization, trading off **exploration vs exploitation**

Popular approach in the statistics literature since Mockus et al. [1978]

- Efficient in #function evaluations
- Works when objective is **nonconvex, noisy, has unknown derivatives**, etc.
- Recent **convergence** results
[Srinivas et al. 2009; Bull et al. 2011; de Freitas et al. 2012; Kawaguchi et al. 2015]



Bayesian Optimization: Pseudocode

BO loop

Require: Search space Λ , cost function c , acquisition function u , predictive model \hat{c} , maximal number of function evaluations T

Result : Best configuration $\hat{\lambda}$ (according to \mathcal{D} or \hat{c})

- 1 Initialize data $\mathcal{D}^{(0)}$ with initial observations
 - 2 **for** $t = 1$ **to** T **do**
 - 3 Fit predictive model $\hat{c}^{(t)}$ on $\mathcal{D}^{(t-1)}$
 - 4 Select next query point: $\lambda^{(t)} \in \arg \max_{\lambda \in \Lambda} u(\lambda; \mathcal{D}^{(t-1)}, \hat{c}^{(t)})$
 - 5 Query $c(\lambda^{(t)})$
 - 6 Update data: $\mathcal{D}^{(t)} \leftarrow \mathcal{D}^{(t-1)} \cup \{\langle \lambda^{(t)}, c(\lambda^{(t)}) \rangle\}$
-

Bayesian Optimization: Origin of the Name

- Bayesian optimization uses Bayes' theorem:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \propto P(B|A) \times P(A)$$

- Bayesian optimization uses this to compute a posterior over functions:

$$P(f|\mathcal{D}_{1:t}) \propto P(\mathcal{D}_{1:t}|f) \times P(f), \quad \text{where } \mathcal{D}_{1:t} = \{\boldsymbol{\lambda}_{1:t}, c(\boldsymbol{\lambda}_{1:t})\}$$

Bayesian Optimization: Origin of the Name

- Bayesian optimization uses Bayes' theorem:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)} \propto P(B|A) \times P(A)$$

- Bayesian optimization uses this to compute a posterior over functions:

$$P(f|\mathcal{D}_{1:t}) \propto P(\mathcal{D}_{1:t}|f) \times P(f), \quad \text{where } \mathcal{D}_{1:t} = \{\lambda_{1:t}, c(\lambda_{1:t})\}$$

- Meaning of the individual terms:
 - ▶ $P(f)$ is the prior over functions, which represents our belief about the space of possible objective functions before we see any data
 - ▶ $\mathcal{D}_{1:t}$ is the data (or observations, evidence)
 - ▶ $P(\mathcal{D}_{1:t}|f)$ is the likelihood of the data given a function
 - ▶ $P(f|\mathcal{D}_{1:t})$ is the posterior probability over functions given the data

Bayesian Optimization: Advantages and Disadvantages

Advantages

- Sample efficient
- Can handle noise
- Native incorporation of priors
- Does not require gradients
- Theoretical guarantees

Bayesian Optimization: Advantages and Disadvantages

Advantages

- Sample efficient
- Can handle noise
- Native incorporation of priors
- Does not require gradients
- Theoretical guarantees

Disadvantages

- Overhead because of model training in each iteration
- Crucially relies on robust surrogate model

Questions to Answer for Yourself / Discuss with Friends

- Repetition. What is Bayesian about Bayesian optimization?
- Repetition. Write down the steps of the BO loop.
- Discussion. Can you think of an expensive blackbox optimization problem other than hyperparameter optimization?

Learning Goals of this Lecture

After this lecture, students can ...

- Explain the basics of Bayesian optimization
- Derive simple acquisition functions
- Describe complex lookahead acquisition functions
- Describe possible surrogate models and their pros and cons
- Discuss the limits of Bayesian optimization and extensions to tackle these
- Discuss success stories of Bayesian optimization

AutoML: Bayesian Optimization for HPO

Computationally Cheap Acquisition Functions

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

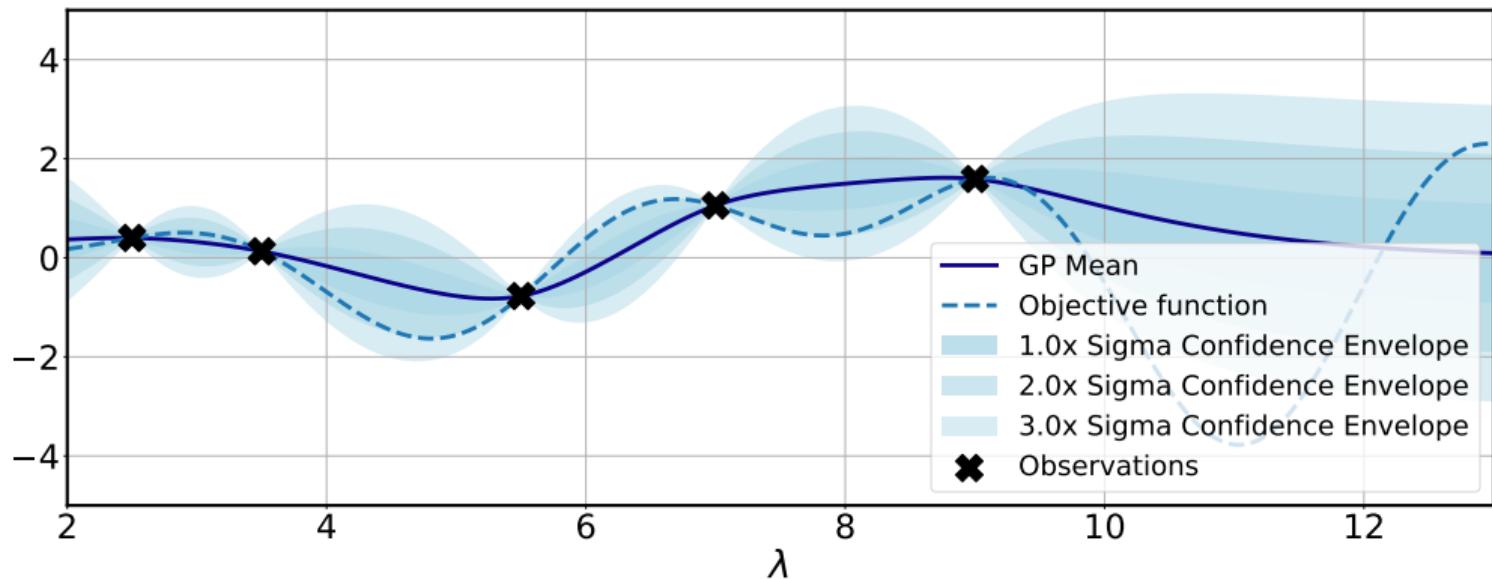
Acquisition Functions: the Basics

- Given the surrogate model $\hat{c}^{(t)}$ at the t -th iteration of BO, the acquisition function $u(\cdot)$ judges the utility (or usefulness) of evaluating f at $\lambda^{(t)} \in \Lambda$ next

Acquisition Functions: the Basics

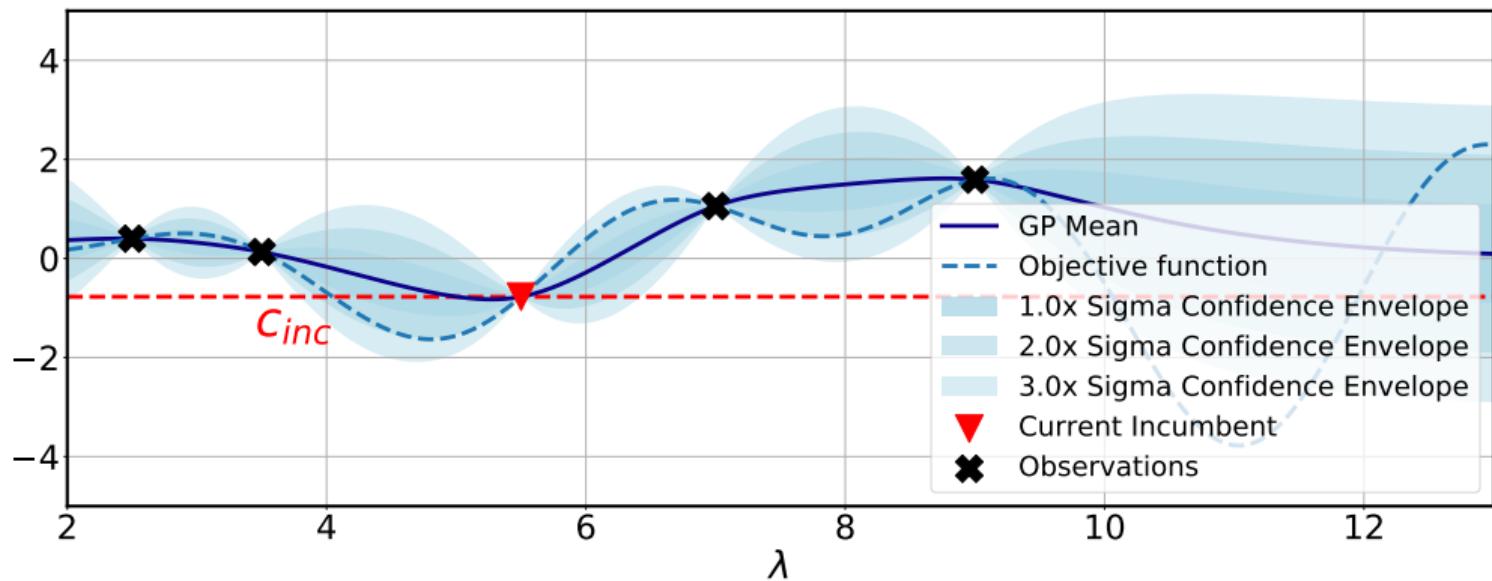
- Given the surrogate model $\hat{c}^{(t)}$ at the t -th iteration of BO, the acquisition function $u(\cdot)$ judges the utility (or usefulness) of evaluating f at $\lambda^{(t)} \in \Lambda$ next
- The acquisition function needs to trade off exploration and exploitation
 - E.g., just picking the λ with lowest predicted mean would be too greedy
 - We also need to take into account the uncertainty of the surrogate model $\hat{c}^{(t)}$ to explore

Probability of Improvement (PI): Concept



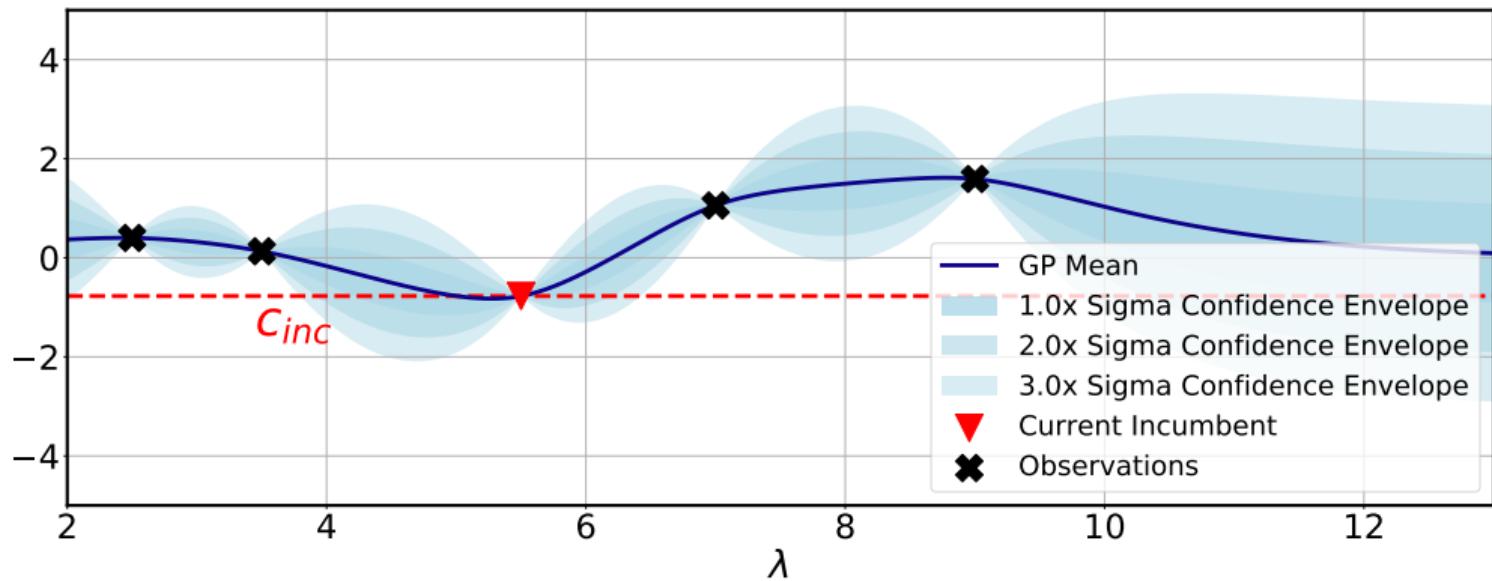
Given the surrogate fit at iteration t

Probability of Improvement (PI): Concept



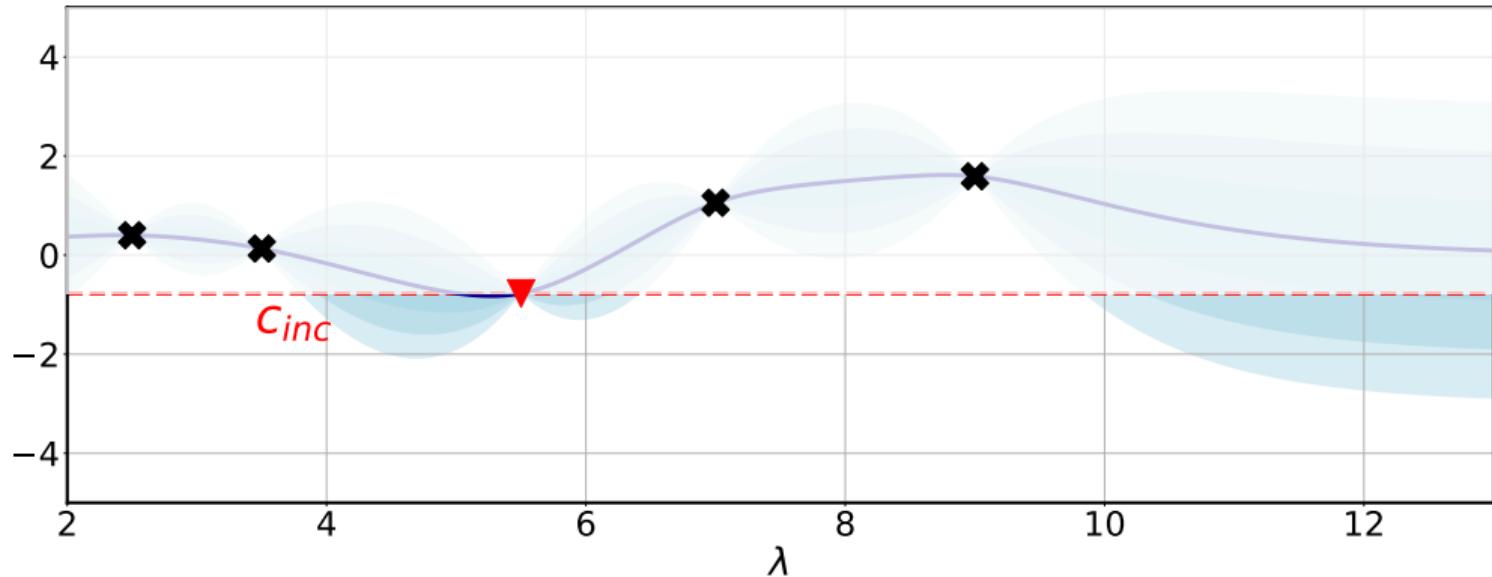
Current incumbent $\hat{\lambda}$ and its observed cost c_{inc}

Probability of Improvement (PI): Concept



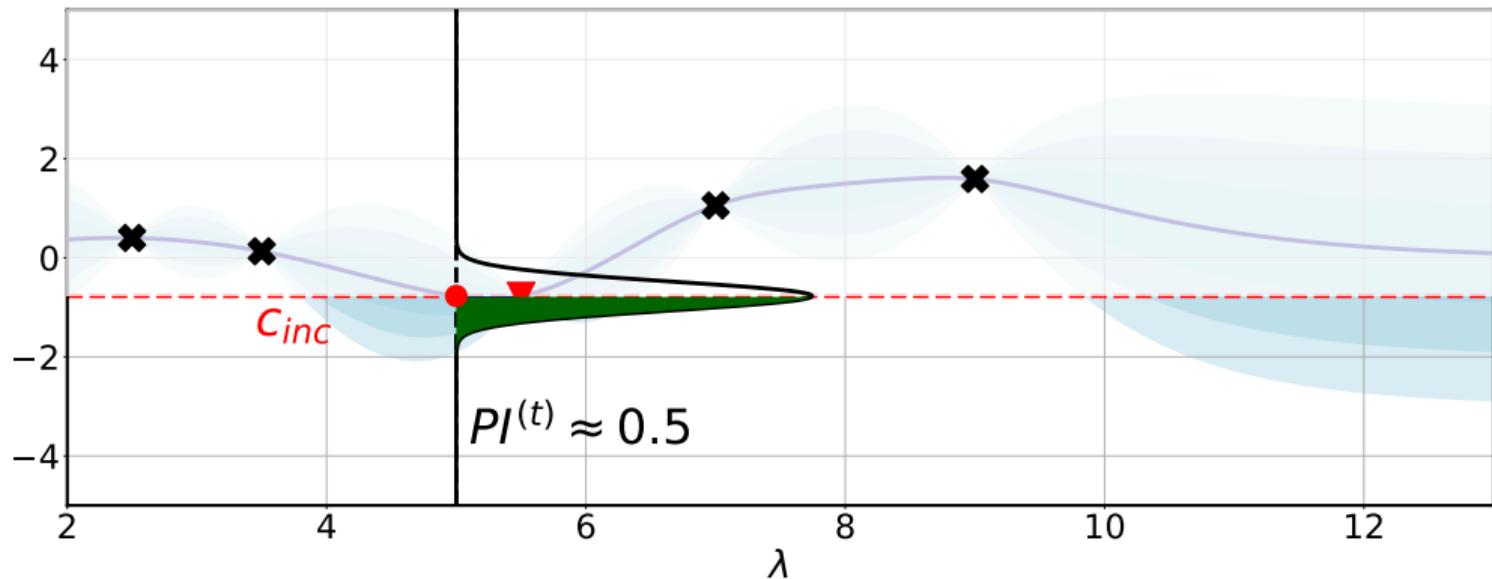
Now let's drop the objective function - it's unknown after all!

Probability of Improvement (PI): Concept



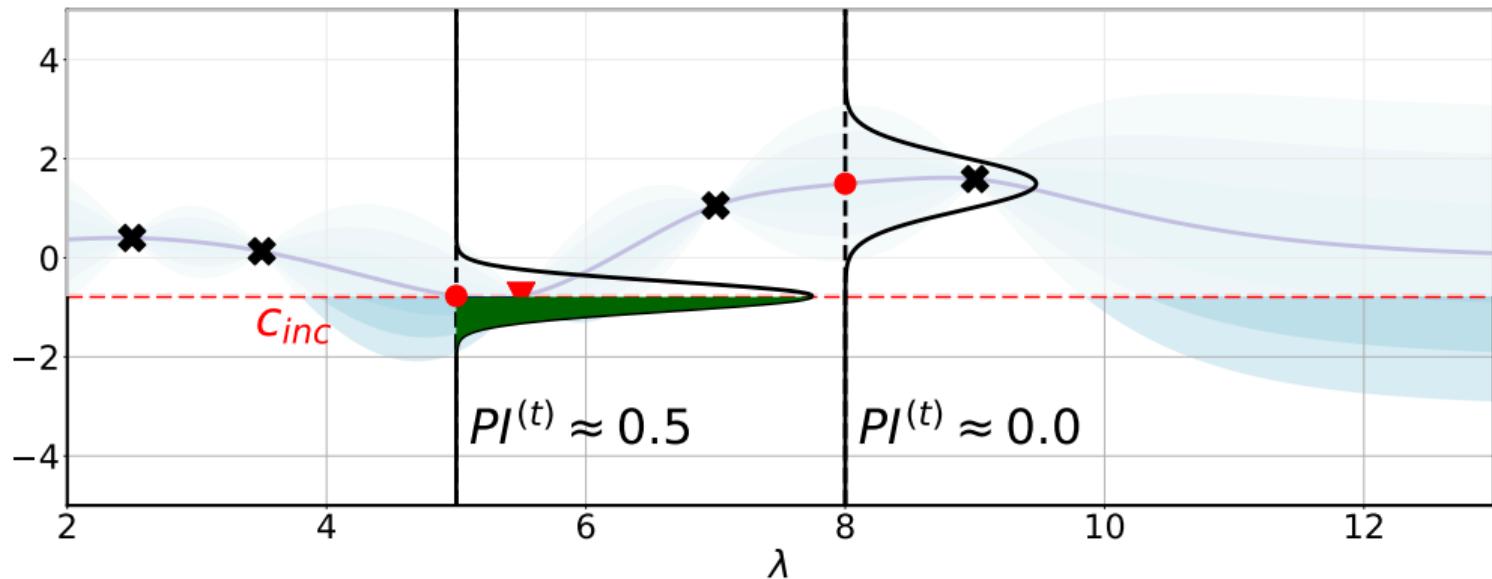
Intuitively, we care about the probability of improving over the current incumbent

Probability of Improvement (PI): Concept



PDF of a good candidate configuration. Only the green area is an improvement.

Probability of Improvement (PI): Concept



PDF of a bad candidate configuration

Probability of Improvement (PI): Formal Definition

- We define the **current incumbent at time step t** as: $\hat{\lambda}^{(t-1)} \in \arg \min_{\lambda' \in \mathcal{D}^{(t-1)}} c(\lambda')$
- We write c_{inc} shorthand for the **cost of the current incumbent**: $c_{inc} = c(\hat{\lambda}^{(t-1)})$
- The **probability of improvement** $u_{PI}(\lambda)$ at a configuration λ is then defined as:

$$u_{PI}^{(t)}(\lambda) = P(c(\lambda) \leq c_{inc}).$$

Probability of Improvement (PI): Formal Definition

- We define the **current incumbent at time step t** as: $\hat{\lambda}^{(t-1)} \in \arg \min_{\lambda' \in \mathcal{D}^{(t-1)}} c(\lambda')$
- We write c_{inc} shorthand for the **cost of the current incumbent**: $c_{inc} = c(\hat{\lambda}^{(t-1)})$
- The **probability of improvement** $u_{PI}(\lambda)$ at a configuration λ is then defined as:

$$u_{PI}^{(t)}(\lambda) = P(c(\lambda) \leq c_{inc}).$$

- Since the predictive distribution for $c(\lambda)$ is a Gaussian $\mathcal{N}(\mu^{(t-1)}(\lambda), \sigma^{2(t-1)}(\lambda))$, this can be written as:

$$u_{PI}^{(t)}(\lambda) = \Phi[Z], \quad \text{with } Z = \frac{c_{inc} - \mu^{(t-1)}(\lambda) - \xi}{\sigma^{(t-1)}(\lambda)},$$

where $\Phi(\cdot)$ is the CDF of the standard normal distribution and ξ is an optional exploration parameter

Probability of Improvement (PI): Formal Definition

- We define the **current incumbent at time step t** as: $\hat{\lambda}^{(t-1)} \in \arg \min_{\lambda' \in \mathcal{D}^{(t-1)}} c(\lambda')$
- We write c_{inc} shorthand for the **cost of the current incumbent**: $c_{inc} = c(\hat{\lambda}^{(t-1)})$
- The **probability of improvement** $u_{PI}(\lambda)$ at a configuration λ is then defined as:

$$u_{PI}^{(t)}(\lambda) = P(c(\lambda) \leq c_{inc}).$$

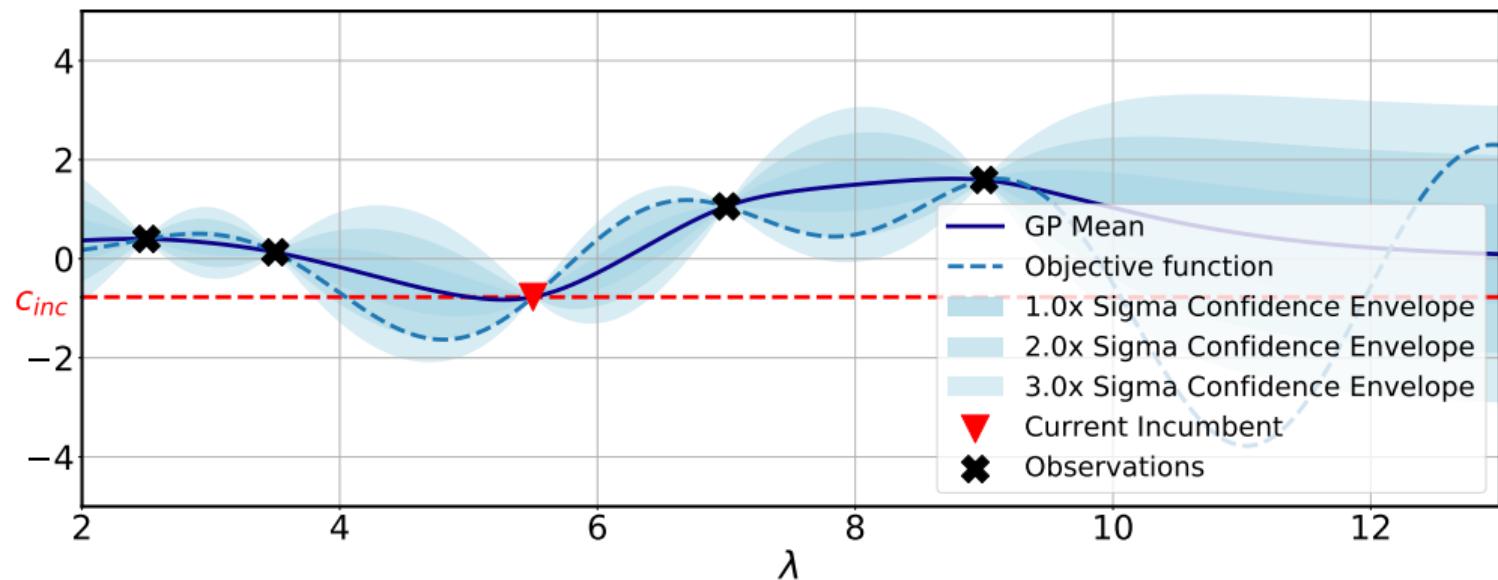
- Since the predictive distribution for $c(\lambda)$ is a Gaussian $\mathcal{N}(\mu^{(t-1)}(\lambda), \sigma^{2(t-1)}(\lambda))$, this can be written as:

$$u_{PI}^{(t)}(\lambda) = \Phi[Z], \quad \text{with } Z = \frac{c_{inc} - \mu^{(t-1)}(\lambda) - \xi}{\sigma^{(t-1)}(\lambda)},$$

where $\Phi(\cdot)$ is the CDF of the standard normal distribution and ξ is an optional exploration parameter

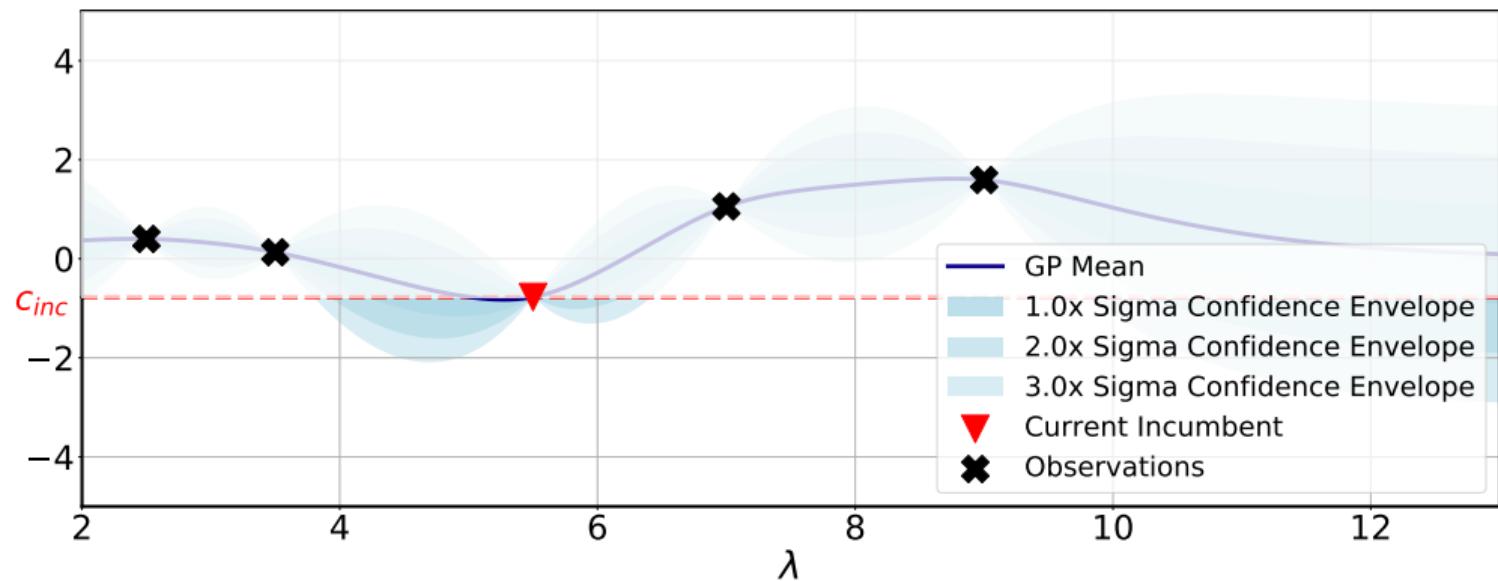
Choose $\lambda^{(t)} \in \arg \max_{\lambda \in \Lambda} (u_{PI}^{(t)}(\lambda))$

Expected Improvement (EI): Concept



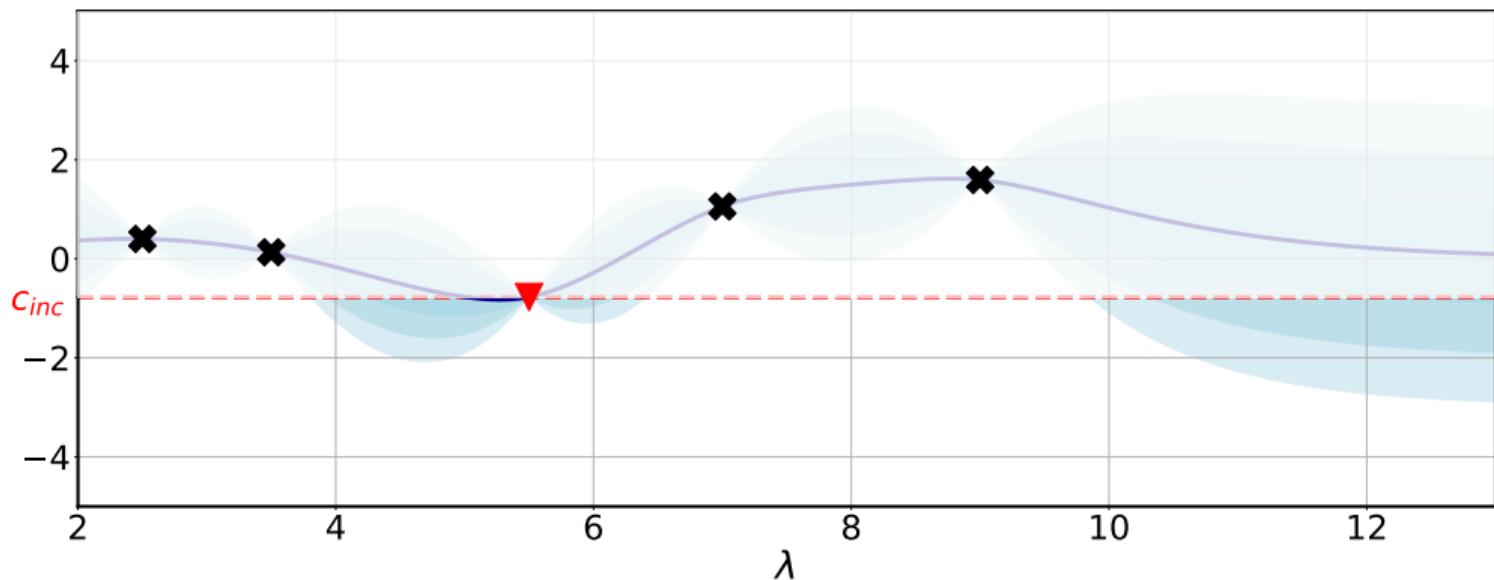
Given the surrogate fit at iteration t

Expected Improvement (EI): Concept



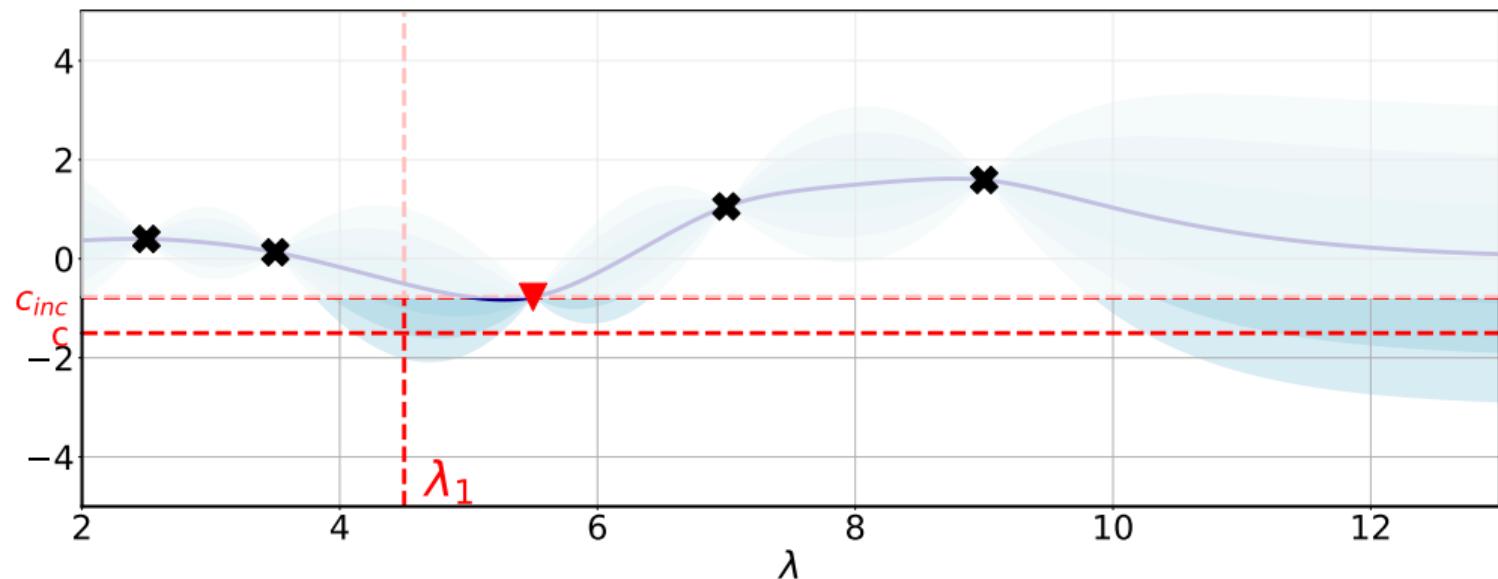
Region of probable improvement – but **how large** is the improvement?

Expected Improvement (EI): Concept



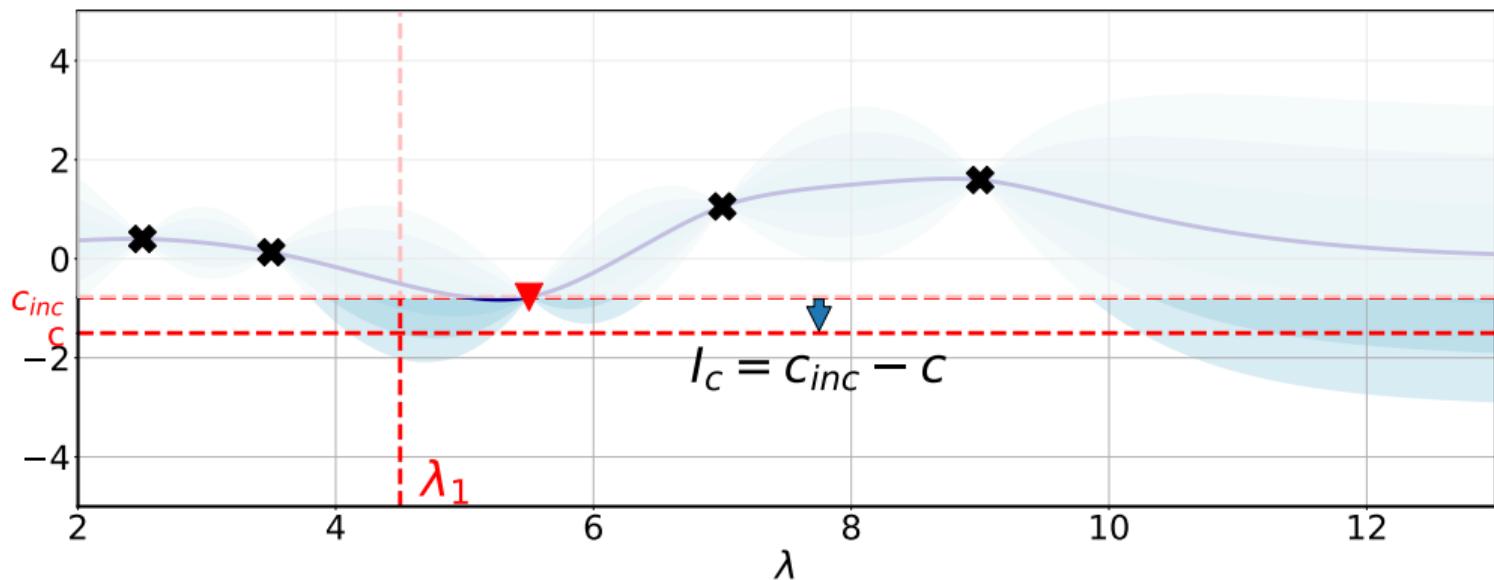
Region of probable improvement – but **how large** is the improvement?

Expected Improvement (EI): Concept



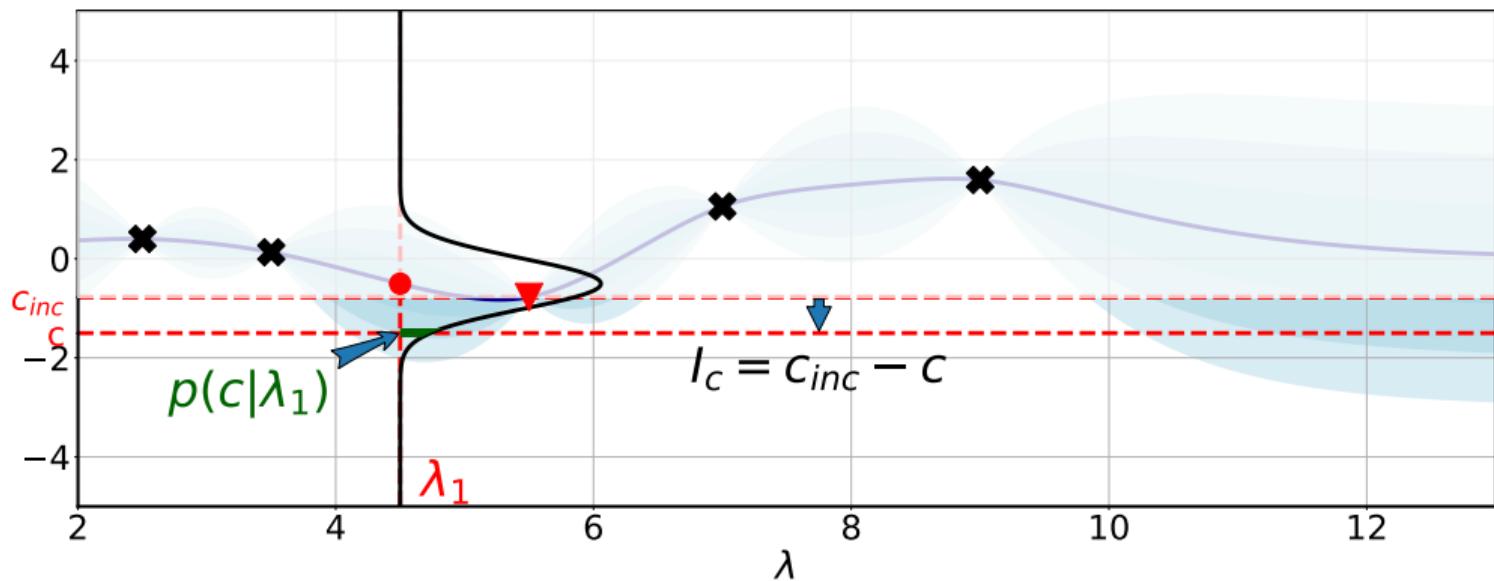
Hypothetical *real* cost c at a given λ - unknown in practice without evaluating

Expected Improvement (EI): Concept



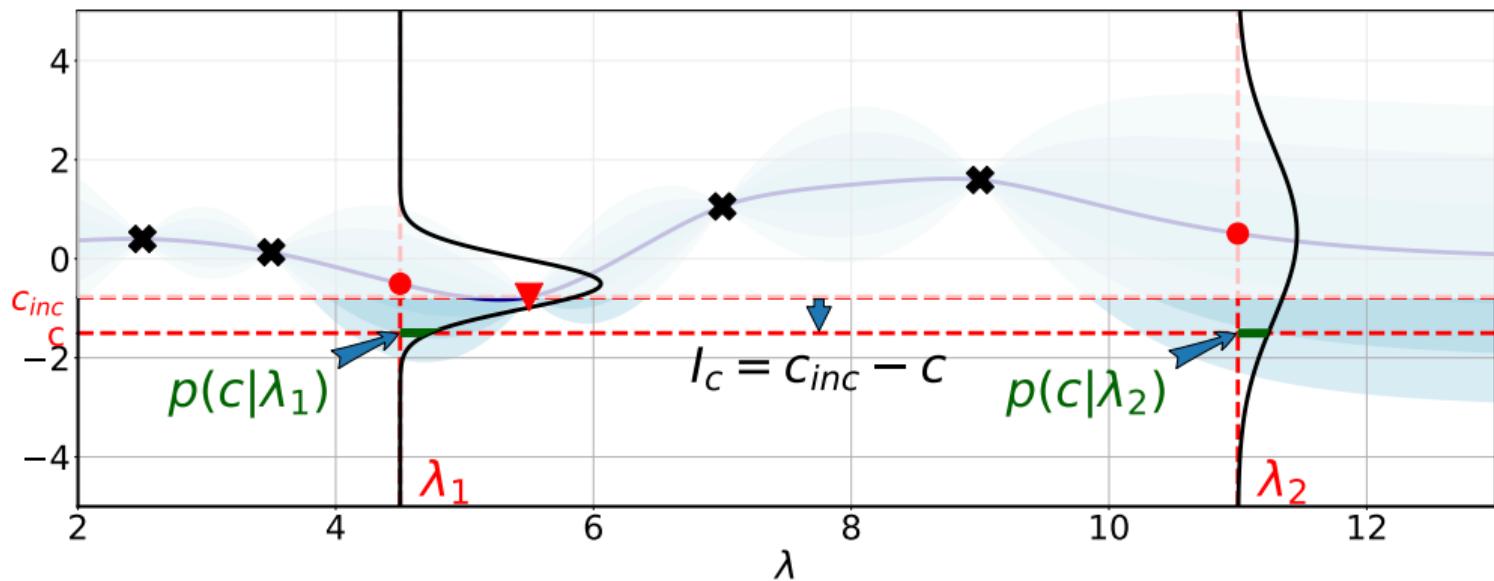
Given a hypothetical c , we can compute the improvement $I_c(\lambda)$

Expected Improvement (EI): Concept



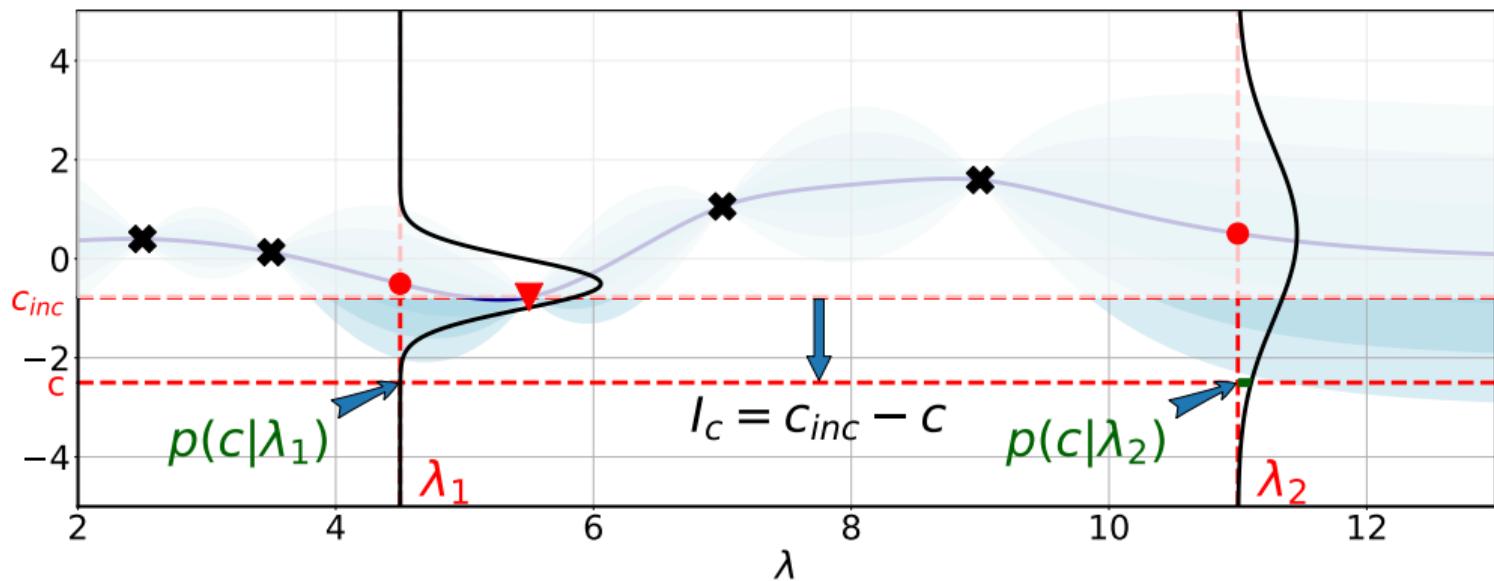
Given $\hat{c}(\lambda) = \mathcal{N}(\mu(\lambda), \sigma^2(\lambda))$, we can also compute $p(c|\lambda)$.

Expected Improvement (EI): Concept



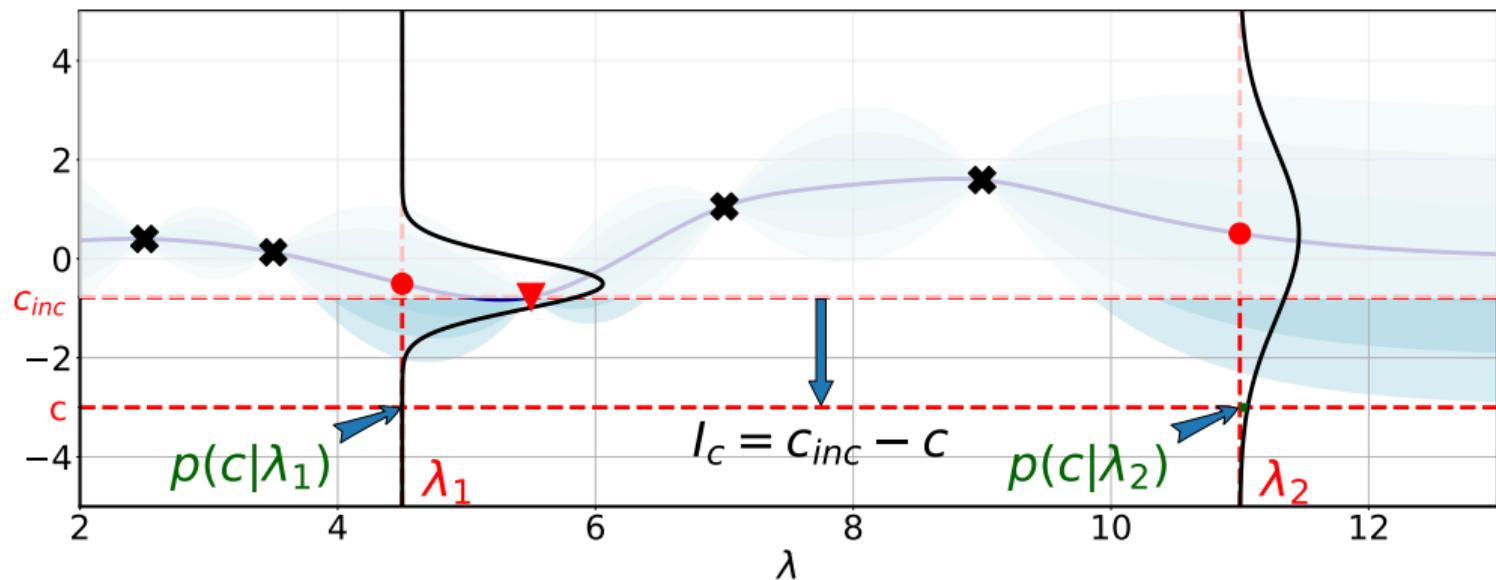
Compare the likelihood of a given improvement for two different configurations λ_1 and λ_2

Expected Improvement (EI): Concept



Now consider the likelihood of a larger improvement.

Expected Improvement (EI): Concept



Larger improvements are more likely in areas of high uncertainty.

To compute $\mathbb{E}[I(\lambda)]$, intuitively, we sum $p(c | \lambda) \times I_c$ over all possible values of c .

Expected Improvement (EI): Formal Definition

- We define the one-step positive improvement over the current incumbent as

$$I^{(t)}(\boldsymbol{\lambda}) = \max(0, c_{inc} - c(\boldsymbol{\lambda}))$$

- Expected Improvement is then defined as

$$u_{EI}^{(t)}(\boldsymbol{\lambda}) = \mathbb{E}[I^{(t)}(\boldsymbol{\lambda})] = \int_{-\infty}^{\infty} p^{(t)}(c \mid \boldsymbol{\lambda}) \times I^{(t)}(\boldsymbol{\lambda}) \ dc.$$

Expected Improvement (EI): Formal Definition

- We define the one-step positive improvement over the current incumbent as

$$I^{(t)}(\boldsymbol{\lambda}) = \max(0, c_{inc} - c(\boldsymbol{\lambda}))$$

- Expected Improvement is then defined as

$$u_{EI}^{(t)}(\boldsymbol{\lambda}) = \mathbb{E}[I^{(t)}(\boldsymbol{\lambda})] = \int_{-\infty}^{\infty} p^{(t)}(c \mid \boldsymbol{\lambda}) \times I^{(t)}(\boldsymbol{\lambda}) \ dc.$$

- Since the posterior distribution of $\hat{c}(\boldsymbol{\lambda})$ is a Gaussian, EI can be computed in closed form (see exercise):

$$u_{EI}^{(t)}(\boldsymbol{\lambda}) = \begin{cases} \sigma^{(t)}(\boldsymbol{\lambda})[Z\Phi(Z) + \phi(Z)], & \text{if } \sigma^{(t)}(\boldsymbol{\lambda}) > 0 \\ 0 & \text{if } \sigma^{(t)}(\boldsymbol{\lambda}) = 0, \end{cases}$$

where $Z = \frac{c_{inc} - \mu^{(t)}(\boldsymbol{\lambda}) - \xi}{\sigma^{(t)}(\boldsymbol{\lambda})}$ and ξ is an optional exploration parameter.

Expected Improvement (EI): Formal Definition

- We define the one-step positive improvement over the current incumbent as

$$I^{(t)}(\boldsymbol{\lambda}) = \max(0, c_{inc} - c(\boldsymbol{\lambda}))$$

- Expected Improvement is then defined as

$$u_{EI}^{(t)}(\boldsymbol{\lambda}) = \mathbb{E}[I^{(t)}(\boldsymbol{\lambda})] = \int_{-\infty}^{\infty} p^{(t)}(c \mid \boldsymbol{\lambda}) \times I^{(t)}(\boldsymbol{\lambda}) \ dc.$$

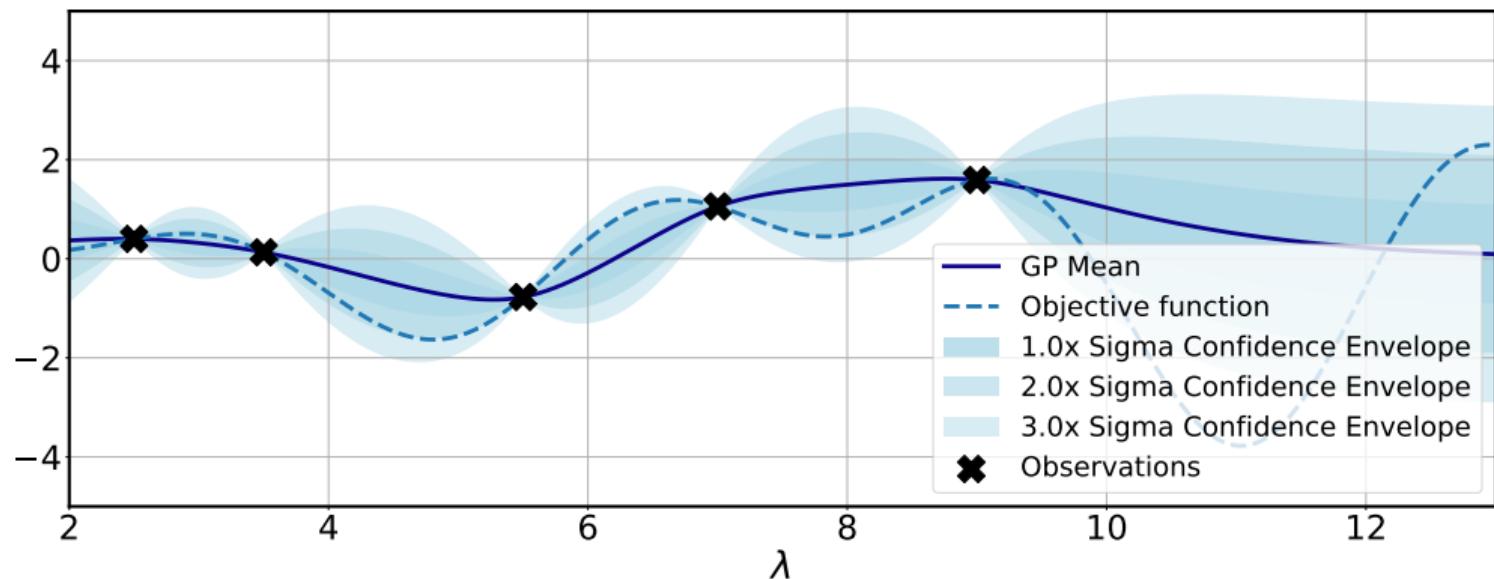
- Since the posterior distribution of $\hat{c}(\boldsymbol{\lambda})$ is a Gaussian, EI can be computed in closed form (see exercise):

$$u_{EI}^{(t)}(\boldsymbol{\lambda}) = \begin{cases} \sigma^{(t)}(\boldsymbol{\lambda})[Z\Phi(Z) + \phi(Z)], & \text{if } \sigma^{(t)}(\boldsymbol{\lambda}) > 0 \\ 0 & \text{if } \sigma^{(t)}(\boldsymbol{\lambda}) = 0, \end{cases}$$

where $Z = \frac{c_{inc} - \mu^{(t)}(\boldsymbol{\lambda}) - \xi}{\sigma^{(t)}(\boldsymbol{\lambda})}$ and ξ is an optional exploration parameter.

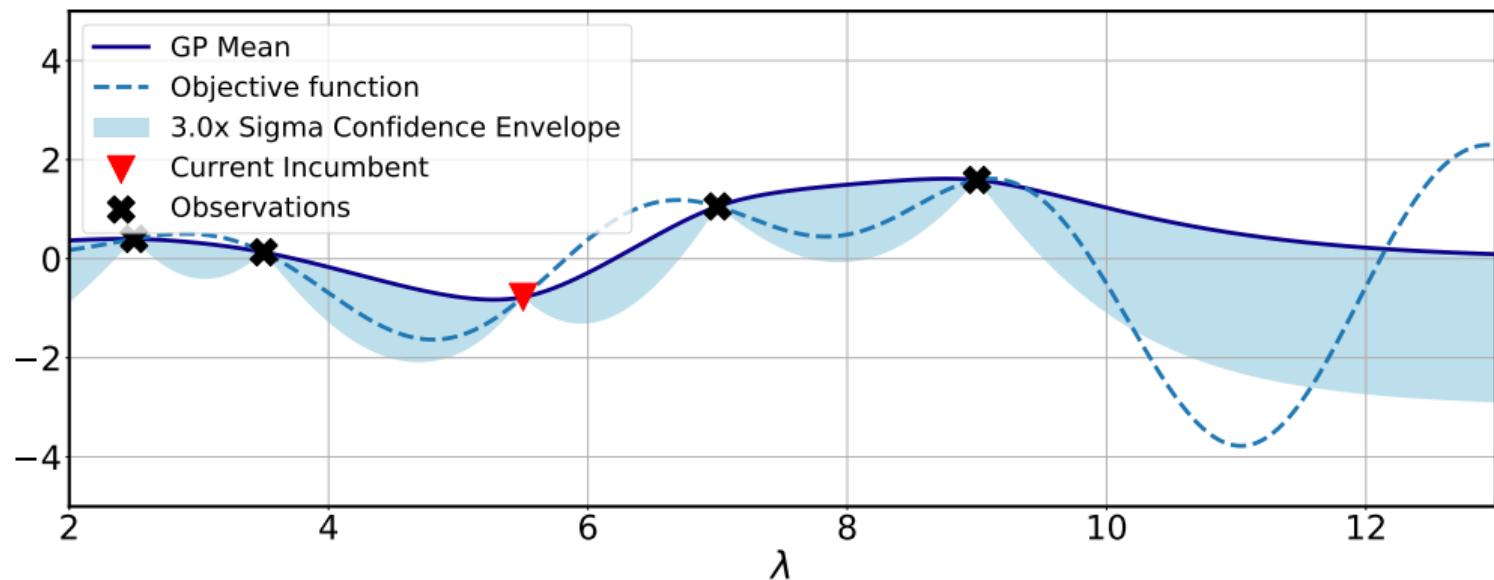
Choose $\boldsymbol{\lambda}^{(t)} \in \arg \max_{\boldsymbol{\lambda} \in \Lambda} (u_{EI}^{(t)}(\boldsymbol{\lambda}))$

Lower/Upper Confidence Bounds (LCB/UCB): Concept



Given the surrogate fit at iteration t

Lower/Upper Confidence Bounds (LCB/UCB): Concept



Lower Confidence Bound, $\mu(\lambda) - \alpha\sigma(\lambda)$ (here, for $\alpha = 3$)

Lower/Upper Confidence Bounds (LCB/UCB): Formal Definition

- We define the Lower Confidence Bound as

$$u_{LCB}^{(t)}(\boldsymbol{\lambda}) = \mu^{(t)}(\boldsymbol{\lambda}) - \alpha \sigma^{(t)}(\boldsymbol{\lambda}), \quad \alpha \geq 0$$

- One can schedule α (e.g., increase it over time [Srinivas et al. 2009])

Choose $\boldsymbol{\lambda}^{(t)} \in \arg \max_{\boldsymbol{\lambda} \in \Lambda} \left(-u_{LCB}^{(t)}(\boldsymbol{\lambda}) \right)$

Lower/Upper Confidence Bounds (LCB/UCB): Formal Definition

- We define the **Lower Confidence Bound** as

$$u_{LCB}^{(t)}(\boldsymbol{\lambda}) = \mu^{(t)}(\boldsymbol{\lambda}) - \alpha \sigma^{(t)}(\boldsymbol{\lambda}), \quad \alpha \geq 0$$

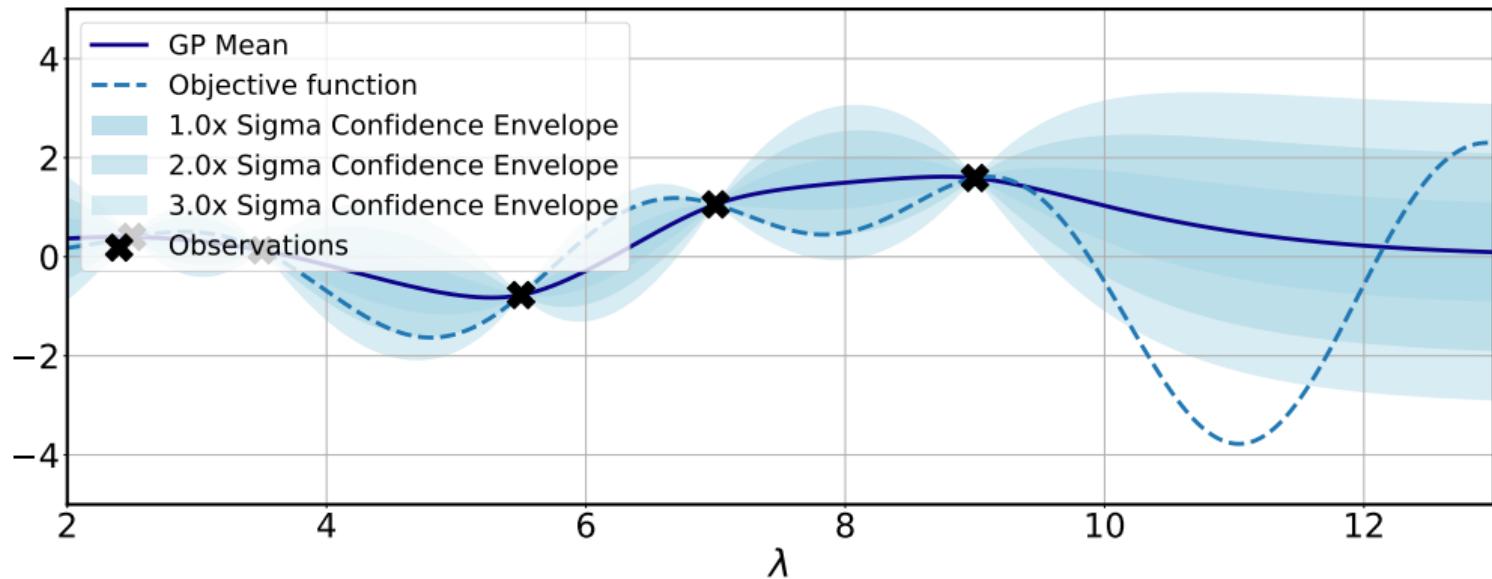
- One can schedule α (e.g., increase it over time [Srinivas et al. 2009])

Choose $\boldsymbol{\lambda}^{(t)} \in \arg \max_{\boldsymbol{\lambda} \in \Lambda} \left(-u_{LCB}^{(t)}(\boldsymbol{\lambda}) \right)$

- Note: when one aims to **maximize** the objective function, one would use UCB instead

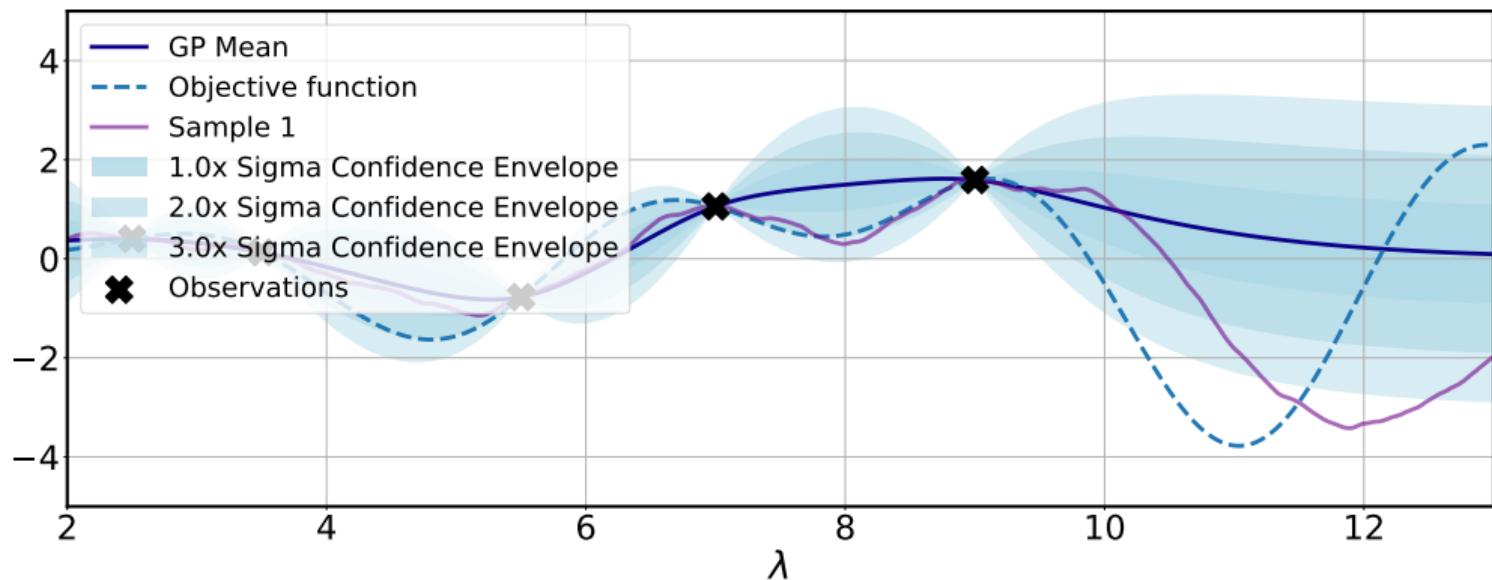
- ▶ $u_{UCB}^{(t)}(\boldsymbol{\lambda}) = \mu^{(t)}(\boldsymbol{\lambda}) + \alpha \sigma^{(t)}(\boldsymbol{\lambda})$
- ▶ For UCB, one would choose $\boldsymbol{\lambda}^{(t)} \in \arg \max_{\boldsymbol{\lambda} \in \Lambda} (u_{UCB}^{(t)}(\boldsymbol{\lambda}))$

Thompson Sampling (TS): Concept



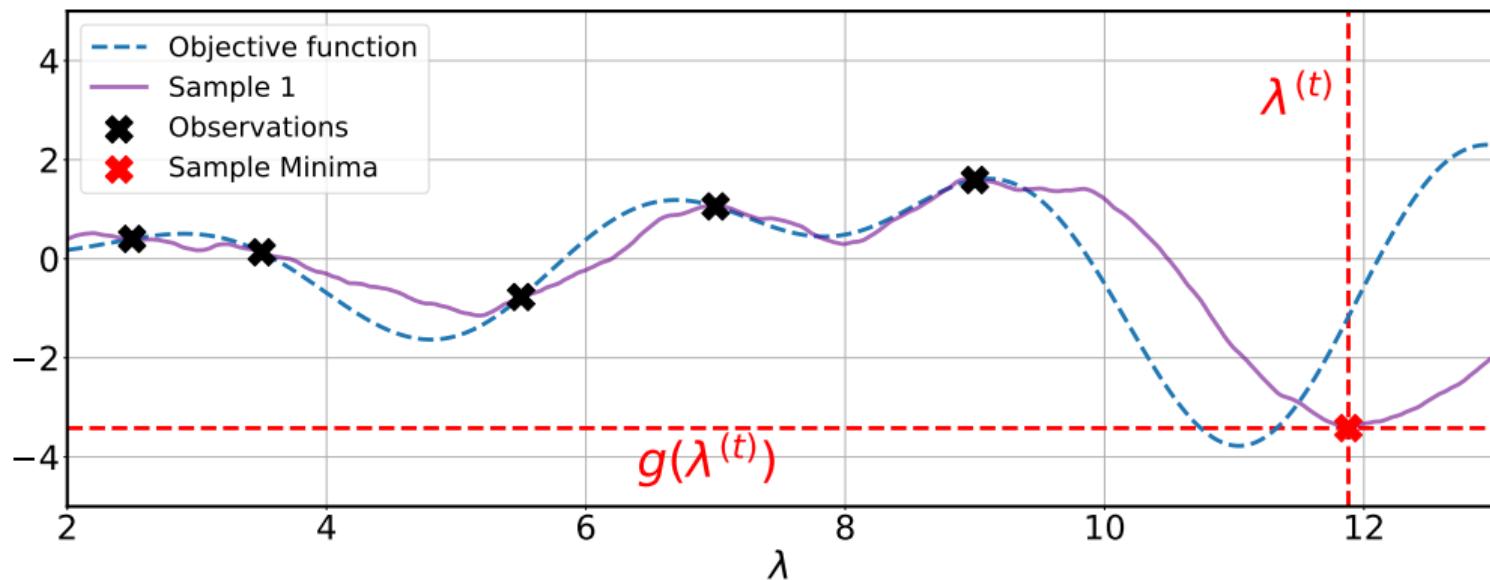
Given the surrogate at iteration t fit on dataset $\mathcal{D}^{(t-1)}$

Thompson Sampling (TS): Concept



Draw a sample g from the predictive surrogate model

Thompson Sampling (TS): Concept



Then choose the minimum of this sample to evaluate at next

Thompson Sampling (TS): Pseudocode

Bayesian Optimization using Thompson Sampling

Require: Search space Λ , cost function c , surrogate model \hat{c} , maximal number of function evaluations T

Result : Best observed configuration $\hat{\lambda}$ according to $\mathcal{D}^{(T)}$ or \mathcal{G}

- 1 Initialize data $\mathcal{D}^{(0)}$ with initial observations
 - 2 **for** $t = 1$ **to** T **do**
 - 3 Fit predictive model $\hat{c}^{(t)}$ on $\mathcal{D}^{(t-1)}$
 - 4 Sample a function from the surrogate: $g \sim \hat{c}^{(t)}$
 - 5 Select next query point: $\lambda^{(t)} \in \arg \min_{\lambda \in \Lambda} g(\lambda)$
 - 6 Query $c(\lambda^{(t)})$
 - 7 Update data: $\mathcal{D}^{(t)} \leftarrow \mathcal{D}^{(t-1)} \cup \{\langle \lambda^{(t)}, c(\lambda^{(t)}) \rangle\}$
-

Questions to Answer for Yourself / Discuss with Friends

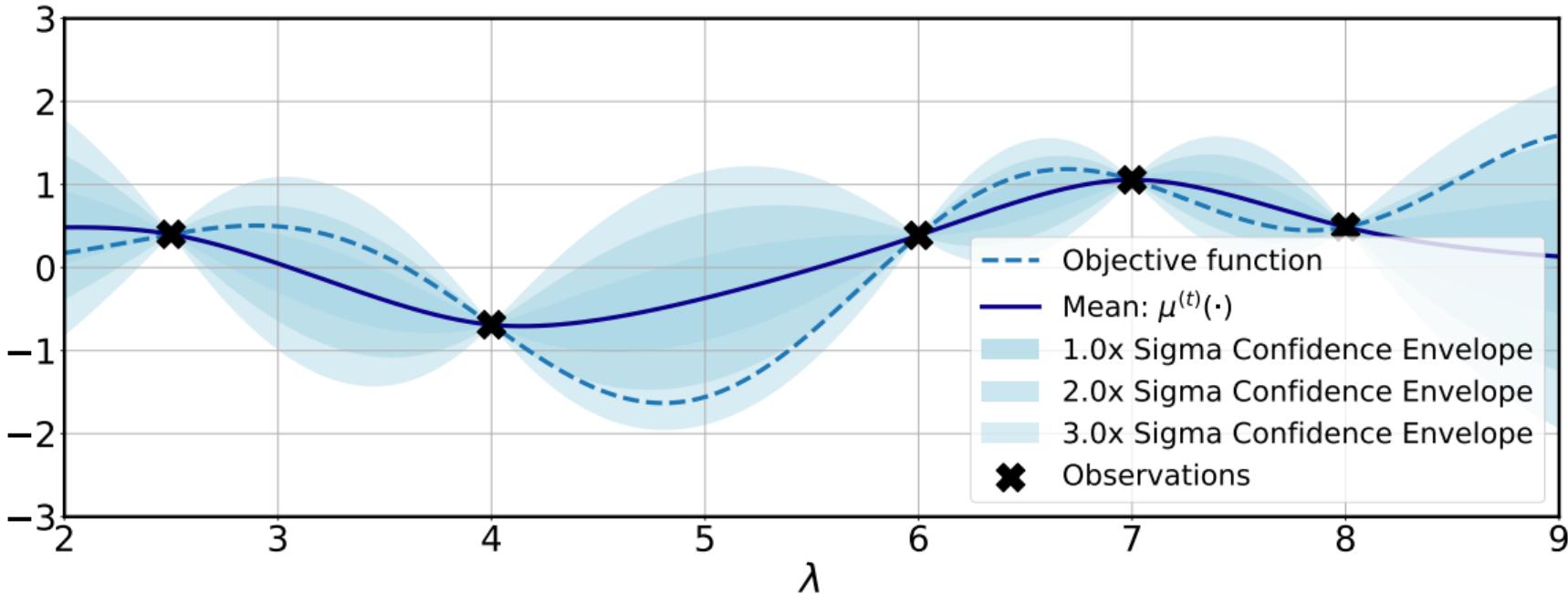
- Discussion. How would you set the exploration parameter ξ for PI if you want to avoid too incremental improvements?
- Derivation. Derive the closed form solution of expected improvement.
- Discussion. In which situations would EI perform substantially differently than PI?

AutoML: Bayesian Optimization for HPO

Computationally Expensive Acquisition Functions

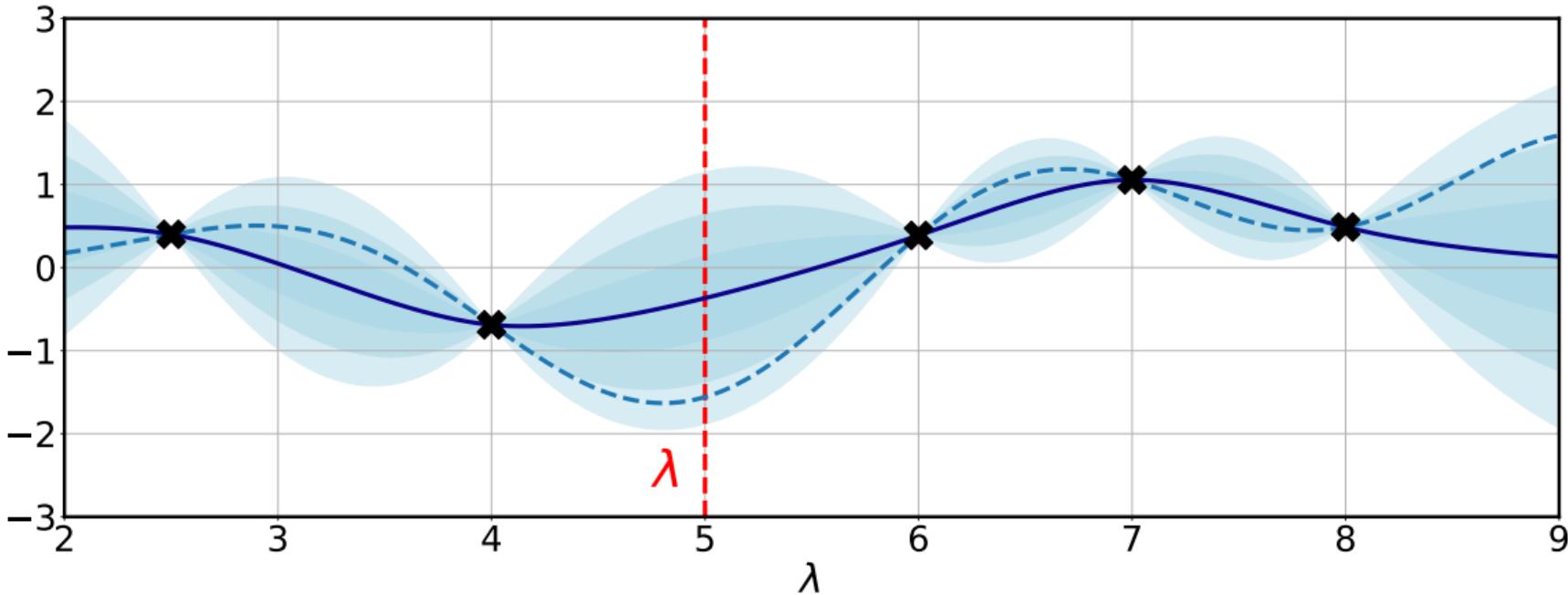
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

A Computationally Expensive Step: One-Step Look Ahead



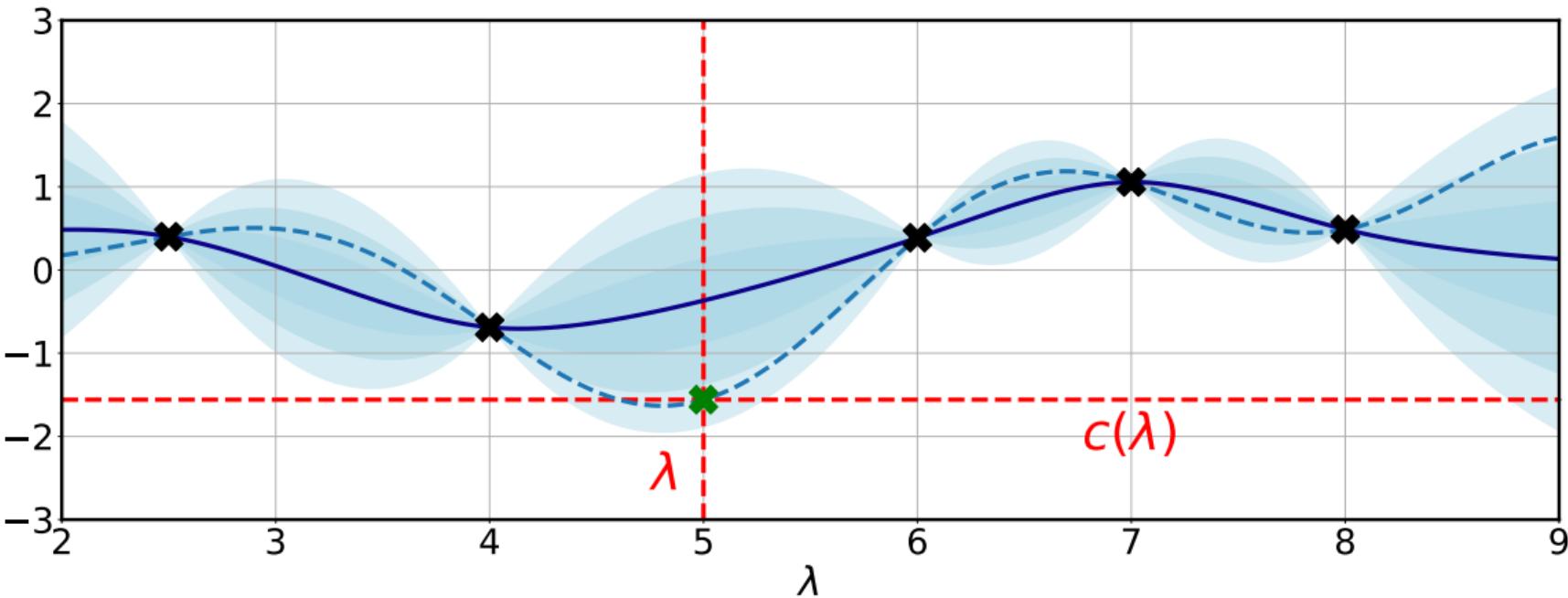
Given the surrogate $\hat{c}^{(t)}$ fit at iteration t

A Computationally Expensive Step: One-Step Look Ahead



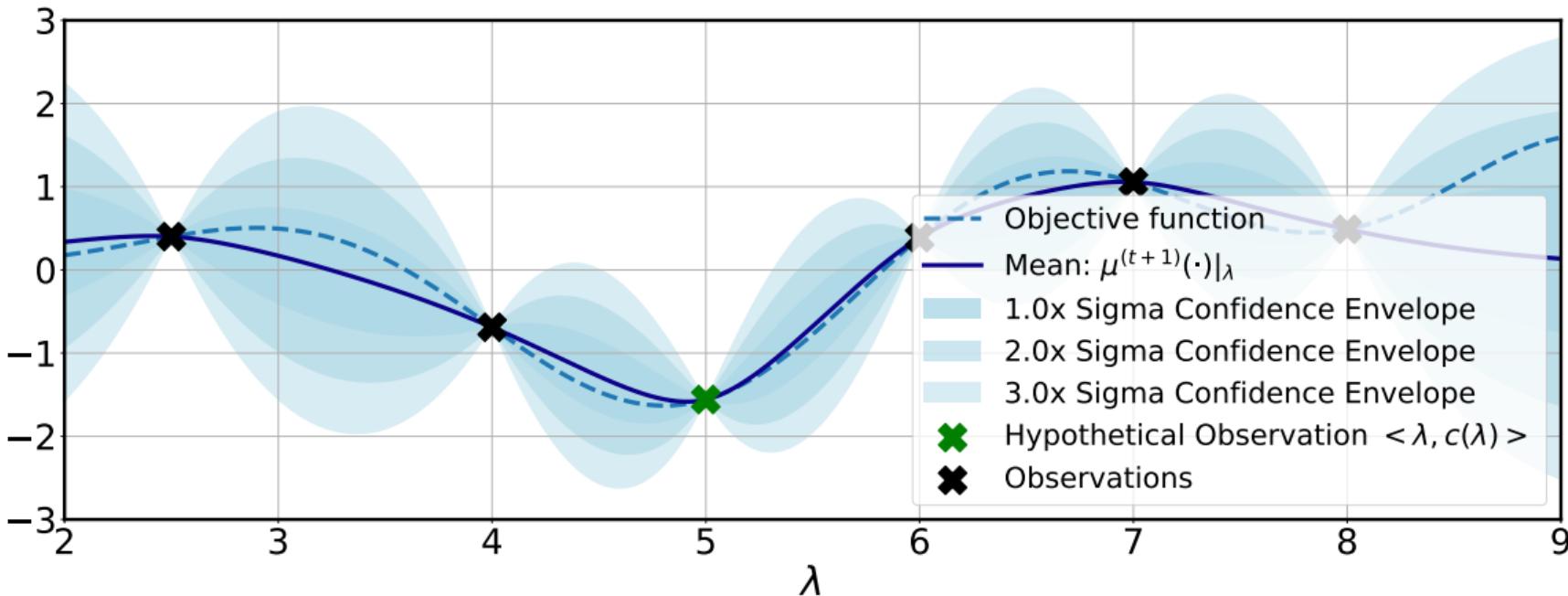
Imagine that we sample at a random configuration λ

A Computationally Expensive Step: One-Step Look Ahead



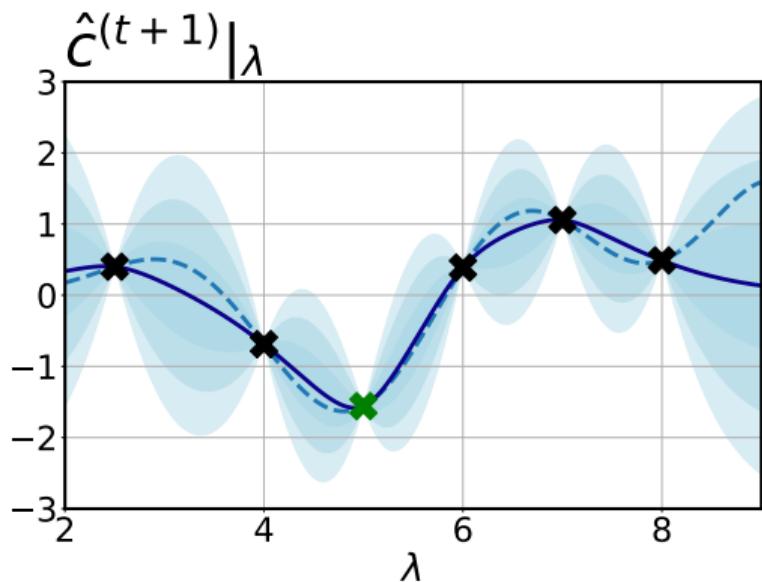
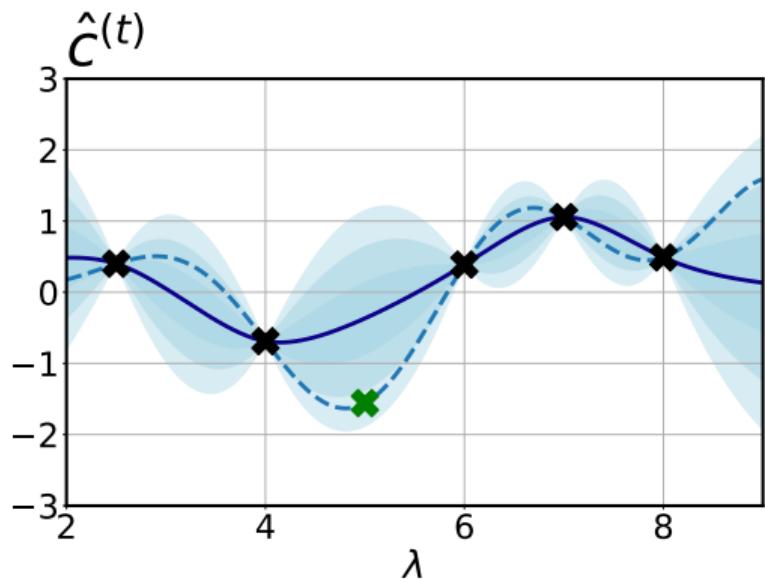
We would then observe the cost $c(\lambda)$ at this imaginary configuration λ

A Computationally Expensive Step: One-Step Look Ahead



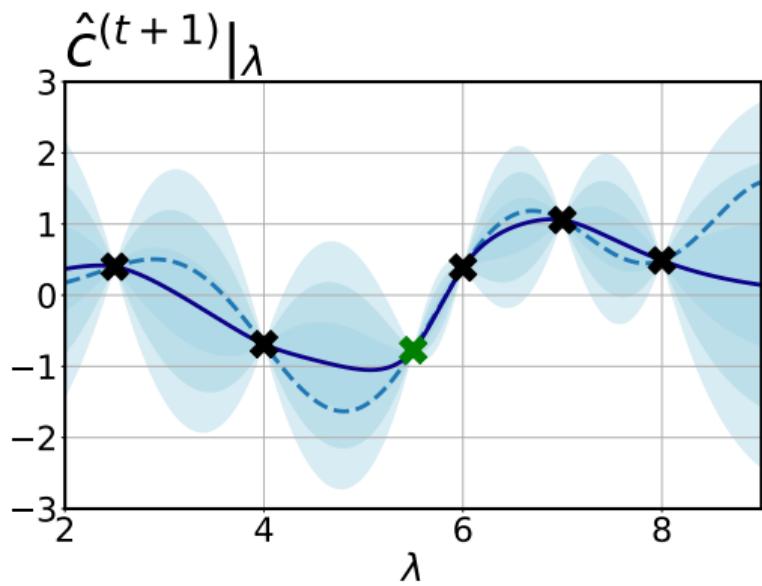
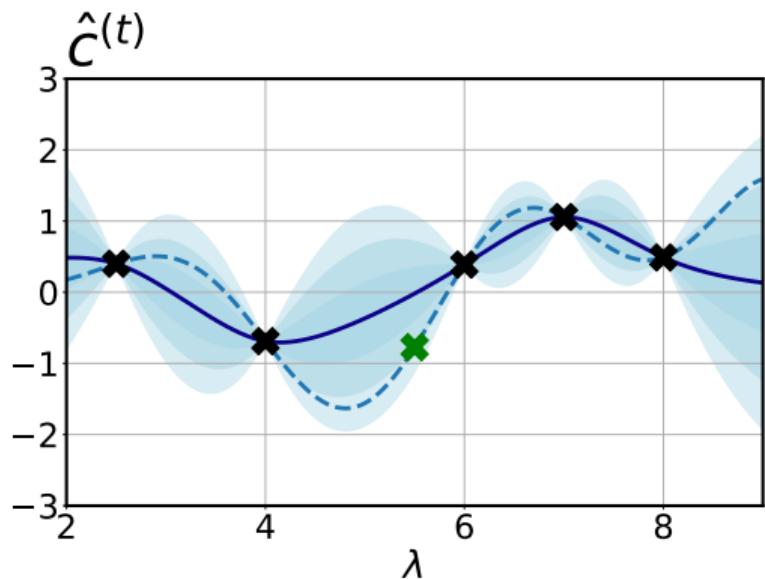
With this hypothetical data point at λ , we'd have this 1-step lookahead surrogate $\hat{c}^{(t+1)}|_{\lambda}(\cdot)$

Visualization of How Different the Lookahead Surrogate Can Be



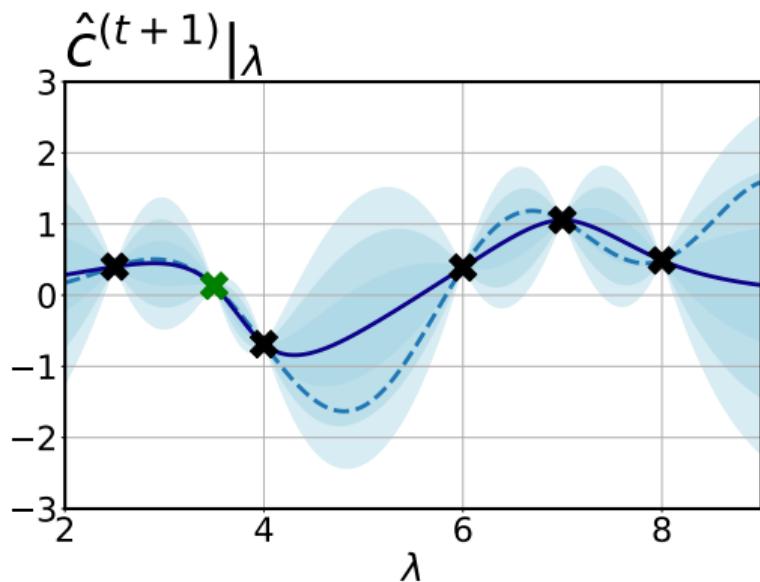
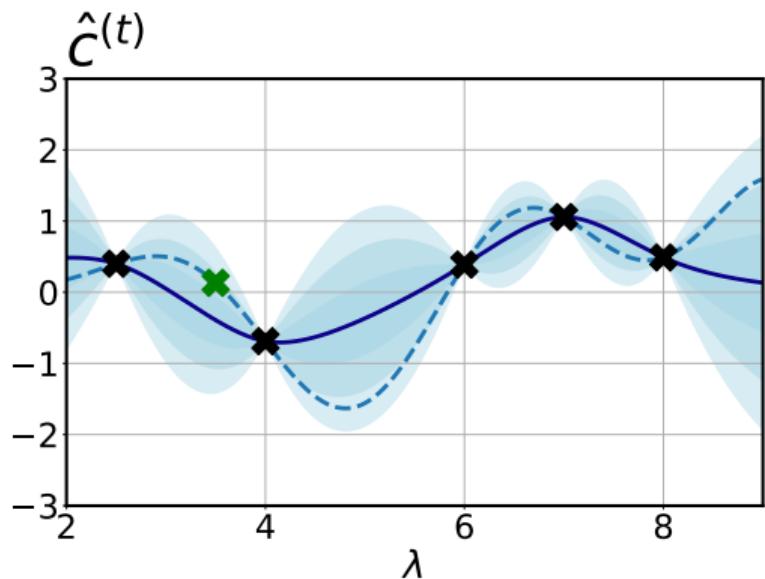
A comparison of $\hat{c}^{(t)}(\cdot)$ and $\hat{c}^{(t+1)}|_{\lambda}(\cdot)$ for a given λ .

Visualization of How Different the Lookahead Surrogate Can Be



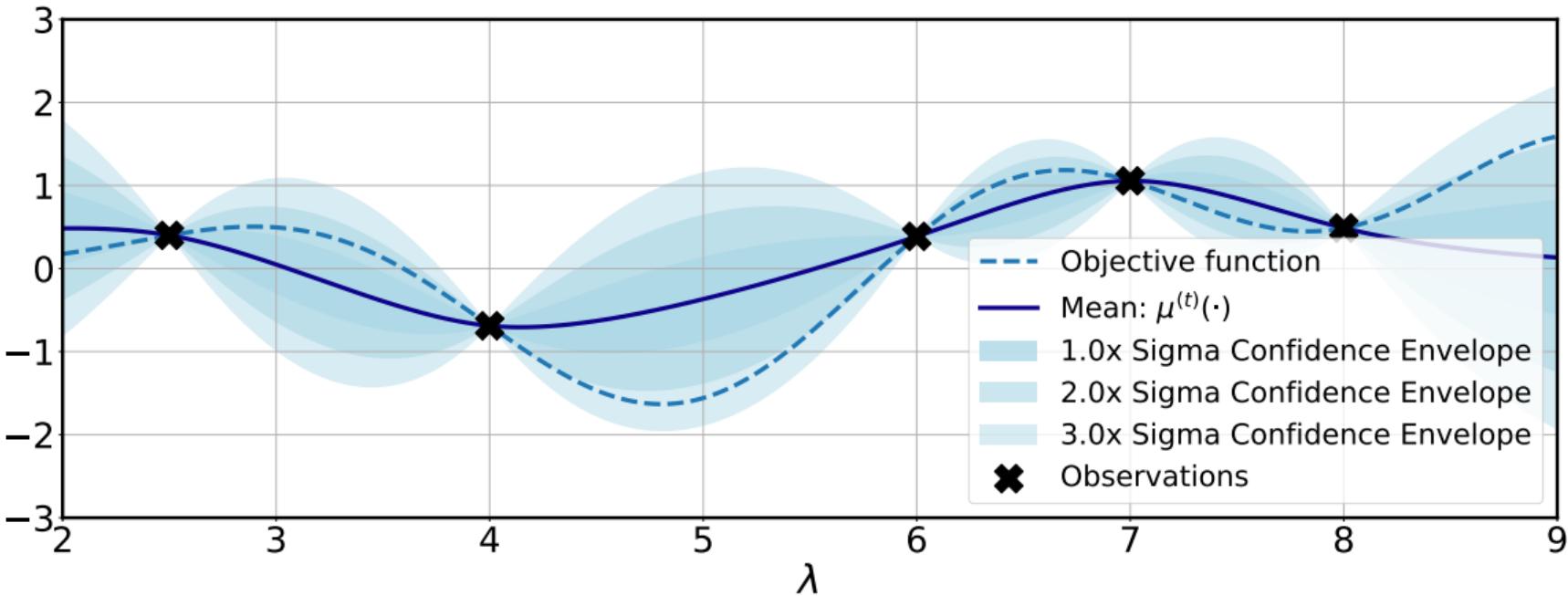
A comparison of $\hat{C}^{(t)}(\cdot)$ and $\hat{C}^{(t+1)}|_{\lambda}(\cdot)$ for a given λ .

Visualization of How Different the Lookahead Surrogate Can Be



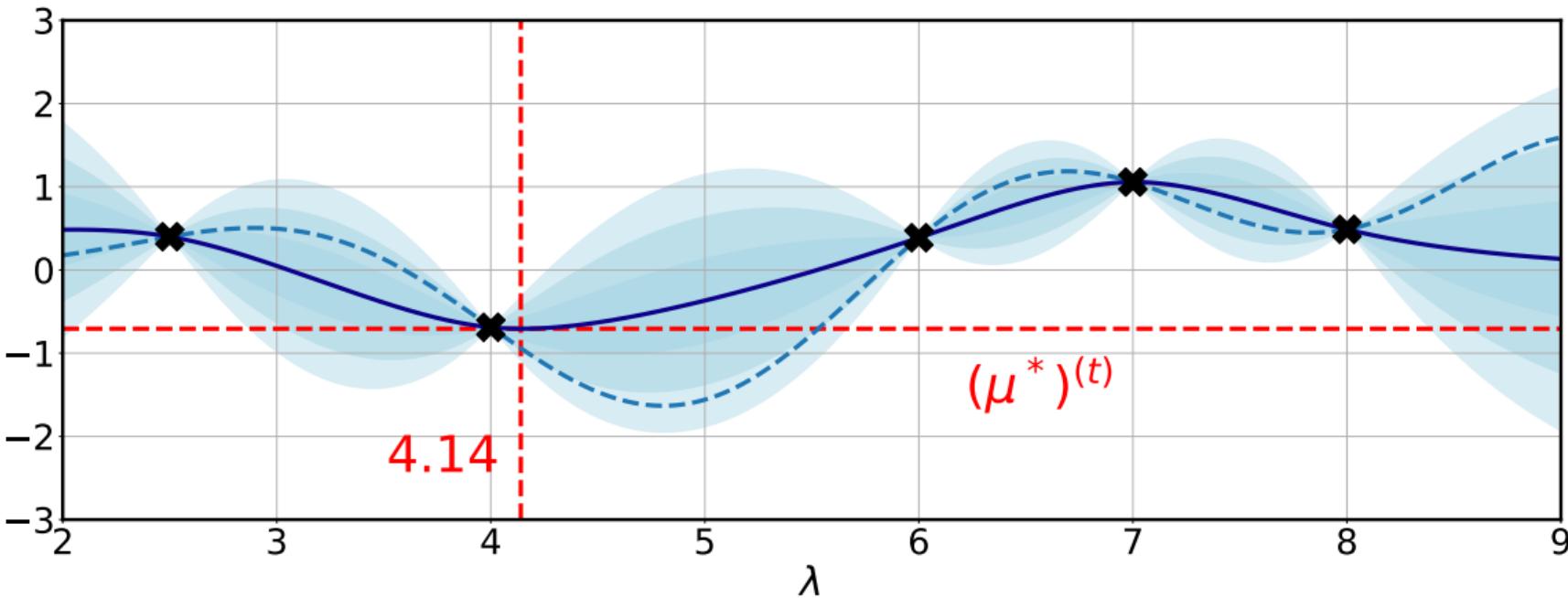
A comparison of $\hat{c}^{(t)}(\cdot)$ and $\hat{c}^{(t+1)}|_{\lambda}(\cdot)$ for a given λ .

Knowledge Gradient (KG): Concept



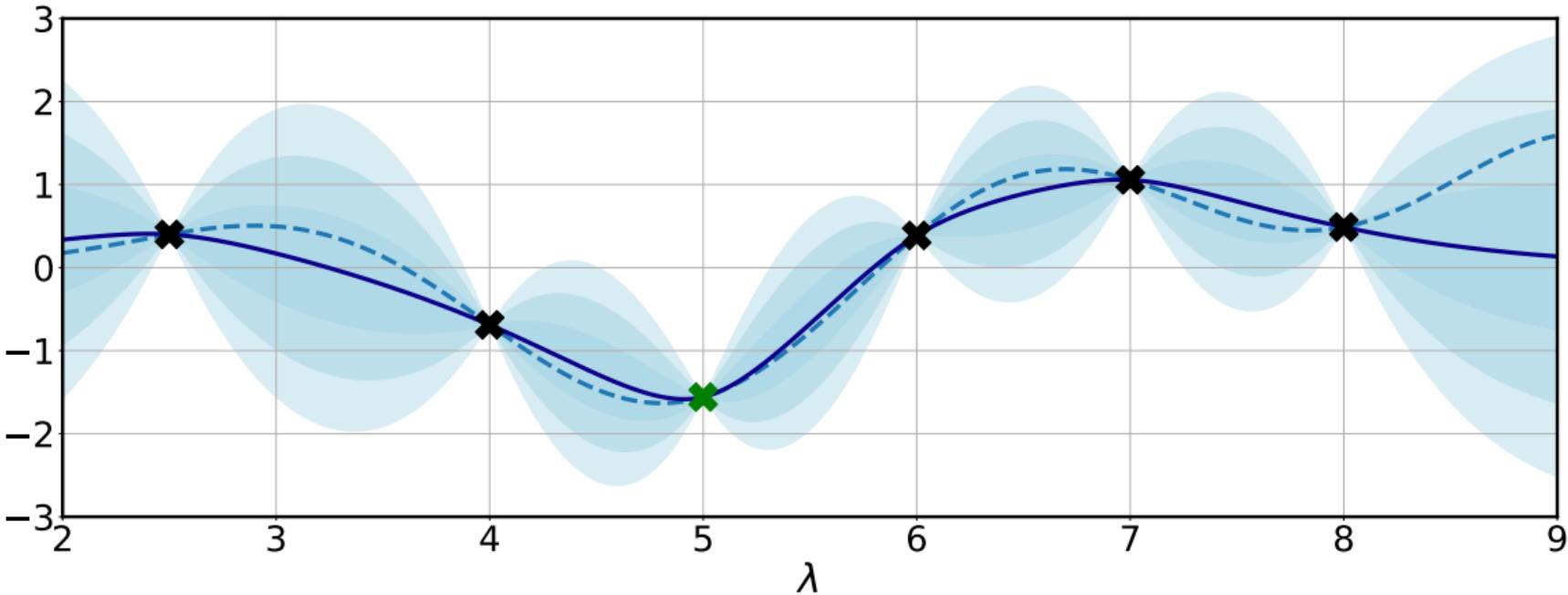
Given the surrogate $\hat{c}(\lambda) = \mathcal{N}(\mu(\lambda), \sigma^2(\lambda))$ fit at iteration t

Knowledge Gradient (KG): Concept



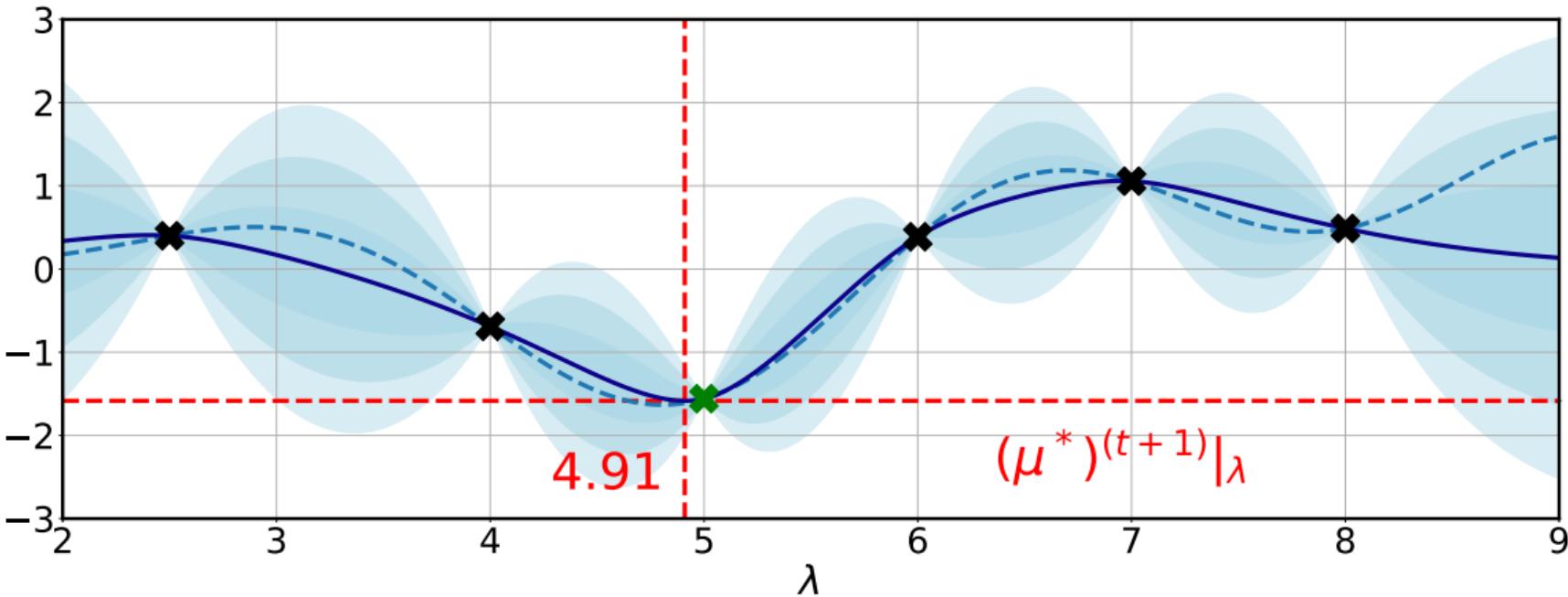
If we are risk-neutral, we'd return $\arg \min_{\lambda} (\mu(\lambda))^{(t)}$ as incumbent, with value $(\mu^*)^{(t)}$

Knowledge Gradient (KG): Concept



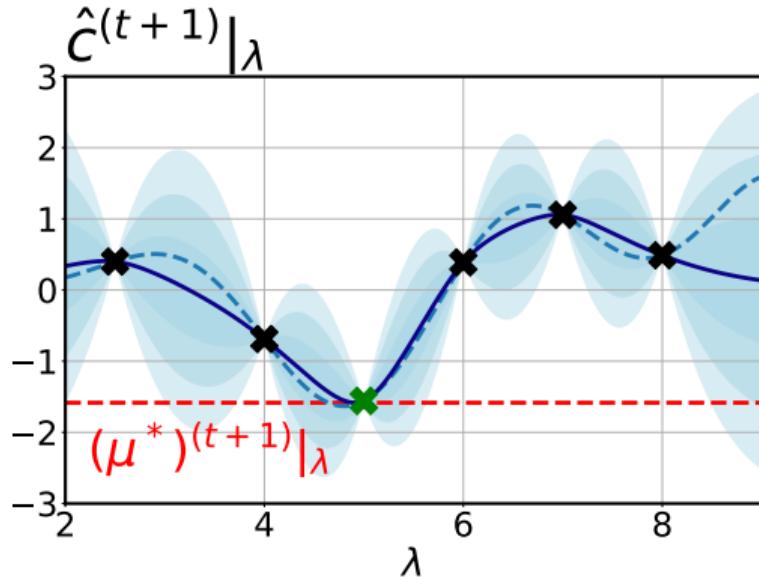
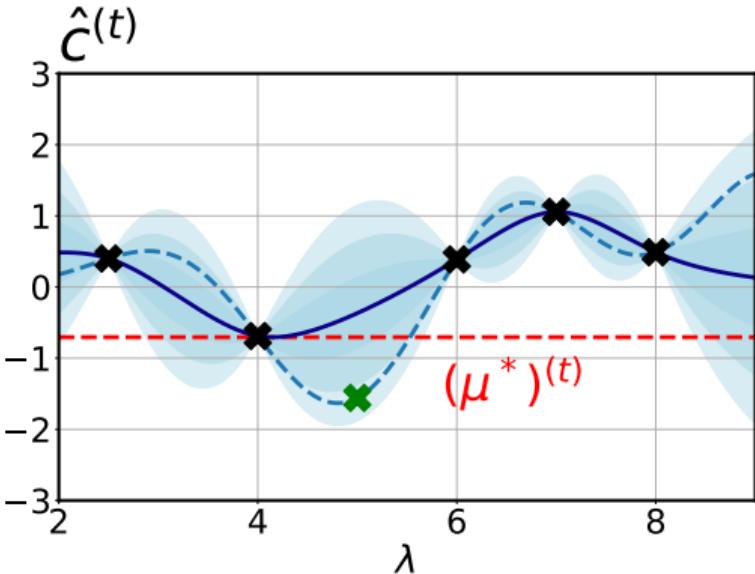
If we perform a one-step look-ahead for configuration λ , we would get $\hat{c}^{(t+1)}|_{\lambda}$

Knowledge Gradient (KG): Concept



We would then be interested in the minimum of the updated mean function $(\mu^*)^{(t+1)} | \lambda$

Knowledge Gradient (KG): Concept



The Knowledge Gradient is then the expectation of the improvement $(\mu^*)^{(t+1)} - (\mu^*)^{(t+1)}|_{\lambda}$

Knowledge Gradient (KG): Formal Definition

- The Knowledge Gradient is the expectation of the improvement $(\mu^*)^{(t+1)} - (\mu^*)^{(t+1)}|_{\lambda}$:

$$\begin{aligned} u_{KG}^{(t)}(\boldsymbol{\lambda}) &= \mathbb{E} \left[(\mu^*)^{(t)} - (\mu^*)^{(t+1)} \Big|_{\boldsymbol{\lambda}^{(t)} = \boldsymbol{\lambda}} \right] \\ &= \min_{\boldsymbol{\lambda}' \in \Lambda} \mu^{(t)} \left(\boldsymbol{\lambda}' | \mathcal{D}^{(t-1)} \right) - \mathbb{E}_{\tilde{c} \sim \hat{c}(\boldsymbol{\lambda})^{(t)}} \left[\min_{\boldsymbol{\lambda}' \in \Lambda} \mu^{(t+1)} \left(\boldsymbol{\lambda}' | \mathcal{D}^{(t-1)} \cup \{ \langle \boldsymbol{\lambda}, \tilde{c} \rangle \} \right) \right] \end{aligned}$$

Knowledge Gradient (KG): Formal Definition

- The Knowledge Gradient is the expectation of the improvement $(\mu^*)^{(t+1)} - (\mu^*)^{(t+1)}|_{\lambda}$:

$$\begin{aligned} u_{KG}^{(t)}(\boldsymbol{\lambda}) &= \mathbb{E} \left[(\mu^*)^{(t)} - (\mu^*)^{(t+1)} \Big|_{\boldsymbol{\lambda}^{(t)} = \boldsymbol{\lambda}} \right] \\ &= \min_{\boldsymbol{\lambda}' \in \Lambda} \mu^{(t)} \left(\boldsymbol{\lambda}' | \mathcal{D}^{(t-1)} \right) - \mathbb{E}_{\tilde{c} \sim \hat{c}(\boldsymbol{\lambda})^{(t)}} \left[\min_{\boldsymbol{\lambda}' \in \Lambda} \mu^{(t+1)} \left(\boldsymbol{\lambda}' | \mathcal{D}^{(t-1)} \cup \{ \langle \boldsymbol{\lambda}, \tilde{c} \rangle \} \right) \right] \end{aligned}$$

Choose $\boldsymbol{\lambda}^{(t)} = \arg \max_{\boldsymbol{\lambda} \in \Lambda} (u_{KG}^{(t)}(\boldsymbol{\lambda}))$

Knowledge Gradient: Pseudocode for Monte Carlo Approximation

$$u_{KG}^{(t)}(\boldsymbol{\lambda}) = \text{const} - \mathbb{E}_{\tilde{c} \sim \hat{c}(\boldsymbol{\lambda})^{(t)}} \left[\min_{\boldsymbol{\lambda}' \in \Lambda} \mu^{(t+1)} \left(\boldsymbol{\lambda}' \mid \mathcal{D}^{(t-1)} \cup \{\langle \boldsymbol{\lambda}, \tilde{c} \rangle\} \right) \right]$$

Sampling Based Knowledge Gradient Acquisition Function

Require: Surrogate \hat{c} , candidate configuration $\boldsymbol{\lambda}$, dataset \mathcal{D}

Result : Utility $u(\boldsymbol{\lambda})$

- 1 **for** $s = 1$ **to** S **do**
 - 2 Sample $\tilde{c}_s \sim \hat{c}(\boldsymbol{\lambda})$
 - 3 Update \hat{c} with $\{\langle \boldsymbol{\lambda}, \tilde{c}_s \rangle\}$ to yield $\hat{c}_s = \mathcal{N}(\mu_s, \sigma_s^2)$
 - 4 $e[s] \leftarrow \min_{\boldsymbol{\lambda}' \in \Lambda} \mu_s$
 - 5 $u \leftarrow \text{const} - \frac{1}{S} \sum_{s=1}^S e[s]$
-

Knowledge Gradient: Pseudocode for Monte Carlo Approximation

$$u_{KG}^{(t)}(\boldsymbol{\lambda}) = \text{const} - \mathbb{E}_{\tilde{c} \sim \hat{c}(\boldsymbol{\lambda})^{(t)}} \left[\min_{\boldsymbol{\lambda}' \in \Lambda} \mu^{(t+1)} \left(\boldsymbol{\lambda}' \mid \mathcal{D}^{(t-1)} \cup \{ \langle \boldsymbol{\lambda}, \tilde{c} \rangle \} \right) \right]$$

Sampling Based Knowledge Gradient Acquisition Function

Require: Surrogate \hat{c} , candidate configuration $\boldsymbol{\lambda}$, dataset \mathcal{D}

Result : Utility $u(\boldsymbol{\lambda})$

- 1 **for** $s = 1$ **to** S **do**
 - 2 Sample $\tilde{c}_s \sim \hat{c}(\boldsymbol{\lambda})$
 - 3 Update \hat{c} with $\{ \langle \boldsymbol{\lambda}, \tilde{c}_s \rangle \}$ to yield $\hat{c}_s = \mathcal{N}(\mu_s, \sigma_s^2)$
 - 4 $e[s] \leftarrow \min_{\boldsymbol{\lambda}' \in \Lambda} \mu_s$
 - 5 $u \leftarrow \text{const} - \frac{1}{S} \sum_{s=1}^S e[s]$
-

This sampling view is useful for intuition;
but in practice, there are more efficient ways to optimize KG [Frazier 2018]

Entropy Search Preliminaries

- Key idea: Evaluate λ which most reduces our uncertainty about the location of λ^*

Entropy Search Preliminaries

- Key idea: Evaluate λ which most reduces our uncertainty about the location of λ^*
- We'll use the p_{min} distribution to characterize the location of λ^* :

$$p_{min}(\lambda | \mathcal{D}) = p(\lambda \in \arg \min_{\lambda' \in \Lambda} (\hat{c}(\lambda') | \mathcal{D}))$$

Entropy Search Preliminaries

- Key idea: Evaluate λ which most reduces our uncertainty about the location of λ^*
- We'll use the p_{min} distribution to characterize the location of λ^* :

$$p_{min}(\lambda | \mathcal{D}) = p(\lambda \in \arg \min_{\lambda' \in \Lambda} (\hat{c}(\lambda') | \mathcal{D}))$$

- Our uncertainty is then captured by the entropy $H(p_{min}(\cdot | \mathcal{D}))$ of the p_{min} distribution

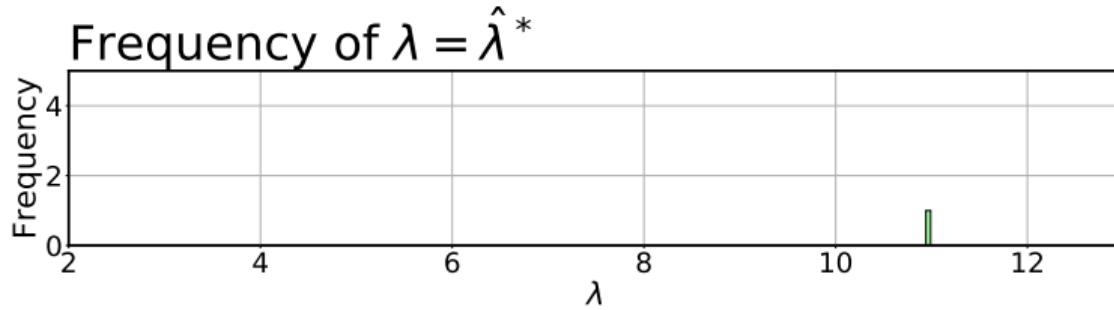
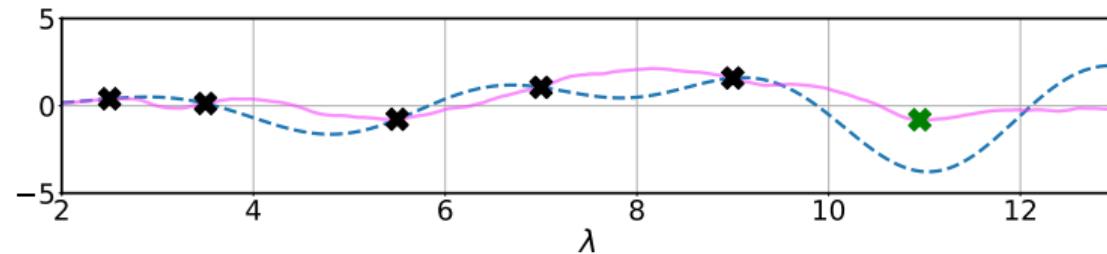
Entropy Search Preliminaries

- Key idea: Evaluate λ which most reduces our uncertainty about the location of λ^*
- We'll use the p_{min} distribution to characterize the location of λ^* :

$$p_{min}(\lambda | \mathcal{D}) = p(\lambda \in \arg \min_{\lambda' \in \Lambda} (\hat{c}(\lambda') | \mathcal{D}))$$

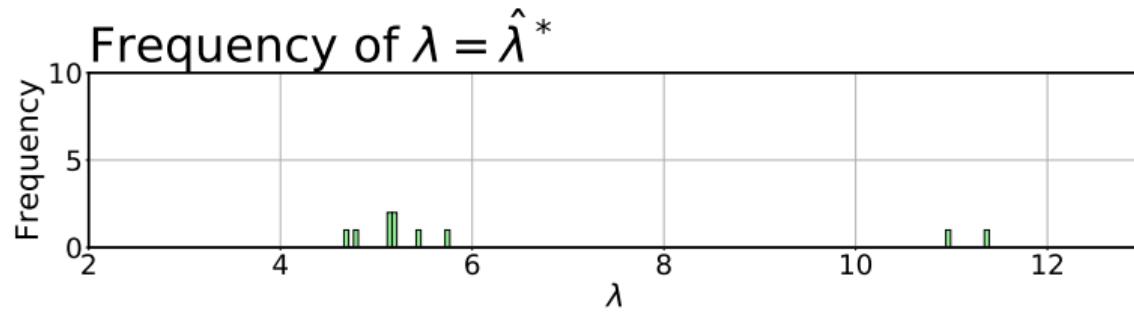
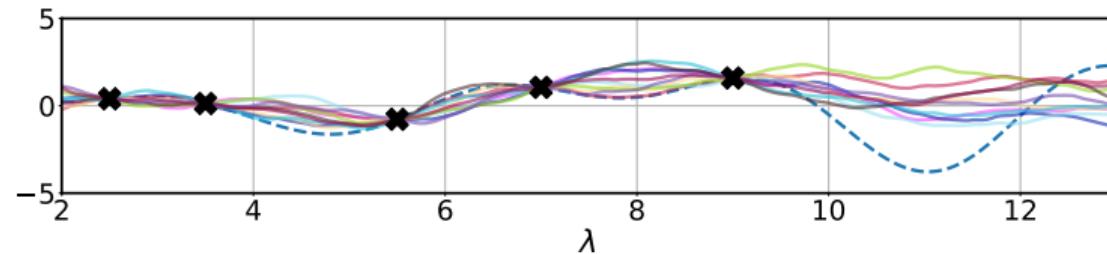
- Our uncertainty is then captured by the entropy $H(p_{min}(\cdot | \mathcal{D}))$ of the p_{min} distribution
- Minimizing $H(p_{min}(\cdot | \mathcal{D}))$ yields a peaked p_{min} distribution, i.e., strong knowledge about the location of λ^*

Entropy Search: Visualization of the p_{min} Distribution



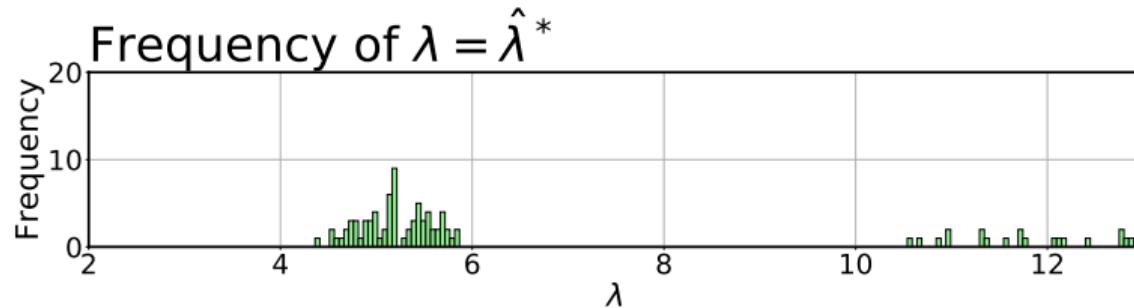
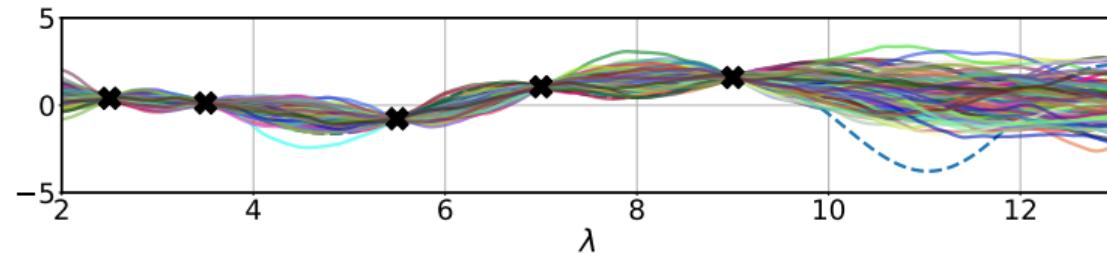
For each sample drawn from \hat{c} , we can compute where λ^* lies

Entropy Search: Visualization of the p_{min} Distribution



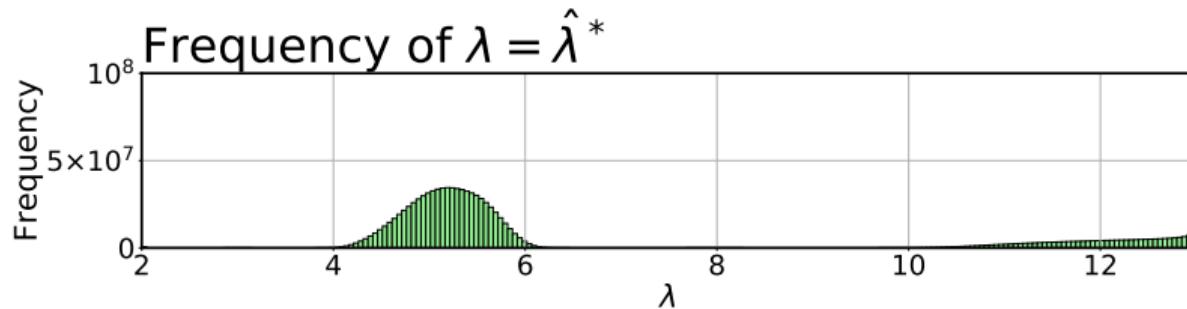
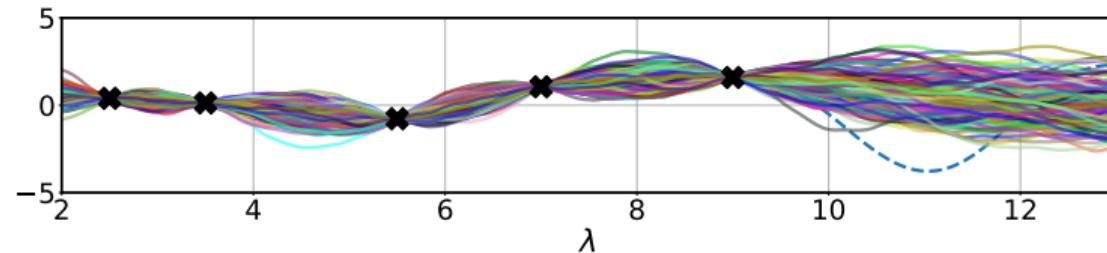
For each sample drawn from \hat{c} , we can compute where λ^* lies

Entropy Search: Visualization of the p_{min} Distribution



From many samples we can approximate the p_{min} distribution

Entropy Search: Visualization of the p_{min} Distribution



From many samples we can approximate the p_{min} distribution

Entropy Search: Formal Definition

- The p_{min} distribution characterizes the location of λ^* :

$$p_{min}(\lambda^* | \mathcal{D}) = p(\lambda^* \in \arg \min_{\lambda' \in \Lambda} (\hat{c}(\lambda') | \mathcal{D}))$$

Entropy Search: Formal Definition

- The p_{min} distribution characterizes the location of λ^* :

$$p_{min}(\lambda^* | \mathcal{D}) = p(\lambda^* \in \arg \min_{\lambda' \in \Lambda} (\hat{c}(\lambda') | \mathcal{D}))$$

- Our uncertainty about the location of λ^* is captured by the entropy $H(p_{min}(\cdot | \mathcal{D}))$ of the p_{min} distribution

Entropy Search: Formal Definition

- The p_{min} distribution characterizes the location of λ^* :

$$p_{min}(\lambda^* | \mathcal{D}) = p(\lambda^* \in \arg \min_{\lambda' \in \Lambda} (\hat{c}(\lambda') | \mathcal{D}))$$

- Our uncertainty about the location of λ^* is captured by the entropy $H(p_{min}(\cdot | \mathcal{D}))$ of the p_{min} distribution
- Entropy search aims to minimize $H(p_{min})$, to yield a peaked p_{min} distribution:

$$u_{ES}(\lambda) = H(p_{min}(\cdot | \mathcal{D})) - \mathbb{E}_{\tilde{c} \sim \hat{c}(\lambda)^{(t)}} H(p_{min}(\cdot | \mathcal{D} \cup \{\langle \lambda, \tilde{c} \rangle\}))$$

Choose $\lambda^{(t)} = \arg \max_{\lambda \in \Lambda} (u_{ES}^{(t)}(\lambda))$

Entropy Search: Pseudocode for Monte Carlo Approximation

$$u_{ES}(\boldsymbol{\lambda}) = \text{const} - \mathbb{E}_{\tilde{c} \sim \hat{c}(\boldsymbol{\lambda})^{(t)}} H(p_{\min}(\cdot | \mathcal{D} \cup \{\langle \boldsymbol{\lambda}, \tilde{c} \rangle\}))$$

Sampling Based Entropy Search Acquisition Function

Require : Surrogate \hat{c} , candidate configuration $\boldsymbol{\lambda}$, finite set of representer points Λ_r , dataset \mathcal{D}

Result : Utility $u(\boldsymbol{\lambda})$

```
1 for  $s = 1$  to  $S$  do
2   Sample  $\tilde{c}_s \sim \hat{c}(\boldsymbol{\lambda})$ ;  $\hat{c}_s \leftarrow$  Update  $\hat{c}$  with  $\{\langle \boldsymbol{\lambda}, \tilde{c}_s \rangle\}$ 
3   Initialize  $F[\boldsymbol{\lambda}] = 0 \quad \forall \boldsymbol{\lambda}' \in \Lambda_r$ 
4   for  $n = 1$  to  $N$  do
5     Sample  $g_n \sim \hat{c}_s$ 
6      $\boldsymbol{\lambda}_s \leftarrow \arg \min_{\boldsymbol{\lambda}' \in \Lambda_r} g_n$ 
7      $F[\boldsymbol{\lambda}_s] \leftarrow F[\boldsymbol{\lambda}_s] + 1$ 
8    $p_{\min,s}(\boldsymbol{\lambda}') \leftarrow F_{\boldsymbol{\lambda}'} / N \quad \forall \boldsymbol{\lambda}' \in \Lambda_r$ 
9    $H_s \leftarrow H(p_{\min,s})$ , computed as  $- \sum_{\boldsymbol{\lambda}' \in \Lambda_r} p_{\min,s}(\boldsymbol{\lambda}') \log p_{\min,s}(\boldsymbol{\lambda}')$ 
10   $u \leftarrow \text{const} - \frac{1}{S} \sum_{s=1}^S H_s$ 
```

Entropy Search: Variations

- The sample-based approximation is slow; for a faster approximation with expectation propagation see the original ES paper [Hennig et al. 2012]
- **Predictive Entropy Search** [Hernández-Lobato et al. 2014] is a frequently-used equivalent formulation that gives rise to more convenient approximations
- **Max-Value Entropy Search** [Wang and Jegelka 2017] is a recent variant that is cheaper to compute and has similar behavior
- Further reading and summary for ES: [Metzen 2016]

Questions to Answer for Yourself / Discuss with Friends

- Repetition. Describe the similarities and differences between KG and EI.
- Discussion. When is there an incentive for entropy search to sample at $\max(p_{min})$?

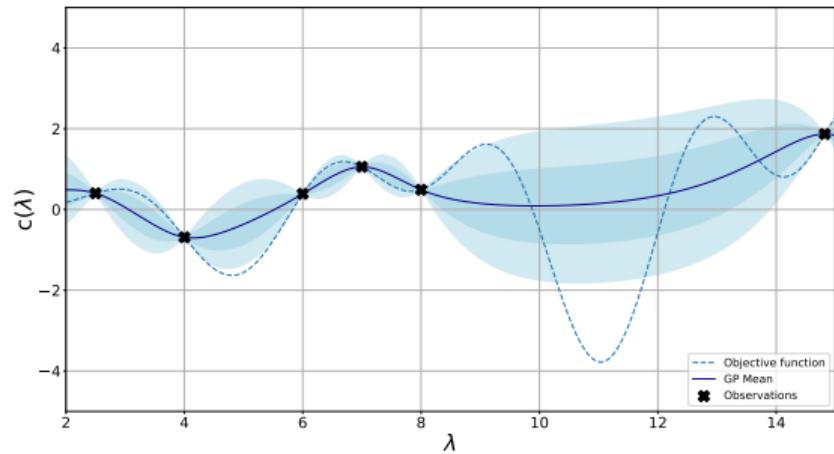
AutoML: Bayesian Optimization for HPO Surrogate Models

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Desiderata for Surrogate Models in Bayesian Optimization

In all cases

- Regression model with uncertainty estimates
- Accurate predictions



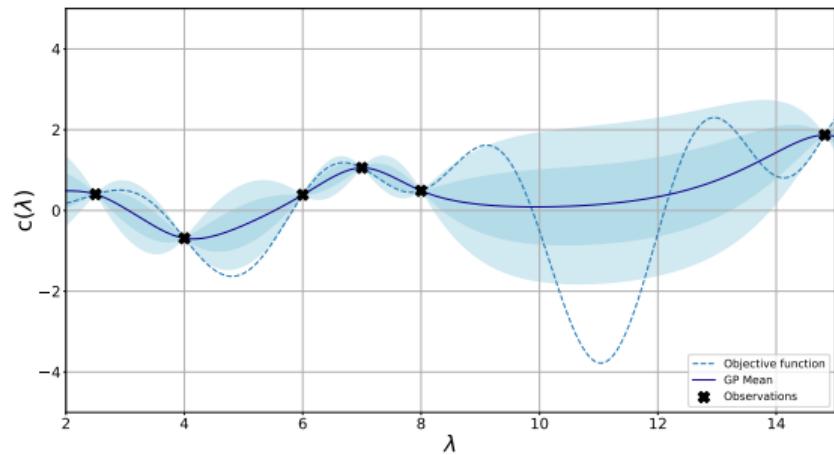
Desiderata for Surrogate Models in Bayesian Optimization

In all cases

- Regression model with uncertainty estimates
- Accurate predictions

Depending on the application

- Is cheap to train
- Scales well in the number of data points
- Scales well in the number of dimensions
- Can handle different types of inputs (categorical and continuous)



Overview of the Surrogate Models We'll Discuss

- Gaussian Processes
- Random Forests
- Bayesian Neural Networks

Gaussian Processes (GPs): Reminder of Pros and Cons

Advantages

- Smooth and reliable uncertainty estimates
- Strong sample efficiency
- We can encode expert knowledge about the design space in the kernel

Gaussian Processes (GPs): Reminder of Pros and Cons

Advantages

- Smooth and reliable uncertainty estimates
- Strong sample efficiency
- We can encode expert knowledge about the design space in the kernel

These advantages make GPs the most commonly-used model in Bayesian optimization

Gaussian Processes (GPs): Reminder of Pros and Cons

Advantages

- Smooth and reliable uncertainty estimates
- Strong sample efficiency
- We can encode expert knowledge about the design space in the kernel

These advantages make GPs the most commonly-used model in Bayesian optimization

Disadvantages

- Performance can be quite sensitive to the choice of kernel
- Cost scales cubically with the number of observations
- Weak performance for high dimensionality
- Not easily applicable in discrete or conditional spaces
- Sensitive to its own hyperparameters

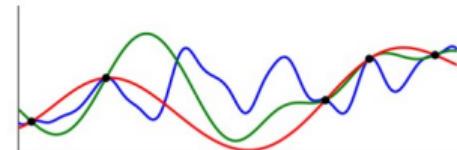
Gaussian Processes (GPs): Kernel Hyperparameters

- We could optimize GP hyperparameters (maximum likelihood, MLE, or maximum a posteriori, MAP)
- But [sampling](#) GP hyperparameters from the posterior distribution performs better; e.g., via [Markov-Chain Monte-Carlo \(MCMC\)](#)

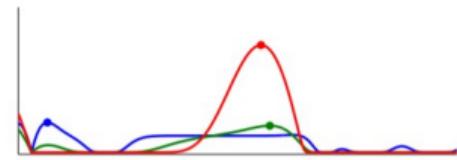
Gaussian Processes (GPs): Kernel Hyperparameters

- We could optimize GP hyperparameters (maximum likelihood, MLE, or maximum a posteriori, MAP)
- But **sampling** GP hyperparameters from the posterior distribution performs better; e.g., via **Markov-Chain Monte-Carlo (MCMC)**
- Marginalize over GP hyperparameters θ and compute an **integrated acquisition function**:

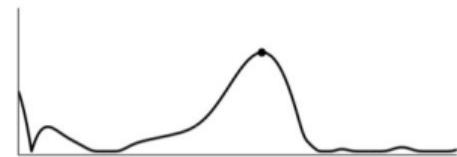
$$\bar{u}(\lambda) = \int u(\lambda, \hat{c}_\theta) p(\theta) d\theta$$



(a) Posterior samples under varying hyperparameters



(b) Expected improvement under varying hyperparameters



(c) Integrated expected improvement

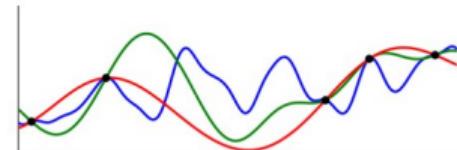
Image source: [Snoek et al. 2015]

Gaussian Processes (GPs): Kernel Hyperparameters

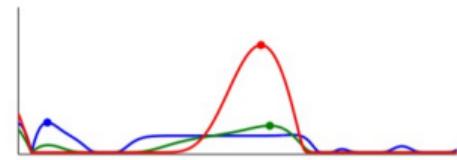
- We could optimize GP hyperparameters (maximum likelihood, MLE, or maximum a posteriori, MAP)
- But **sampling** GP hyperparameters from the posterior distribution performs better; e.g., via **Markov-Chain Monte-Carlo (MCMC)**
- Marginalize over GP hyperparameters θ and compute an **integrated acquisition function**:

$$\bar{u}(\lambda) = \int u(\lambda, \hat{c}_\theta) p(\theta) d\theta$$

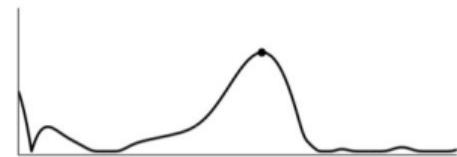
- Downside: computational expense
 - ▶ MCMC is computationally expensive
 - ▶ Acquisition function now has to be calculated for each sample



(a) Posterior samples under varying hyperparameters



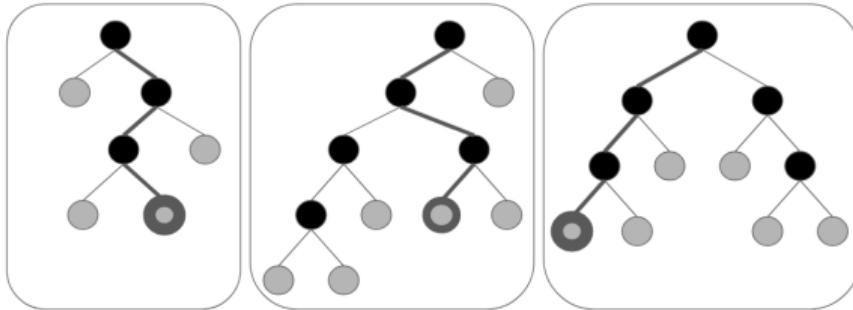
(b) Expected improvement under varying hyperparameters



(c) Integrated expected improvement

Image source: [Snoek et al. 2015]

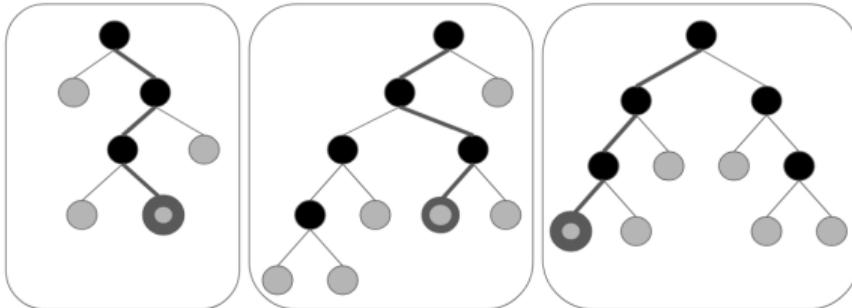
Random Forests (RFs): Reminder & How To Compute Uncertainties



RF Training

- Fit a set of randomized regression trees
- Randomization via bootstrapping & random selection of split variables / split points
- Each tree yields a possible explanation for the observations

Random Forests (RFs): Reminder & How To Compute Uncertainties



RF Training

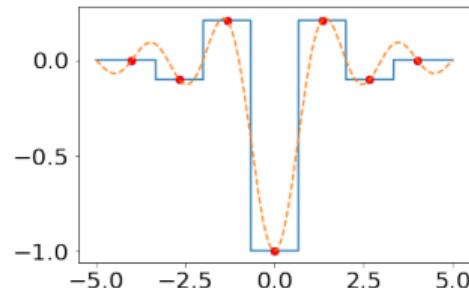
- Fit a set of randomized regression trees
- Randomization via bootstrapping & random selection of split variables / split points
- Each tree yields a possible explanation for the observations

RF Prediction

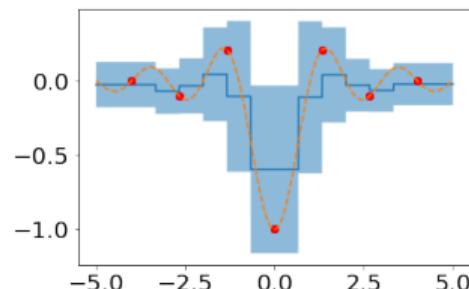
- Predict with each tree
- Aggregate predictions (e.g., average)
- Uncertainty estimate:
empirical variance across tree predictions

Random Forests (RFs): Impact of Basic Model Choices

(a) no bootstrapping,
no random splits

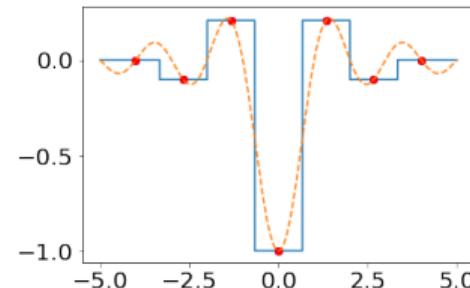


(b) with bootstrapping,
no random splits

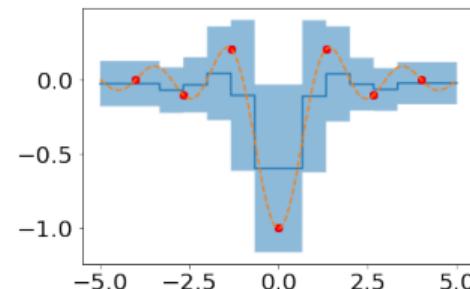


Random Forests (RFs): Impact of Basic Model Choices

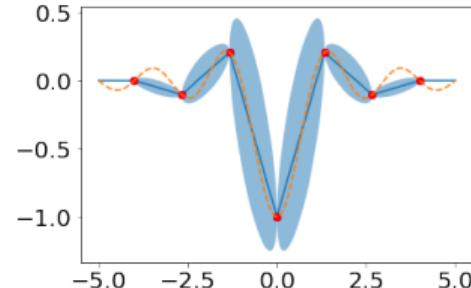
(a) no bootstrapping,
no random splits



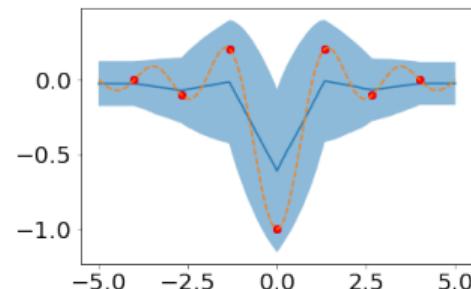
(b) with bootstrapping,
no random splits



(c) no bootstrapping,
with random splits



(d) with bootstrapping,
with random splits



Random Forests (RFs): Overview of Pros and Cons

Advantages

- Cheap to train
- Scales well with #observations n :
 - ▶ Fitting: $O(n \log n)$
 - ▶ Prediction: $O(\log n)$
- Scales well with #dimensions
- Training can be parallelized
- Can easily handle conditional, categorical, continuous and discrete spaces
- Quite robust against its own hyperparameters

Random Forests (RFs): Overview of Pros and Cons

Advantages

- Cheap to train
- Scales well with #observations n :
 - ▶ Fitting: $O(n \log n)$
 - ▶ Prediction: $O(\log n)$
- Scales well with #dimensions
- Training can be parallelized
- Can easily handle conditional, categorical, continuous and discrete spaces
- Quite robust against its own hyperparameters

Disadvantages

- Poor uncertainty estimates
- Poor extrapolation (constant)
- Priors cannot be incorporated easily

Random Forests (RFs): Overview of Pros and Cons

Advantages

- Cheap to train
- Scales well with #observations n :
 - ▶ Fitting: $O(n \log n)$
 - ▶ Prediction: $O(\log n)$
- Scales well with #dimensions
- Training can be parallelized
- Can easily handle conditional, categorical, continuous and discrete spaces
- Quite robust against its own hyperparameters

Disadvantages

- Poor uncertainty estimates
- Poor extrapolation (constant)
- Priors cannot be incorporated easily

These qualities make RFs a **robust** option for Bayesian optimization in **high dimensions**, for **categorical spaces**, or when function evaluations are quite fast

Bayesian Neural Networks: Overview

- Neural networks are more flexible & scalable than Gaussian processes
- But for use in Bayesian optimization, neural networks need to be made probabilistic

Bayesian Neural Networks: Overview

- Neural networks are more flexible & scalable than Gaussian processes
- But for use in Bayesian optimization, neural networks need to be made probabilistic
- Bayesian deep learning aims to deal with all sources of uncertainty

Bayesian Neural Networks: Overview

- Neural networks are more flexible & scalable than Gaussian processes
- But for use in Bayesian optimization, neural networks need to be made probabilistic
- Bayesian deep learning aims to deal with all sources of uncertainty
 - ▶ E.g., we don't have a single weight vector anymore, but a distribution over weights

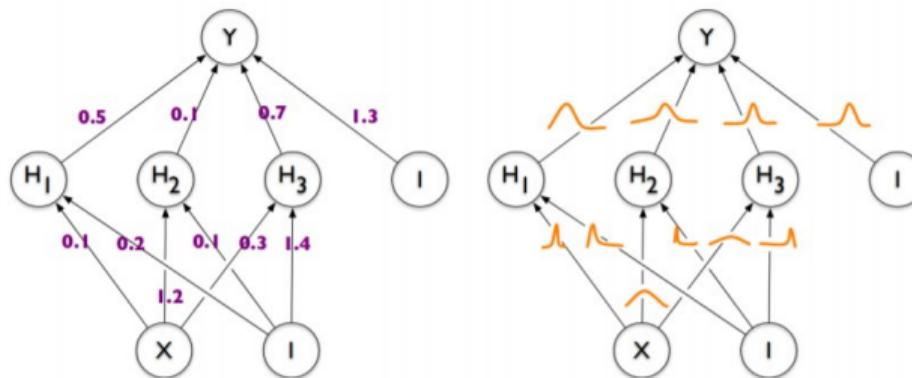


Image source: [Blundell et al. 2015]

Simplest Way of Incorporating Uncertainty in Neural Networks: DNGO

- Fit a standard regression neural network to the data (with a linear output layer)
- Use the representation in the last hidden layer as **basis functions** $\phi(x)$ of the input x
- Use Bayesian linear regression with these basis functions

Simplest Way of Incorporating Uncertainty in Neural Networks: DNGO

- Fit a standard regression neural network to the data (with a linear output layer)
- Use the representation in the last hidden layer as **basis functions** $\phi(x)$ of the input x
- Use **Bayesian linear regression with these basis functions**
 - ▶ The last layer is linear in its parameters θ
 - ▶ Therefore, the Bayesian linear regression formulas work directly
 - ▶ Feasible in closed form, in time $O(Nd^3)$, where N is the number of data points and d is the number of hidden units in the last layer
- Not fully Bayesian yet, but already allows scalable Bayesian optimization [Snoek et al. 2015]

Bayesian Optimization with BNNs: Overview of Existing Approaches

- Scalable Bayesian Optimization Using Deep Neural Networks (DNGO) [Snoek et al. 2015]
- Bayesian Optimization with Robust Bayesian Neural Networks [Springenberg et al. 2016]
- Parallel and Distributed Thompson Sampling for Large-scale Accelerated Exploration of Chemical Space [Hernández-Lobato et al. 2017]

Bayesian Optimization with BNNs: Overview of Existing Approaches

- Scalable Bayesian Optimization Using Deep Neural Networks (DNGO) [Snoek et al. 2015]
- Bayesian Optimization with Robust Bayesian Neural Networks [Springenberg et al. 2016]
- Parallel and Distributed Thompson Sampling for Large-scale Accelerated Exploration of Chemical Space [Hernández-Lobato et al. 2017]
- Hyperparameter Optimization with Factorized Multilayer Perceptrons [Schilling et al. 2015]
- Scalable Hyperparameter Transfer Learning [Perrone et al. 2018]

Bayesian Neural Networks (BNNs): Overview of Pros and Cons

Advantages

- Scales linearly with #observations
- Can obtain nice and smooth uncertainty estimates
- Flexibility: handling of categorical, continuous and discrete spaces

Bayesian Neural Networks (BNNs): Overview of Pros and Cons

Advantages

- Scales linearly with #observations
- Can obtain nice and smooth uncertainty estimates
- Flexibility: handling of categorical, continuous and discrete spaces

Disadvantages

- Usually needs more data than Gaussian processes
- Uncertainty estimates often worse than for Gaussian processes
- Many meta-design decisions
- No robust off-the-shelf implementation

Bayesian Neural Networks (BNNs): Overview of Pros and Cons

Advantages

- Scales linearly with #observations
- Can obtain nice and smooth uncertainty estimates
- Flexibility: handling of categorical, continuous and discrete spaces

Disadvantages

- Usually needs more data than Gaussian processes
- Uncertainty estimates often worse than for Gaussian processes
- Many meta-design decisions
- No robust off-the-shelf implementation

These qualities make BNNs an
ever-more promising alternative

Bayesian Neural Networks (BNNs): Further Reading

There is a lot more work on BNNs that hasn't been applied to Bayesian optimization yet:

- Ensembles obtained simply by running SGD several times [Lakshminarayanan et al. 2016]
- Dropout [Gal and Ghahramani. 2015]
- Monte Carlo Batch Normalization [Teye et al. 2018]
- Snapshot Ensembles [Gao Huang et al. 2017]

Questions to Answer for Yourself / Discuss with Friends

- **Discussion.** For which optimization problems would you rather use a RF than a GP? When would you use a BNN?
- **Discussion.** Why can DNGO's Bayesian Linear Regression approach only be applied to the last layer of a Deep Neural Network, not to all layers?
- **Open Question.** All of the surrogate models we saw have pros and cons. Would it be possible to select the best model (and its hyperparameters) dependent on the data at hand, and could this be done effectively? (This is a possible research project.)

AutoML: Bayesian Optimization for HPO

Extensions of Bayesian Optimization

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Beyond the Standard Bayesian Optimization Setting

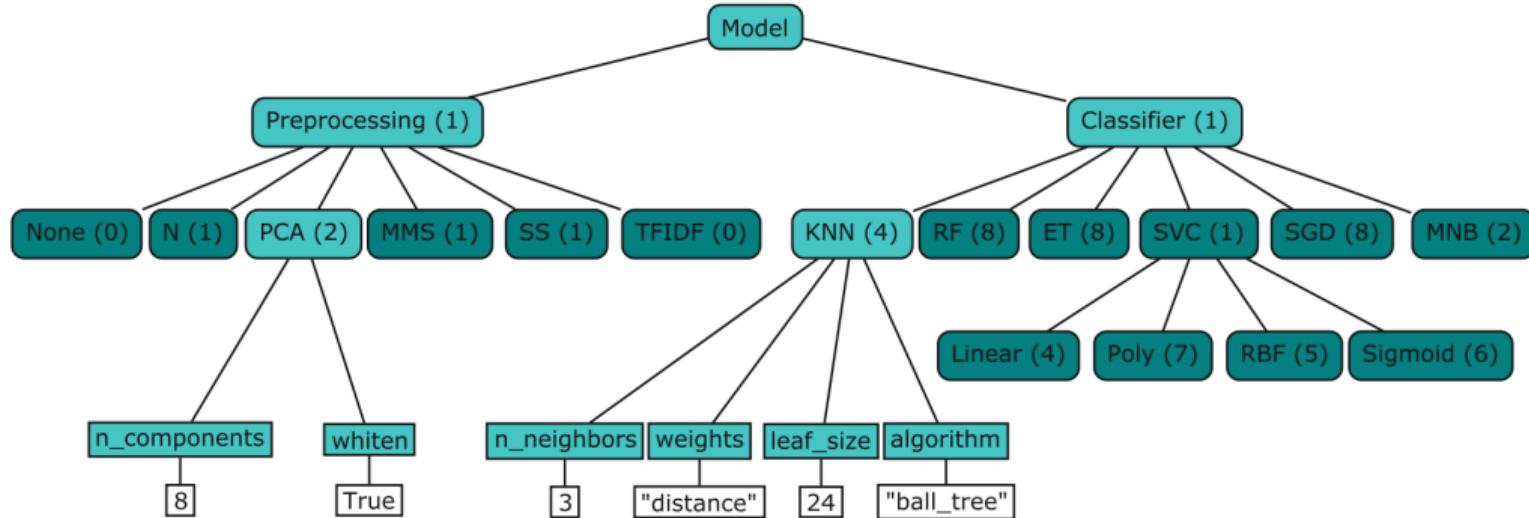
Standard Bayesian optimization problems

- Low-dimensional functions
- Continuous, smooth functions
- Sequential optimization

Extensions

- Structured search spaces: categorical & conditional hyperparameters
- High dimensions
- Parallel evaluations
- Optimization with constraints

Structured Search Spaces: Categorical & Conditional Hyperparameters

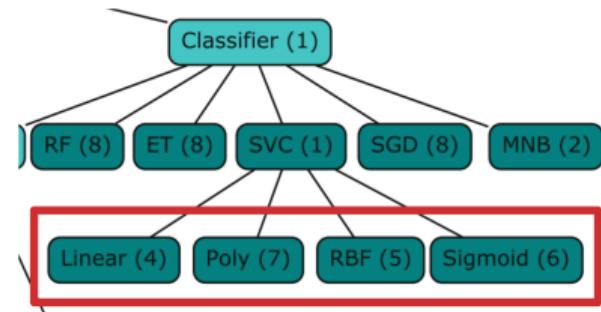


Example of a structured search space [Hutter et al. 2019]

Structured Search Spaces: Categorical Hyperparameters

Properties of categorical hyperparameters:

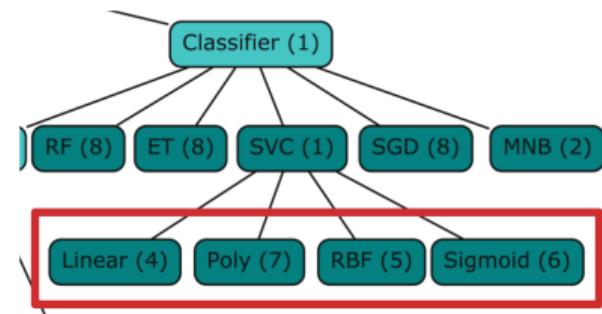
- Finite, discrete set of values
- No natural order between values
- Potentially different distances between values



Structured Search Spaces: Categorical Hyperparameters

Properties of categorical hyperparameters:

- Finite, discrete set of values
- No natural order between values
- Potentially different distances between values



This has to be taken into account by the surrogate model:

- Random Forests natively handle categorical inputs [Hutter et al, 2011]
- One-hot encoding provides a simple general solution
- Gaussian Processes can use a (weighted) Hamming Distance Kernel [Hutter. 2009]:

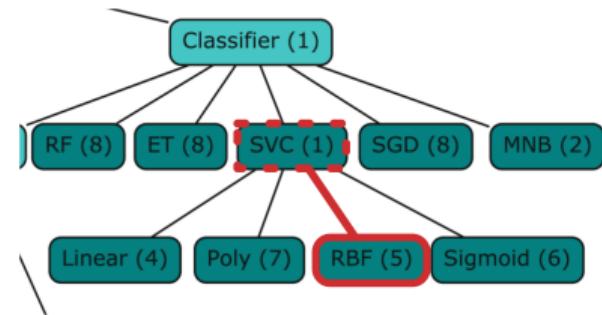
$$\kappa_{\theta}(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_j) = \exp \sum_{l=1}^d (-\theta \cdot \delta(\lambda_{i,l} \neq \lambda_{j,l}))$$

- Neural networks can learn entity embeddings for categorical inputs [Guc and Berkhahn. 2016]

Structured Search Spaces: Conditional Hyperparameters

Conditional hyperparameters:

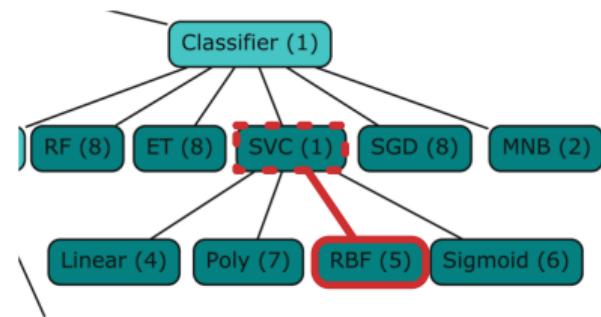
- Are **only relevant if** certain other hyperparameters take on certain values
- Should be ignored by the model **if not active**



Structured Search Spaces: Conditional Hyperparameters

Conditional hyperparameters:

- Are **only relevant if** certain other hyperparameters take on certain values
- Should be ignored by the model **if not active**



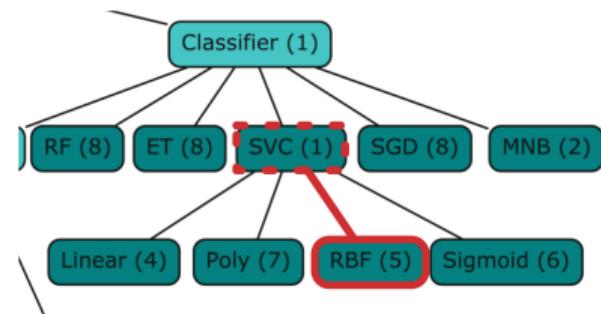
Modelling conditional hyperparameters:

- Setting the values for inactive hyperparameter to a specific value (e.g. 0)
- Random Forests [Hutter et al. 2011] and Tree Parzen Estimators [Bergstra et al. 2011] can **natively** handle conditional inputs
- There exist **several kernels for Gaussian Processes** to handle conditional inputs
[Hutter and Osborne. 2013; Lévesque et al. 2017; Jenatton et al. 2017]

Structured Search Spaces: Conditional Hyperparameters

Conditional hyperparameters:

- Are **only relevant if** certain other hyperparameters take on certain values
- Should be ignored by the model **if not active**



Modelling conditional hyperparameters:

- Setting the values for inactive hyperparameter to a specific value (e.g. 0)
- Random Forests [Hutter et al. 2011] and Tree Parzen Estimators [Bergstra et al. 2011] can **natively** handle conditional inputs
- There exist **several kernels for Gaussian Processes** to handle conditional inputs
[Hutter and Osborne. 2013; Lévesque et al. 2017; Jenatton et al. 2017]

Overall, structured search spaces are **still an active research topic** and far from solved

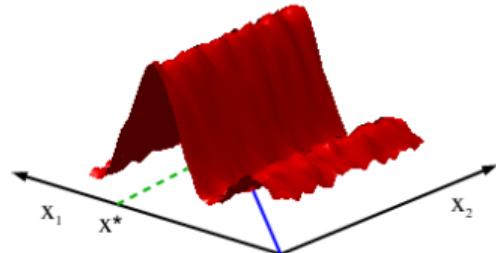
High Dimensions

- Issues

- ▶ Standard Gaussian processes do not tend to fit well in high dimensions
- ▶ Maximizing the acquisition function is computationally challenging

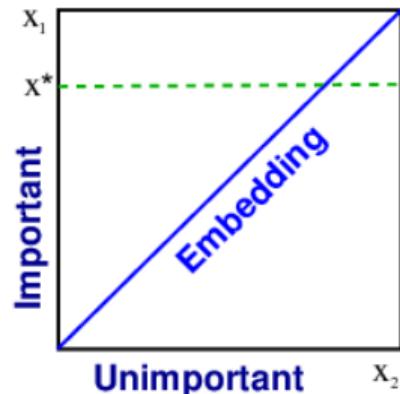
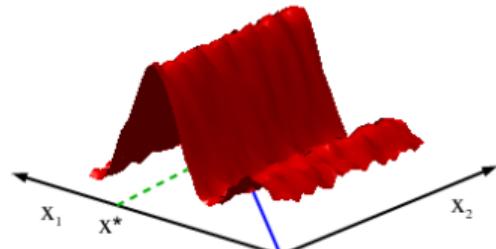
High Dimensions

- Issues
 - ▶ Standard Gaussian processes do not tend to fit well in high dimensions
 - ▶ Maximizing the acquisition function is computationally challenging
- There is still hope
 - ▶ Many optimization problems have **low effective dimensionality**
 - ▶ Not all dimensions interact with each other



High Dimensions

- Issues
 - ▶ Standard Gaussian processes do not tend to fit well in high dimensions
 - ▶ Maximizing the acquisition function is computationally challenging
- There is still hope
 - ▶ Many optimization problems have **low effective dimensionality**
 - ▶ Not all dimensions interact with each other
- Possible solutions
 - ▶ Optimize in a lower-dimensional embedding [Wang et al. 2016]
 - ▶ Fit additive models on subsets of dimensions [Kandasamy et al. 2015]
 - ▶ Use other models; e.g., random forests



Parallel Bayesian Optimization: Multi-point Acquisition Functions

- Often, we have many parallel compute units
- How should these be exploited in (the typically inherently sequential) Bayesian optimization?

Parallel Bayesian Optimization: Multi-point Acquisition Functions

- Often, we have many parallel compute units
- How should these be exploited in (the typically inherently sequential) Bayesian optimization?
- To select a batch of q points in parallel, we need to compute the multi-point acquisition function. E.g., for expected improvement:

$$q\text{-EI}(\boldsymbol{\lambda}_{1,\dots,q}) = \mathbb{E} \left[c(\hat{\boldsymbol{\lambda}}^{(t)}) - \min_{i=1,\dots,q} \hat{c}(\boldsymbol{\lambda}_i) \right]$$

Parallel Bayesian Optimization: Multi-point Acquisition Functions

- Often, we have many parallel compute units
- How should these be exploited in (the typically inherently sequential) Bayesian optimization?
- To select a batch of q points in parallel, we need to compute the multi-point acquisition function. E.g., for expected improvement:

$$q\text{-EI}(\boldsymbol{\lambda}_{1,\dots,q}) = \mathbb{E} \left[c(\hat{\boldsymbol{\lambda}}^{(t)}) - \min_{i=1,\dots,q} \hat{c}(\boldsymbol{\lambda}_i) \right]$$

- For EI and KG, this requires *expensive-to-compute* q -dimensional Gaussian cumulative distributions [Ginsbourger et al. 2008; Wu and Frazier. 2018; Wang et al. 2019]

Parallel Bayesian Optimization: Multi-point Acquisition Functions

- Often, we have many parallel compute units
- How should these be exploited in (the typically inherently sequential) Bayesian optimization?
- To select a batch of q points in parallel, we need to compute the multi-point acquisition function. E.g., for expected improvement:

$$q\text{-EI}(\boldsymbol{\lambda}_{1,\dots,q}) = \mathbb{E} \left[c(\hat{\boldsymbol{\lambda}}^{(t)}) - \min_{i=1,\dots,q} \hat{c}(\boldsymbol{\lambda}_i) \right]$$

- For EI and KG, this requires *expensive-to-compute* q -dimensional Gaussian cumulative distributions [Ginsbourger et al. 2008; Wu and Frazier. 2018; Wang et al. 2019]
- Nevertheless, multi-point acquisition functions can be optimized efficiently with gradient descent via the reparameterization trick [Wilson et al. 2018]

Asynchronous Parallel Bayesian Optimization with Pending Evaluations

- In practice, typically, not all function evaluations take the same amount of time
 - ▶ Thus, we need to select **some new points** while we're still waiting for **Pending evaluations** at other points

Asynchronous Parallel Bayesian Optimization with Pending Evaluations

- In practice, typically, not all function evaluations take the same amount of time
 - ▶ Thus, we need to select **some new points** while we're still waiting for **Pending evaluations** at other points
- Simple solution: **hallucinate observations for pending evaluations**, and use otherwise standard methods:

Asynchronous Parallel Bayesian Optimization with Pending Evaluations

- In practice, typically, not all function evaluations take the same amount of time
 - ▶ Thus, we need to select **some new points** while we're still waiting for **Pending evaluations** at other points
- Simple solution: **hallucinate observations for pending evaluations**, and use otherwise standard methods:
 - ▶ **Constant Liar**: Choose a fixed value (constant) [Ginsbourger et al. 2010]
 - ▶ **Kriging Believer**: Use the current mean prediction (belief) [Ginsbourger et al. 2010]
 - ▶ **Monte Carlo Fantasies**

Asynchronous Parallel Bayesian Optimization with Pending Evaluations

- In practice, typically, not all function evaluations take the same amount of time
 - ▶ Thus, we need to select **some new points** while we're still waiting for **Pending evaluations** at other points
- Simple solution: **hallucinate observations for pending evaluations**, and use otherwise standard methods:
 - ▶ **Constant Liar**: Choose a fixed value (constant) [Ginsbourger et al. 2010]
 - ▶ **Kriging Believer**: Use the current mean prediction (belief) [Ginsbourger et al. 2010]
 - ▶ **Monte Carlo Fantasies**
 - ★ Sample pending evaluations from the model
 - ★ Update copy of the model with these samples
 - ★ Compute acquisition function under each updated copy
 - ★ Define acquisition function as an average over these sampled acquisition functions

Bayesian Optimization with Constraints

Several types of constraints

- ① **Known constraints:**
can be accounted for when optimizing u
- ② **Hidden constraints:** no function value is observed due to
a failed function evaluation [Lee et al. 2010]
- ③ **Unknown constraints:** there's an additional, but
unknown constraint function (e.g., memory used), which
can be observed and modeled

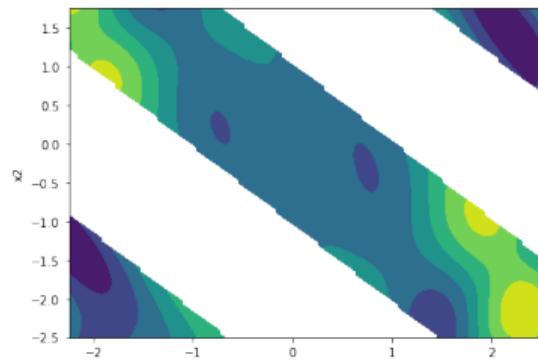


Image source: [GPFlowOpt Tutorial,
Apache 2 License]

Bayesian Optimization with Constraints

Several types of constraints

- ① **Known constraints:**
can be accounted for when optimizing u
- ② **Hidden constraints:** no function value is observed due to a failed function evaluation [Lee et al. 2010]
- ③ **Unknown constraints:** there's an additional, but unknown constraint function (e.g., memory used), which can be observed and modeled

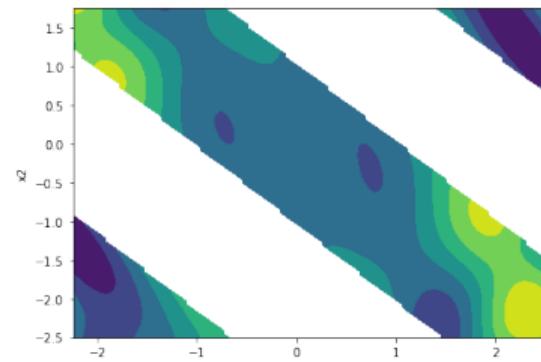


Image source: [GPFlowOpt Tutorial, Apache 2 License]

Most general solution: **Expected Constrained Improvement** [Lee et al. 2010]:

$$ECI(\lambda) = EI(\lambda)h(\lambda),$$

where $h(\lambda)$ is the probability that λ is a valid configuration.

Further literature in [Frazier. 2018] and [Feurer and Hutter 2019].

Even more extensions

Bayesian optimization has been extended to numerous scenarios:

- Multi-task, Multi-fidelity and Meta-learning → separate lecture
- Multi-objective Bayesian optimization → separate lecture
- Bayesian optimization with safety guarantees [Sui et al. 2015]
- Directly optimizing for ensemble performance [Lévesque et al. 2016]
- Combination with local search methods [Taddy et al. 2009; Eriksson et al. 2019]
- Optimization of arbitrary spaces that can be described by a kernel (e.g., neural network architectures [Kandasamy et al. 2018] or molecules [Griffiths and Hernández-Lobato. 2017])
- Many more (too many to mention)

Questions to Answer for Yourself / Discuss with Friends

- **Discussion.** What would happen if you treat a categorical hyperparameter as continuous (e.g., $\{A, B, C\}$ as $\{0, 0.5, 1\}$), in Bayesian optimization using a Gaussian Process?
- **Repetition.** Which methods can you use to impute values for outstanding evaluations? What are advantages and disadvantages of each method?
- **Discussion.** What are worst case scenarios that could happen if you ignore the noise during Bayesian optimization?

AutoML: Bayesian Optimization for HPO

The Tree-Parzen Estimator (TPE)

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Overview of TPE [Bergstra et al. 2011]

- Standard Bayesian optimization models the probability $p(y | \lambda)$ of observations y given configurations λ
- Instead, TPE fits kernel density estimators (KDEs) $l(\lambda | y \leq \gamma)$ and $g(\lambda | y > \gamma)$
 - ▶ These KDEs are for “good configurations” (leading to objective function values below a threshold γ) and “bad configurations”
 - ▶ By default, γ is set to the 15% quantile of the observations

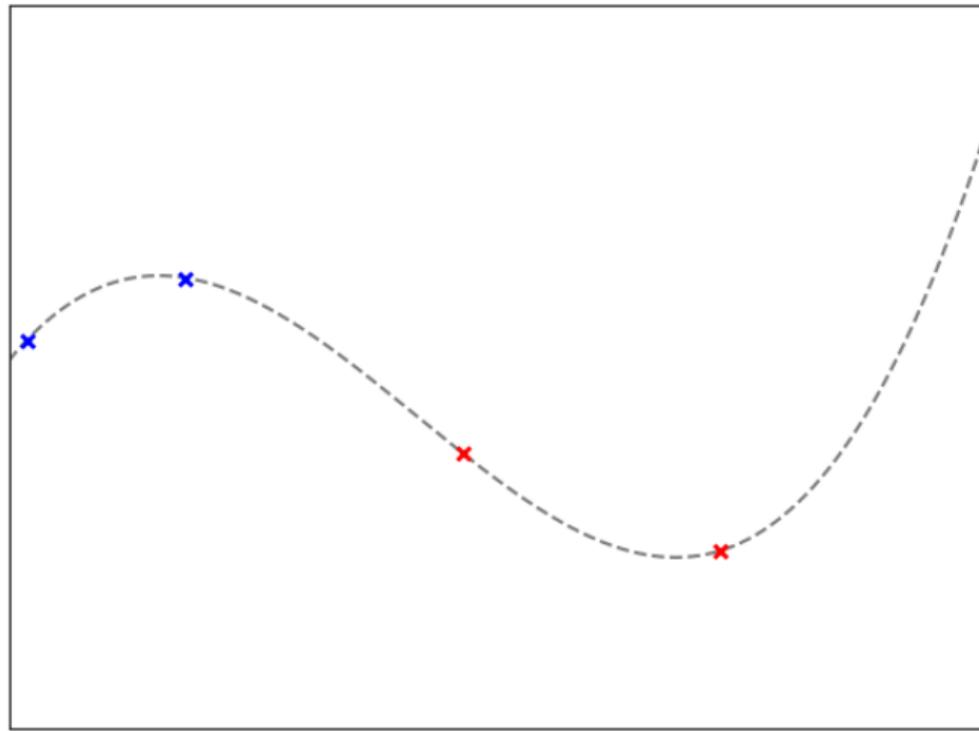
Overview of TPE [Bergstra et al. 2011]

- Standard Bayesian optimization models the probability $p(y | \lambda)$ of observations y given configurations λ
- Instead, TPE fits kernel density estimators (KDEs) $l(\lambda | y \leq \gamma)$ and $g(\lambda | y > \gamma)$
 - ▶ These KDEs are for “good configurations” (leading to objective function values below a threshold γ) and “bad configurations”
 - ▶ By default, γ is set to the 15% quantile of the observations
- Optimizing $l(\lambda)/g(\lambda)$ is equivalent to optimizing standard expected improvement in Bayesian optimization [Bergstra et al. 2011]

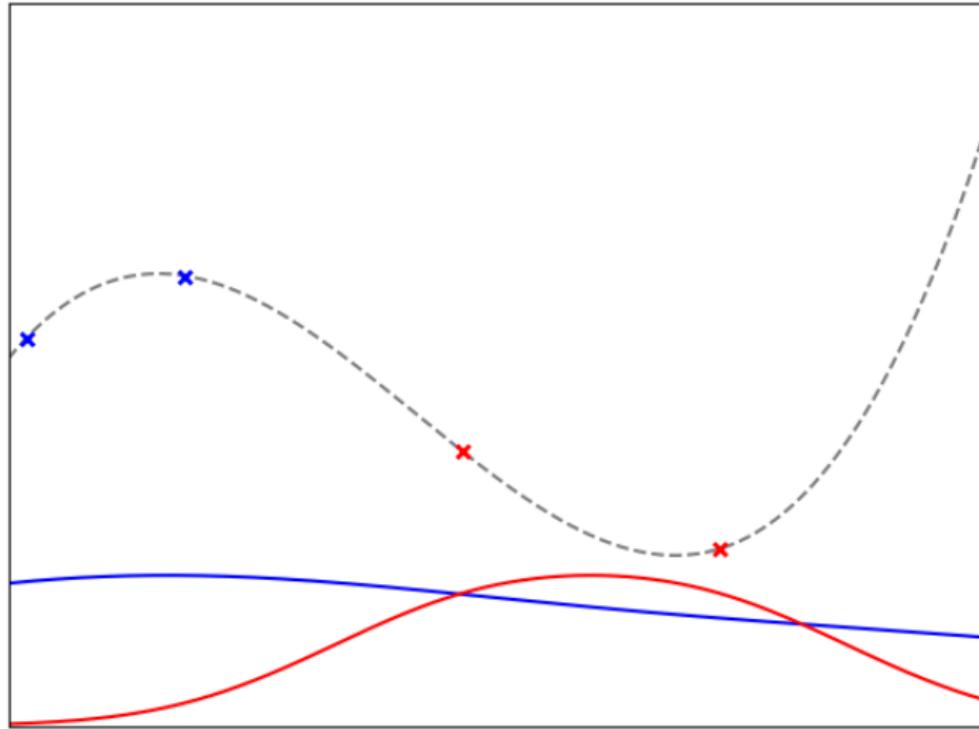
Overview of TPE [Bergstra et al. 2011]

- Standard Bayesian optimization models the probability $p(y | \lambda)$ of observations y given configurations λ
- Instead, TPE fits kernel density estimators (KDEs) $l(\lambda | y \leq \gamma)$ and $g(\lambda | y > \gamma)$
 - ▶ These KDEs are for “good configurations” (leading to objective function values below a threshold γ) and “bad configurations”
 - ▶ By default, γ is set to the 15% quantile of the observations
- Optimizing $l(\lambda)/g(\lambda)$ is equivalent to optimizing standard expected improvement in Bayesian optimization [Bergstra et al. 2011]
- Why is the technique called TPE?
 - ▶ The used KDEs are Parzen estimators
 - ▶ TPE can handle tree-structured search spaces

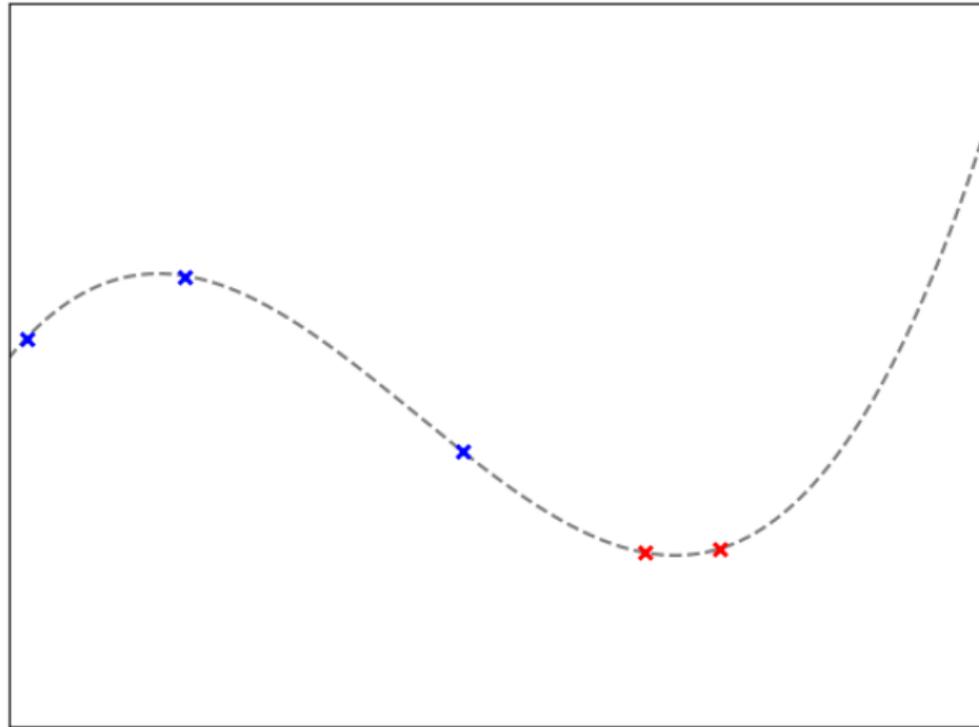
TPE Example



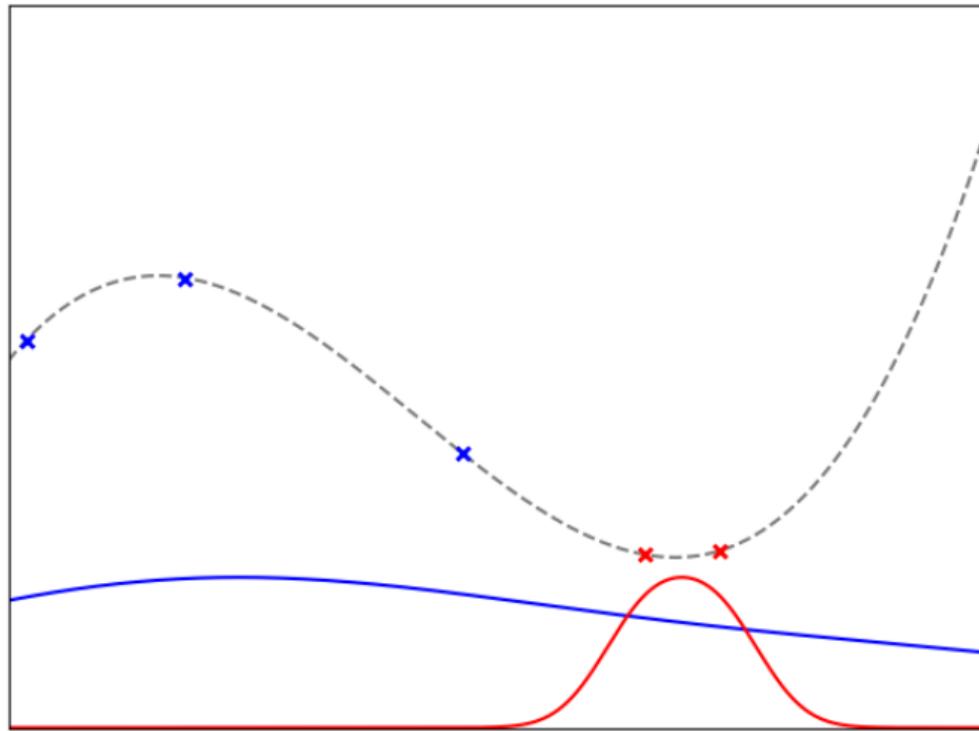
TPE Example



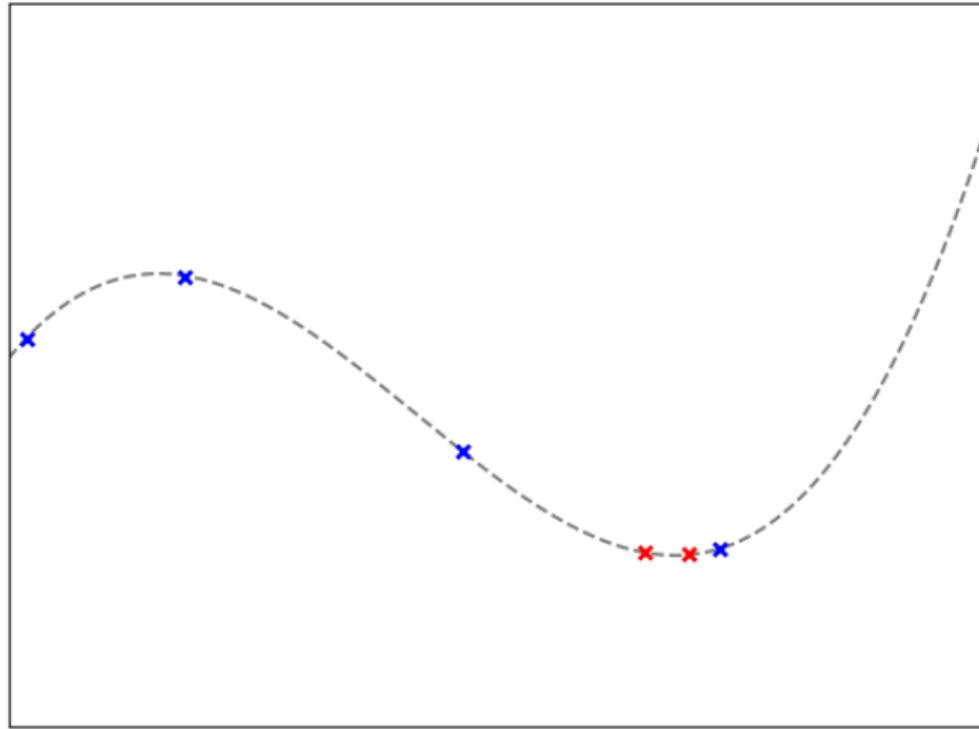
TPE Example



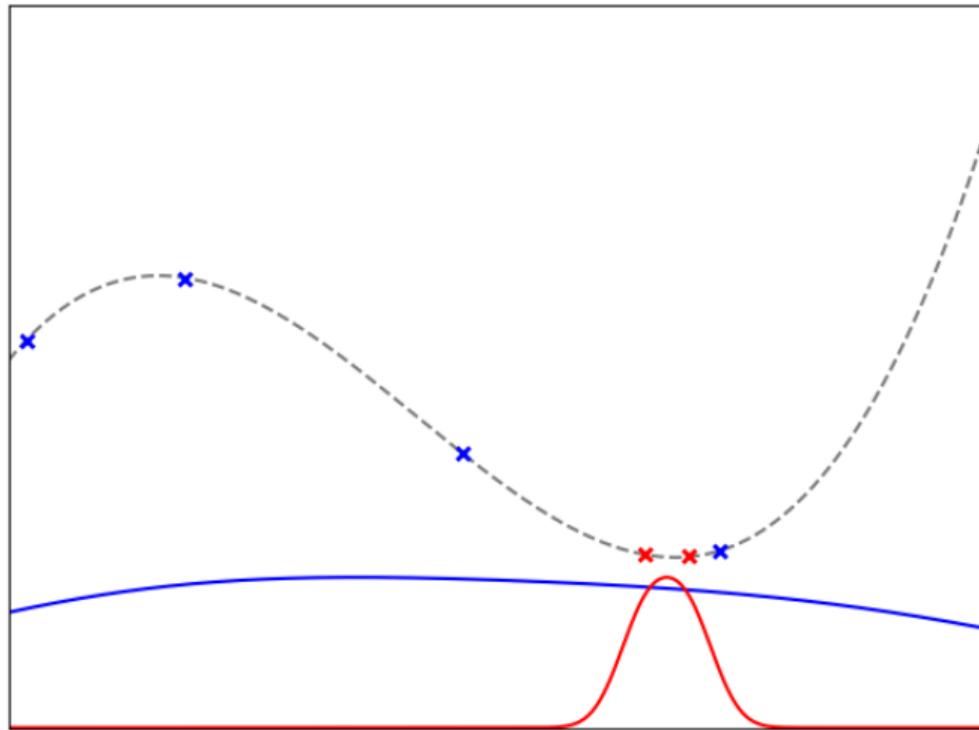
TPE Example



TPE Example



TPE Example



TPE Pseudocode

TPE loop

Require: Search space Λ , cost function c , percentile γ , maximal number of function evaluations T

Result : Best observed configuration λ according to $\mathcal{D}^{(T)}$

- 1 Initialize data $\mathcal{D}^{(0)}$ with initial observations
 - 2 **for** $t = 1$ **to** T **do**
 - 3 $\mathcal{D}_{\text{good}}, \mathcal{D}_{\text{bad}} \leftarrow$ split $\mathcal{D}^{(t-1)}$ according to quantile γ
 - 4 $l(\lambda), g(\lambda) \leftarrow$ fit KDE on $\mathcal{D}_{\text{good}}, \mathcal{D}_{\text{bad}}$ respectively
 - 5 $\Lambda_{\text{cand}} \leftarrow$ draw samples from l ;
 - 6 Select next query point: $\lambda^{(t)} \in \arg \max_{\lambda \in \Lambda_{\text{cand}}} l(\lambda)/g(\lambda)$
 - 7 Query $c(\lambda^{(t)})$
 - 8 $\mathcal{D}^{(t)} \leftarrow \mathcal{D}^{(t-1)} \cup \{\langle \lambda^{(t)}, c(\lambda^{(t)}) \rangle\}$
 - 9 **end**
-

Further Details

Remarks:

- TPE models $p(\boldsymbol{\lambda}|c(\boldsymbol{\lambda}))$
 - ▶ we can multiply it with a prior to add expert knowledge

Further Details

Remarks:

- TPE models $p(\lambda|c(\lambda))$
 - ▶ we can multiply it with a prior to add expert knowledge
- Performance of TPE depends on:
 - ▶ setting of γ to trade-off exploration and exploitation
 - ▶ bandwidth of the KDEs

Further Details

Remarks:

- TPE models $p(\lambda|c(\lambda))$
 - ▶ we can multiply it with a prior to add expert knowledge
- Performance of TPE depends on:
 - ▶ setting of γ to trade-off exploration and exploitation
 - ▶ bandwidth of the KDEs
- A successful tool implementing TPE is Hyperopt [Bergstra et al.]

Summary

Advantages

- Computationally efficient: $O(Nd)$
- Parallelizable
- Robust
- Can handle complex search spaces with priors

Summary

Advantages

- Computationally efficient: $O(Nd)$
- Parallelizable
- Robust
- Can handle complex search spaces with priors

Disadvantages

- Less sample-efficient than GPs

Questions to Answer for Yourself / Discuss with Friends

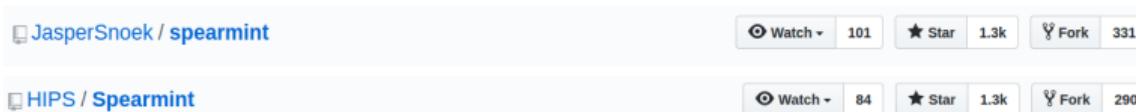
- **Discussion.** Is TPE really Bayesian optimization?
- **Discussion.** How does γ impact the optimization procedure?
- **Derivation.** Go through the derivation that optimizing $l(\lambda)/g(\lambda)$ is equivalent to optimizing expected improvement; see Section 4.1 in [Bergstra et al. 2011].

AutoML: Bayesian Optimization for HPO Success Stories

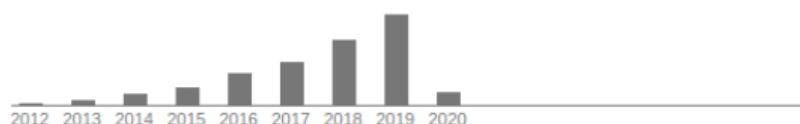
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Spearmint [Snoek et al. 2012]

- First successful open source Bayesian optimization implementation
- Implements standard Bayesian optimization with MCMC integration of the acquisition function, asynchronous parallelism, input warping and constraints
- Startup based on Spearmint got acquired by Twitter in 2015
- Still heavily used and cited and available at <https://github.com/HIPS/spearmint>:



Cited by 3073



[Practical bayesian optimization of machine learning algorithms](#)

J Snoek, H Larochelle, RP Adams - Advances in neural information processing systems, 2012

[Cited by 3073](#) [Related articles](#) [All 26 versions](#)

Hyperopt [Bergstra et al. 2011, Bergstra et al., 2013, Bergstra et al., 2013, Bergstra et al., 2015]

- Hyperopt is another successful open source Bayesian optimization package
- Implements the TPE algorithm and supports asynchronous parallel evaluations
- Maintained since 2013
- Available at <https://github.com/hyperopt/hyperopt>

 [hyperopt / hyperopt](https://github.com/hyperopt/hyperopt)

 Watch ▾ 118

 Star 4.4k

 Fork 747

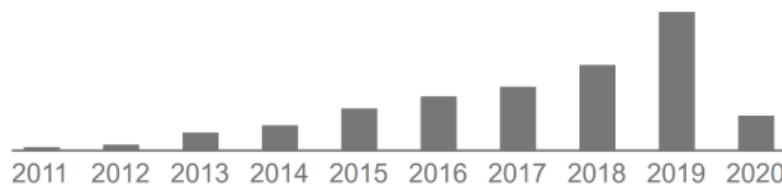
- Standard BO tool based on random forests (RFs), reflecting the strengths of RFs in terms of **scalability & flexibility**:
 - ▶ High dimensionality (low effective dimensionality)
 - ▶ Computational efficiency (\rightarrow low overhead)
 - ▶ Supports continuous/categorical/conditional parameters
 - ▶ Supports non-standard noise (non-Gaussian, heteroscedastic)
 - ▶ Usability off the shelf (robustness towards model's own hyperparameters)

- Standard BO tool based on random forests (RFs), reflecting the strengths of RFs in terms of **scalability & flexibility**:
 - ▶ High dimensionality (low effective dimensionality)
 - ▶ Computational efficiency (\rightarrow low overhead)
 - ▶ Supports continuous/categorical/conditional parameters
 - ▶ Supports non-standard noise (non-Gaussian, heteroscedastic)
 - ▶ Usability off the shelf (robustness towards model's own hyperparameters)
- SMAC also handles a more general problem: $\arg \min_{\lambda \in \Lambda} \sum_{i=1}^N c(\lambda, i)$

SMAC [Hutter et al. 2011]

- Standard BO tool based on random forests (RFs), reflecting the strengths of RFs in terms of **scalability & flexibility**:
 - ▶ High dimensionality (low effective dimensionality)
 - ▶ Computational efficiency (\rightarrow low overhead)
 - ▶ Supports continuous/categorical/conditional parameters
 - ▶ Supports non-standard noise (non-Gaussian, heteroscedastic)
 - ▶ Usability off the shelf (robustness towards model's own hyperparameters)
- SMAC also handles a more general problem: $\arg \min_{\lambda \in \Lambda} \sum_{i=1}^N c(\lambda, i)$
- Maintained since 2011, now available in version 3: <https://github.com/automl/SMAC3>

Cited by 1318



Sequential model-based optimization for general algorithm configuration
F Hutter, HH Hoos, K Leyton-Brown - International conference on
learning and intelligent ..., 2011

Tuning AlphaGo [Chen et al. 2018]

- “During the development of AlphaGo, its many hyperparameters were tuned with Bayesian optimization multiple times.”
- “This automatic tuning process resulted in substantial improvements in playing strength. For example, prior to the match with Lee Sedol, we tuned the latest AlphaGo agent and this improved its win-rate from 50% to 66.5% in self-play games. This tuned version was deployed in the final match.”
- Of course, since we tuned AlphaGo many times during its development cycle, the compounded contribution was even higher than this percentage.

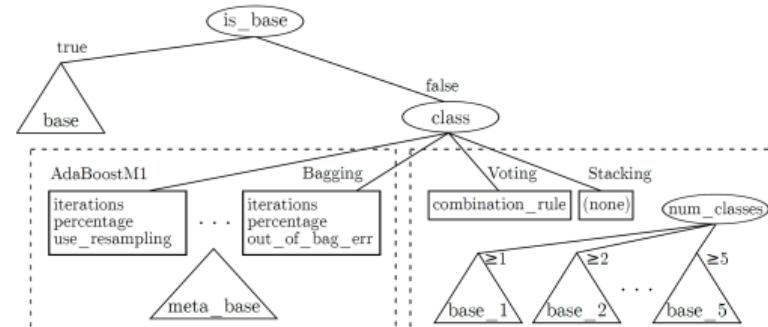
Company usage

- SIGOPT: startup offering Bayesian optimization as a service
- Facebook provides an open source Bayesian optimization package [BoTorch]
- Amazon provides an open source Bayesian optimization package [EmuKit]
- Uber tunes algorithms for *Uber Pool*, *UberX* and *Uber Eats* [source]
- Many more, but less openly

- First general AutoML system, carrying out **Combined Algorithm Selection and Hyperparameter optimization (CASH)**, jointly optimizing
 - ▶ Choice of algorithm (out of 26 classifiers)
 - ▶ The algorithm's hyperparameters (up to 10)
 - ▶ Choice of preprocessing method and its hyperparameters
 - ▶ Choice of ensemble & meta methods

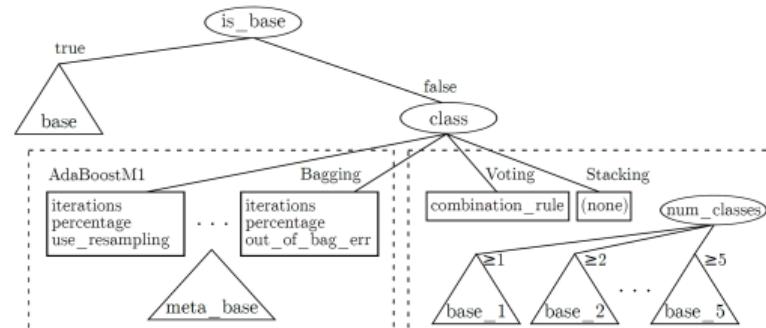
Auto-WEKA [Thornton et al, 2013, Kotthoff et al, 2017, Kotthoff et al. 2019]

- First general AutoML system, carrying out **Combined Algorithm Selection and Hyperparameter optimization (CASH)**, jointly optimizing
 - ▶ Choice of algorithm (out of 26 classifiers)
 - ▶ The algorithm's hyperparameters (up to 10)
 - ▶ Choice of preprocessing method and its hyperparameters
 - ▶ Choice of ensemble & meta methods
- Parameterized WEKA [Frank et al, 2016]: **768 hyperparameters**, 4 leves of conditionality



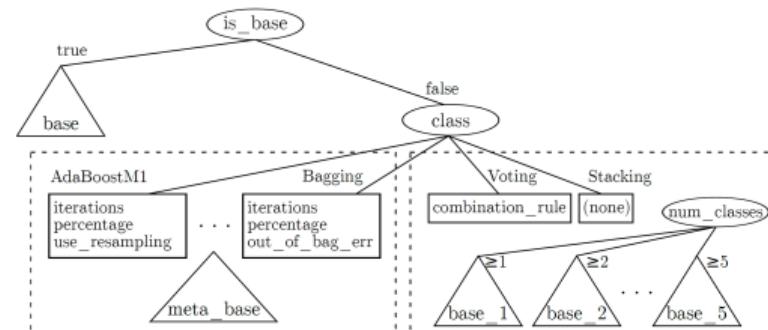
Auto-WEKA [Thornton et al, 2013, Kotthoff et al, 2017, Kotthoff et al. 2019]

- First general AutoML system, carrying out **Combined Algorithm Selection and Hyperparameter optimization (CASH)**, jointly optimizing
 - ▶ Choice of algorithm (out of 26 classifiers)
 - ▶ The algorithm's hyperparameters (up to 10)
 - ▶ Choice of preprocessing method and its hyperparameters
 - ▶ Choice of ensemble & meta methods
- Parameterized WEKA [Frank et al, 2016]: **768 hyperparameters**, 4 levels of conditionality
- Optimized 10-fold cross-validation via SMAC [Hutter et al, 2011]



Auto-WEKA [Thornton et al, 2013, Kotthoff et al, 2017, Kotthoff et al. 2019]

- First general AutoML system, carrying out **Combined Algorithm Selection and Hyperparameter optimization (CASH)**, jointly optimizing
 - ▶ Choice of algorithm (out of 26 classifiers)
 - ▶ The algorithm's hyperparameters (up to 10)
 - ▶ Choice of preprocessing method and its hyperparameters
 - ▶ Choice of ensemble & meta methods
- Parameterized WEKA [Frank et al, 2016]: **768 hyperparameters**, 4 levels of conditionality
- Optimized 10-fold cross-validation via SMAC [Hutter et al, 2011]
- Results:
 - ▶ Better than an oracle of the 26 base classifiers with default hyperparameters
 - ▶ 100× faster than grid search over base classifiers, and still better in 14/21 cases
 - ▶ Better than the only other applicable method TPE in 19/21 cases
- Impact for practitioners: Auto-WEKA plugin was downloaded tens of thousands of times



Questions to Answer for Yourself / Discuss with Friends

- Repetition. List several success stories of Bayesian optimization
- Repetition. List several prominent tools for Bayesian optimization
- Discussion. Recall the algorithm selection problem; how does CASH relate to this (after all, it also has “algorithm selection” as part of its name)? (Hint: they are quite different.)

AutoML: Bayesian Optimization for HPO

Further Reading

Bernd Bischl [Frank Hutter](#) Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Further Reading

Tutorials on Bayesian Optimization

- A tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning [Brochu et al. 2010]
- Taking the Human out of the Loop: A Review of Bayesian Optimization [Shahriari et al. 2016]
- A Tutorial on Bayesian Optimization [Frazier 2018]

Survey on hyperparameter optimization: [Feurer and Hutter 2019]

Speedup Techniques for Hyperparameter Optimization

Overview

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Beyond Black-box Optimization

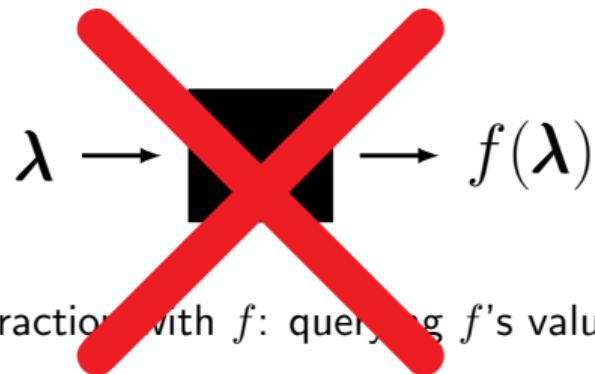
Recall general blackbox optimization:

$$\lambda \rightarrow \boxed{\quad} \rightarrow f(\lambda)$$

Only mode of interaction with f : querying f 's value at a given λ

Beyond Black-box Optimization

Recall general blackbox optimization:



Only mode of interaction with f : querying f 's value at a given λ

Too slow for tuning expensive models

Methods for Going Beyond Blackbox Bayesian Optimization

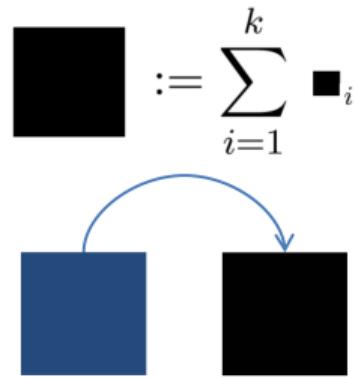
- Sum of little black boxes

- ▶ Each little black box is fast but only yields a noisy estimate
- ▶ SMAC [Hutter et al. 2011] directly solves $\arg \min_{\lambda \in \Lambda} \sum_{i=1}^N c(\lambda, i)$
- ▶ Auto-WEKA [Thornton et al, 2013] used this to optimize 10-fold cross-validation performance

$$\blacksquare := \sum_{i=1}^k \blacksquare_i$$

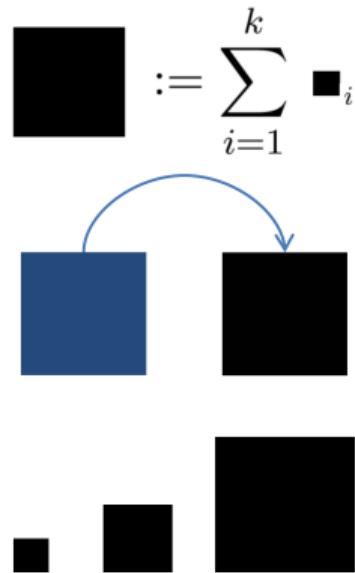
Methods for Going Beyond Blackbox Bayesian Optimization

- Sum of little black boxes
 - ▶ Each little black box is fast but only yields a noisy estimate
 - ▶ SMAC [Hutter et al. 2011] directly solves $\arg \min_{\lambda \in \Lambda} \sum_{i=1}^N c(\lambda, i)$
 - ▶ Auto-WEKA [Thornton et al, 2013] used this to optimize 10-fold cross-validation performance
- Meta-learning across problems / datasets



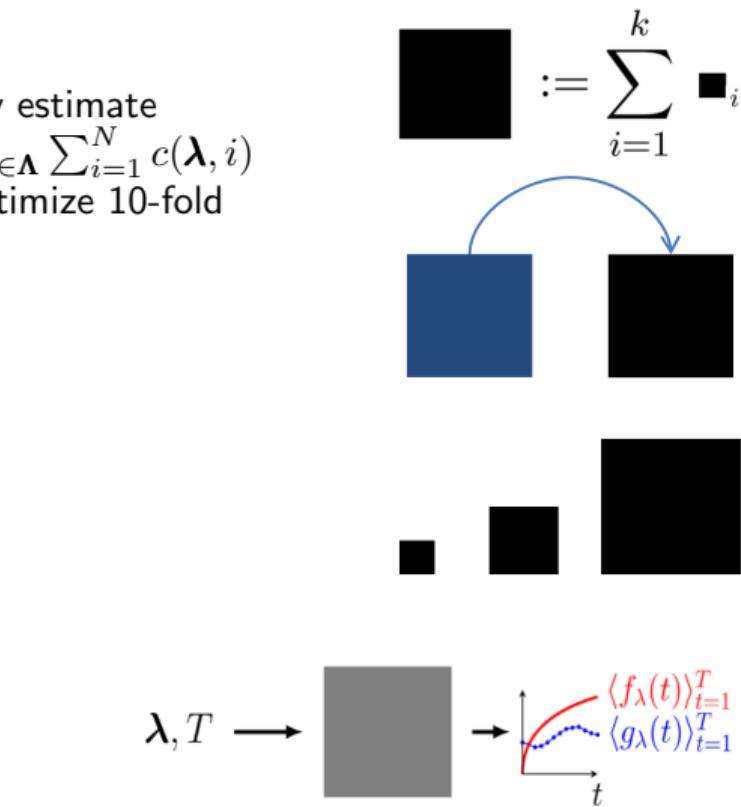
Methods for Going Beyond Blackbox Bayesian Optimization

- Sum of little black boxes
 - ▶ Each little black box is fast but only yields a noisy estimate
 - ▶ SMAC [Hutter et al. 2011] directly solves $\arg \min_{\lambda \in \Lambda} \sum_{i=1}^N c(\lambda, i)$
 - ▶ Auto-WEKA [Thornton et al, 2013] used this to optimize 10-fold cross-validation performance
- Meta-learning across problems / datasets
- Multi-fidelity optimization



Methods for Going Beyond Blackbox Bayesian Optimization

- Sum of little black boxes
 - ▶ Each little black box is fast but only yields a noisy estimate
 - ▶ SMAC [Hutter et al. 2011] directly solves $\arg \min_{\lambda \in \Lambda} \sum_{i=1}^N c(\lambda, i)$
 - ▶ Auto-WEKA [Thornton et al, 2013] used this to optimize 10-fold cross-validation performance
- Meta-learning across problems / datasets
- Multi-fidelity optimization
- Graybox optimization / learning curve prediction



Learning Goals of this Lecture

After this lecture, students can ...

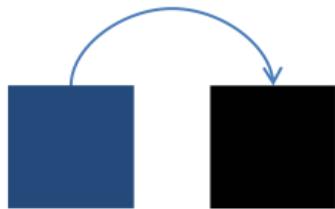
- Describe many different ways of using [meta-learning](#) to speed up HPO
- Explain the concept of [multi-fidelity](#) optimization to speed up HPO
- Explain the [Successive Halving](#) and [Hyperband](#) algorithms
- Explain how to combine Bayesian optimization and Hyperband in BOHB
- Describe how to [exploit multiple fidelities](#) in Bayesian optimization
- Discuss several ways of predicting [learning curves](#)
- Discuss [success stories](#) of speeding up Bayesian optimization

Speedup Techniques for Hyperparameter Optimization

Meta-Learning

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Introduction



- Learning essentially never stops:
 - ▶ Many models are periodically re-fit to track changes in the data
 - ▶ Many models are re-fit to perform well on new tasks
- The best hyperparameter configuration tends to remain quite stable across tasks

Introduction



- Learning essentially never stops:
 - ▶ Many models are periodically re-fit to track changes in the data
 - ▶ Many models are re-fit to perform well on new tasks
- The best hyperparameter configuration tends to remain quite stable across tasks

For a good introduction to meta-learning in general, see [AutoML Book: Chapter 2]

Problem Statement

Given:

- a set of prior tasks: $t_j \in \mathcal{T}_{\text{meta}} \subset \mathcal{T}$,
- a set of new tasks: $t_{\text{new}} \in \mathcal{T}$,

Problem Statement

Given:

- a set of prior tasks: $t_j \in \mathcal{T}_{\text{meta}} \subset \mathcal{T}$,
- a set of new tasks: $t_{\text{new}} \in \mathcal{T}$,
- a set of learning algorithms, fully defined by $\theta_i \in \Theta$

Problem Statement

Given:

- a set of prior tasks: $t_j \in \mathcal{T}_{\text{meta}} \subset \mathcal{T}$,
- a set of new tasks: $t_{\text{new}} \in \mathcal{T}$,
- a set of learning algorithms, fully defined by $\theta_i \in \Theta$
- a set of prior evaluations $\mathcal{D}_{\text{meta}}$ on $t_j \in \mathcal{T}_{\text{meta}}$
- a set of evaluations \mathcal{D}_{new} on new task t_{new}

Problem Statement

Given:

- a set of prior tasks: $t_j \in \mathcal{T}_{\text{meta}} \subset \mathcal{T}$,
- a set of new tasks: $t_{\text{new}} \in \mathcal{T}$,
- a set of learning algorithms, fully defined by $\theta_i \in \Theta$
- a set of prior evaluations $\mathcal{D}_{\text{meta}}$ on $t_j \in \mathcal{T}_{\text{meta}}$
- a set of evaluations \mathcal{D}_{new} on new task t_{new}

Goal of meta-learning:

- use meta-data $\mathcal{D}_{\text{meta}}$ to choose $\theta_i \in \Theta$ for t_{new} better than only based on \mathcal{D}_{new} .

[adapted from AutoML Book: Chapter 2]

The Role of Meta-Features

- We can often extract additional characteristics for each task, called **meta-features**
- Each task t_j can be described by a vector of K meta-features:

$$m(t_j) = (m_{j,1}, \dots, m_{j,K})$$

The Role of Meta-Features

- We can often extract additional characteristics for each task, called **meta-features**
- Each task t_j can be described by a vector of K meta-features:

$$m(t_j) = (m_{j,1}, \dots, m_{j,K})$$

- This vector can be used to define a **similarity measure** between two tasks
 - ▶ e.g., calculating the Euclidean distance between $m(t_i)$ and $m(t_j)$
 - ▶ Based on similarity, we can transfer information from the most similar tasks to new task t_{new}

Overview of Meta-Features in Machine Learning

- **Simple** - easily extracted from the data, describe the basic dataset structure
 - ▶ e.g., number of features, data points or classes

Overview of Meta-Features in Machine Learning

- **Simple** - easily extracted from the data, describe the basic dataset structure
 - ▶ e.g., number of features, data points or classes
- **Statistical** - characterize the data via descriptive statistics:
 - ▶ e.g., average or standard deviation of features, or their correlation with the labels

Overview of Meta-Features in Machine Learning

- **Simple** - easily extracted from the data, describe the basic dataset structure
 - ▶ e.g., number of features, data points or classes
- **Statistical** - characterize the data via descriptive statistics:
 - ▶ e.g., average or standard deviation of features, or their correlation with the labels
- **Information-theoretic** - measure the class entropy in the data
 - ▶ capture the amount of information in the data

Overview of Meta-Features in Machine Learning

- **Simple** - easily extracted from the data, describe the basic dataset structure
 - ▶ e.g., number of features, data points or classes
- **Statistical** - characterize the data via descriptive statistics:
 - ▶ e.g., average or standard deviation of features, or their correlation with the labels
- **Information-theoretic** - measure the class entropy in the data
 - ▶ capture the amount of information in the data
- **Model-based** - extracted from a model induced using the training data
 - ▶ these are often based on properties of decision tree models
 - ▶ e.g., number of leaves, number of nodes, shape of the tree

Overview of Meta-Features in Machine Learning

- **Simple** - easily extracted from the data, describe the basic dataset structure
 - ▶ e.g., number of features, data points or classes
- **Statistical** - characterize the data via descriptive statistics:
 - ▶ e.g., average or standard deviation of features, or their correlation with the labels
- **Information-theoretic** - measure the class entropy in the data
 - ▶ capture the amount of information in the data
- **Model-based** - extracted from a model induced using the training data
 - ▶ these are often based on properties of decision tree models
 - ▶ e.g., number of leaves, number of nodes, shape of the tree
- **Landmarking** - computed by running several fast ML algorithms on the dataset
 - ▶ e.g., is fast algorithm A better than fast algorithm B on this dataset?
 - ▶ this can capture different properties of the dataset, e.g., linear separability

Overview of Meta-Features in Machine Learning

- **Simple** - easily extracted from the data, describe the basic dataset structure
 - ▶ e.g., number of features, data points or classes
- **Statistical** - characterize the data via descriptive statistics:
 - ▶ e.g., average or standard deviation of features, or their correlation with the labels
- **Information-theoretic** - measure the class entropy in the data
 - ▶ capture the amount of information in the data
- **Model-based** - extracted from a model induced using the training data
 - ▶ these are often based on properties of decision tree models
 - ▶ e.g., number of leaves, number of nodes, shape of the tree
- **Landmarking** - computed by running several fast ML algorithms on the dataset
 - ▶ e.g., is fast algorithm A better than fast algorithm B on this dataset?
 - ▶ this can capture different properties of the dataset, e.g., linear separability
- **Others** - not included in the previous groups
 - ▶ e.g., time related measures, clustering and distance-based measures

Meta-Learning for HPO Approach 1: Warmstarting

- Experts often start HPO from a strong default (rather than random configurations)

Meta-Learning for HPO Approach 1: Warmstarting

- Experts often start HPO from a strong default (rather than random configurations)
- Can we learn from meta-data $\mathcal{D}_{\text{meta}}$ how to [initialize](#) HPO?

Meta-Learning for HPO Approach 1: Warmstarting

- Experts often start HPO from a strong default (rather than random configurations)
- Can we learn from meta-data $\mathcal{D}_{\text{meta}}$ how to [initialize](#) HPO?
- Note: just a single default configuration often does not perform great on a new dataset
 - ▶ Otherwise there would be no point in HPO

Meta-Learning for HPO Approach 2: Model-Warmstarting

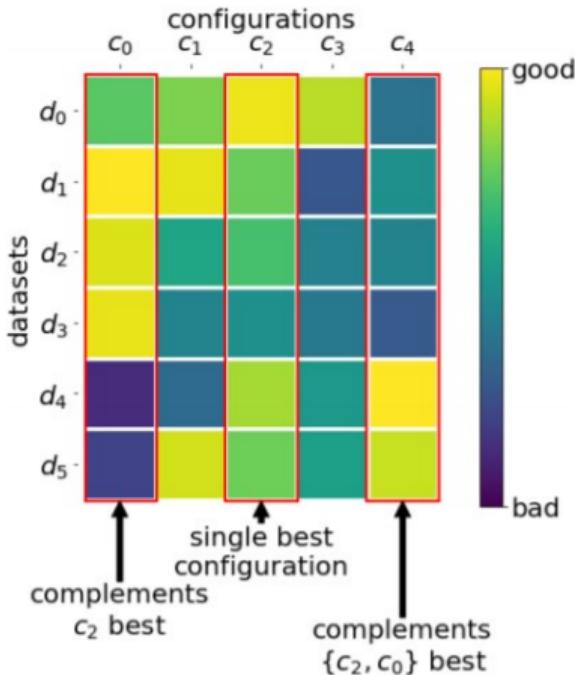
- Many HPO methods use a predictive model (e.g., Bayesian optimization)
- By running HPO on different datasets, we learn something about the search landscape
 - ▶ E.g., what are bad regions of the configuration space in general

Meta-Learning for HPO Approach 2: Model-Warmstarting

- Many HPO methods use a predictive model (e.g., Bayesian optimization)
- By running HPO on different datasets, we learn something about the search landscape
 - ▶ E.g., what are bad regions of the configuration space in general
- Given: n predictive models $\hat{c}_{\mathcal{D}_i} : \Lambda \rightarrow \mathbb{R}$ from HPO on $\mathcal{T}_{\text{meta}}$
- How can we use these $\hat{c}_{\mathcal{D}_i}$ to speed up HPO?

Meta-Learning for HPO Approach 3: Task-independent Recommendations

- *Idea:* learn a sorted list of defaults
- *Method:* mostly greedy on $\mathcal{T}_{\text{meta}}$
- *Results:* surprisingly strong,
better than Bayesian Optimization



[Wistuba et al. 2017; Feurer et al. 2018; Pfisterer et al. 2018]

Meta-Learning for HPO Approach 3: Task-independent Recommendations

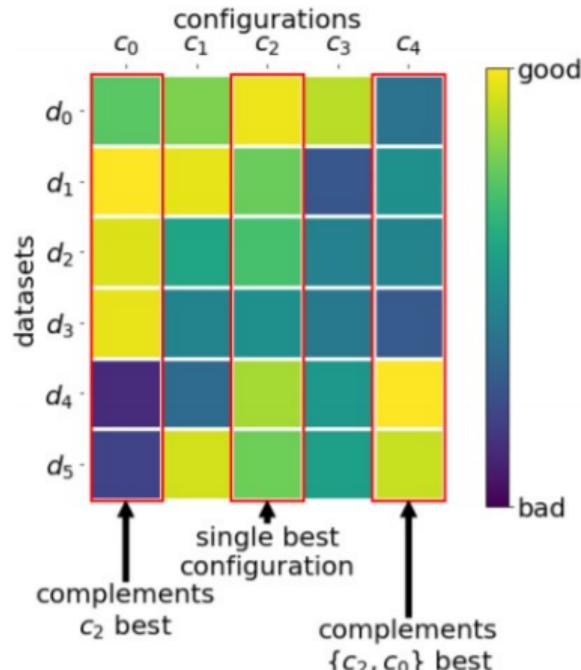
- Idea: learn a sorted list of defaults
- Method: mostly greedy on $\mathcal{T}_{\text{meta}}$
- Results: surprisingly strong,
better than Bayesian Optimization

Advantages

- Easy to share and use
- Strong anytime performance
- Embarrassingly parallel

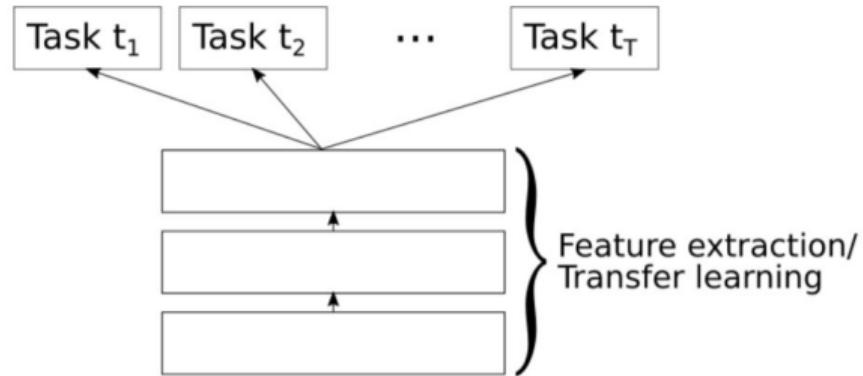
Disadvantages

- Not adaptive



Meta-Learning for HPO Approach 4: Joint model for Bayesian optimization

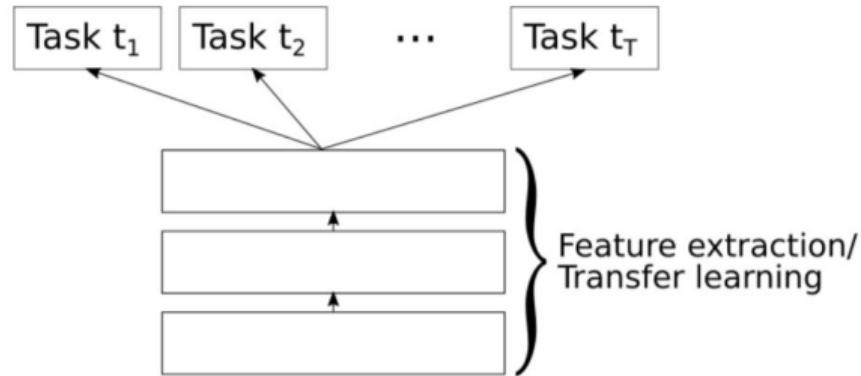
- Jointly train a “deep” neural network **on all tasks**



[Perrone et al. 2018]

Meta-Learning for HPO Approach 4: Joint model for Bayesian optimization

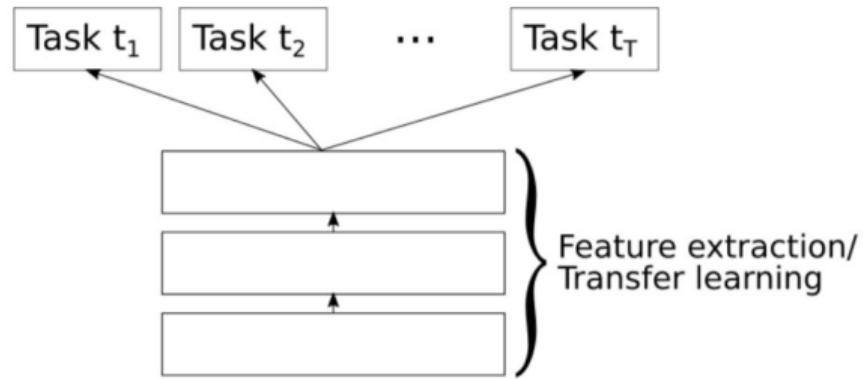
- Jointly train a “deep” neural network **on all tasks**
 - ▶ Have a separate output layer (head) for each task
 - ▶ Each head is a Bayesian linear regression (recall DNGO)



[Perrone et al. 2018]

Meta-Learning for HPO Approach 4: Joint model for Bayesian optimization

- Jointly train a “deep” neural network **on all tasks**
 - ▶ Have a separate output layer (head) for each task
 - ▶ Each head is a Bayesian linear regression (recall DNGO)
- This uses meta-learning for feature extraction on the hyperparameter configurations



[Perrone et al. 2018]

Meta-Learning for HPO Approach 5: Learning a Black-box Optimization Algorithm from Data

- Learning a blackbox optimization algorithm
 - ▶ Use $\mathcal{D}_{\text{meta}}$ to learn a mapping from \mathcal{D}_{new} to the next configuration λ to evaluate
 - ▶ This mapping can be a (recurrent) neural net $\text{NN}_\phi : \mathcal{D}_{\text{new}} \mapsto \lambda$ parameterized by weights ϕ

Meta-Learning for HPO Approach 5: Learning a Black-box Optimization Algorithm from Data

- Learning a blackbox optimization algorithm
 - ▶ Use $\mathcal{D}_{\text{meta}}$ to learn a mapping from \mathcal{D}_{new} to the next configuration λ to evaluate
 - ▶ This mapping can be a (recurrent) neural net $\text{NN}_\phi : \mathcal{D}_{\text{new}} \mapsto \lambda$ parameterized by weights ϕ
 - ▶ This mapping NN_ϕ constitutes a blackbox optimization algorithm

Meta-Learning for HPO Approach 5: Learning a Black-box Optimization Algorithm from Data

- Learning a blackbox optimization algorithm
 - ▶ Use $\mathcal{D}_{\text{meta}}$ to learn a mapping from \mathcal{D}_{new} to the next configuration λ to evaluate
 - ▶ This mapping can be a (recurrent) neural net $\text{NN}_\phi : \mathcal{D}_{\text{new}} \mapsto \lambda$ parameterized by weights ϕ
 - ▶ This mapping NN_ϕ constitutes a blackbox optimization algorithm
- Existing approaches for learning a blackbox optimizer
 - ▶ Gradient descent on ϕ [Chen et al. 2016]
 - ★ Simplest technique, but requires backpropagation through the optimization trace
 - ★ This also requires the blackbox functions f used for training to be differentiable

Meta-Learning for HPO Approach 5: Learning a Black-box Optimization Algorithm from Data

- Learning a blackbox optimization algorithm
 - ▶ Use $\mathcal{D}_{\text{meta}}$ to learn a mapping from \mathcal{D}_{new} to the next configuration λ to evaluate
 - ▶ This mapping can be a (recurrent) neural net $\text{NN}_\phi : \mathcal{D}_{\text{new}} \mapsto \lambda$ parameterized by weights ϕ
 - ▶ This mapping NN_ϕ constitutes a blackbox optimization algorithm
- Existing approaches for learning a blackbox optimizer
 - ▶ Gradient descent on ϕ [Chen et al. 2016]
 - ★ Simplest technique, but requires backpropagation through the optimization trace
 - ★ This also requires the blackbox functions f used for training to be differentiable
 - ▶ Reinforcement learning [Li and Malik. 2016]
 - ★ Can be harder to get to work, but does not require differentiable f

Meta-Learning for HPO Approach 6: Learning Algorithm Parts

- Learning a complete optimization algorithm requires a lot of data
- It would be more sample-efficient to only replace hand-designed parts of an algorithm

Meta-Learning for HPO Approach 6: Learning Algorithm Parts

- Learning a complete optimization algorithm requires a lot of data
- It would be more sample-efficient to only replace hand-designed parts of an algorithm
- In Bayesian optimization, a critical hand-designed heuristic is the acquisition function
 - ▶ Trade-off between exploitation and exploration, e.g., via PI, EI, UCB, ES, KG, ...
 - ▶ Depending on the problem at hand, you might need a different acquisition function

Meta-Learning for HPO Approach 6: Learning Algorithm Parts

- Learning a complete optimization algorithm requires a lot of data
- It would be more sample-efficient to only replace hand-designed parts of an algorithm
- In Bayesian optimization, a critical hand-designed heuristic is the acquisition function
 - ▶ Trade-off between exploitation and exploration, e.g., via PI, EI, UCB, ES, KG, ...
 - ▶ Depending on the problem at hand, you might need a different acquisition function
- Idea: Learn a neural acquisition function from data, but still make use of the sample efficiency of Gaussian processes [Volpp et al. 2019]

Meta-Learning for HPO Approach 6: Learning Algorithm Parts

- Learning a complete optimization algorithm requires a lot of data
- It would be more sample-efficient to only replace hand-designed parts of an algorithm
- In Bayesian optimization, a critical hand-designed heuristic is the acquisition function
 - ▶ Trade-off between exploitation and exploration, e.g., via PI, EI, UCB, ES, KG, ...
 - ▶ Depending on the problem at hand, you might need a different acquisition function
- Idea: Learn a neural acquisition function from data, but still make use of the sample efficiency of Gaussian processes [Volpp et al. 2019]
- Two options:
 - ▶ Only depend on predicted mean and variance: $u_\phi(\lambda) = u_\phi(\mu_t(\lambda), \sigma_t(\lambda))$
 - ★ This allows to learn a general acquisition function

Meta-Learning for HPO Approach 6: Learning Algorithm Parts

- Learning a complete optimization algorithm requires a lot of data
- It would be more sample-efficient to only replace hand-designed parts of an algorithm
- In Bayesian optimization, a critical hand-designed heuristic is the acquisition function
 - ▶ Trade-off between exploitation and exploration, e.g., via PI, EI, UCB, ES, KG, ...
 - ▶ Depending on the problem at hand, you might need a different acquisition function
- Idea: Learn a neural acquisition function from data, but still make use of the sample efficiency of Gaussian processes [Volpp et al. 2019]
- Two options:
 - ▶ Only depend on predicted mean and variance: $u_\phi(\lambda) = u_\phi(\mu_t(\lambda), \sigma_t(\lambda))$
 - ★ This allows to learn a general acquisition function
 - ▶ Also depend on the λ value: $u_\phi(\lambda) = u_\phi(\mu_t(\lambda), \sigma_t(\lambda), \lambda)$
 - ★ This allows to fine-tune to the characteristics of $\mathcal{D}_{\text{meta}}$ (e.g., avoid poor parts of the space)

Questions to Answer for Yourself / Discuss with Friends

- **Repetition.** What are the different kinds of meta-features which can be used to describe machine learning datasets?
- **Repetition.** List all the different ways of using the meta data for HPO you recall
- **Discussion.** In the various meta-learning approaches, what will happen if all prior tasks are dissimilar to the target task?

Speedup Techniques for Hyperparameter Optimization

Overview of Multi-Fidelity Optimization

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Motivating Example

- One possible cheap approximation of an expensive function: use a data subset
 - ▶ Many cheap evaluations on small subsets
 - ▶ Few expensive evaluations on the full data



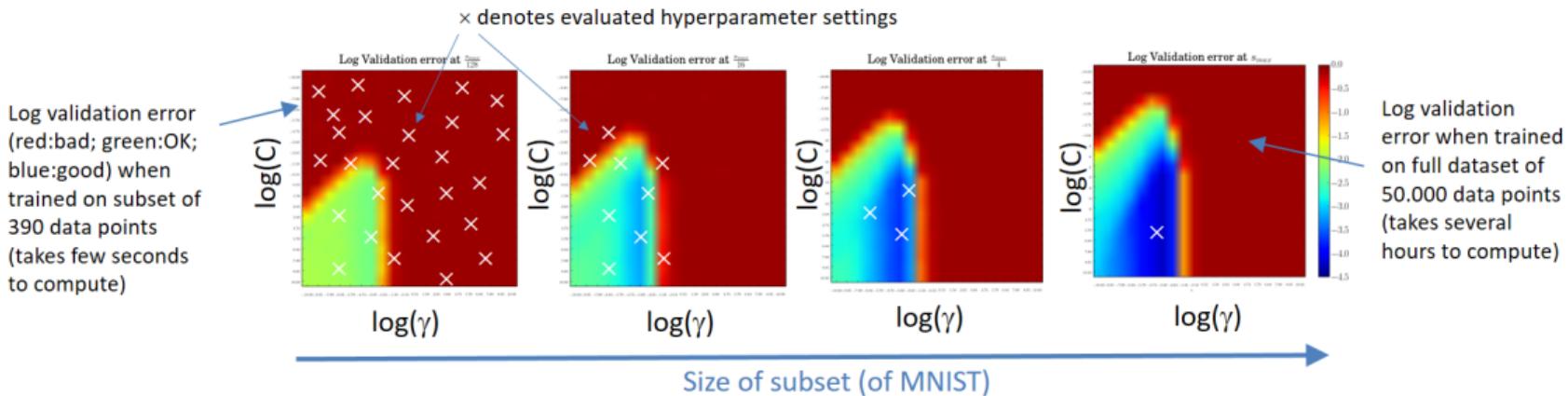
Motivating Example

- One possible cheap approximation of an expensive function: use a data subset

- ▶ Many cheap evaluations on small subsets
 - ▶ Few expensive evaluations on the full data

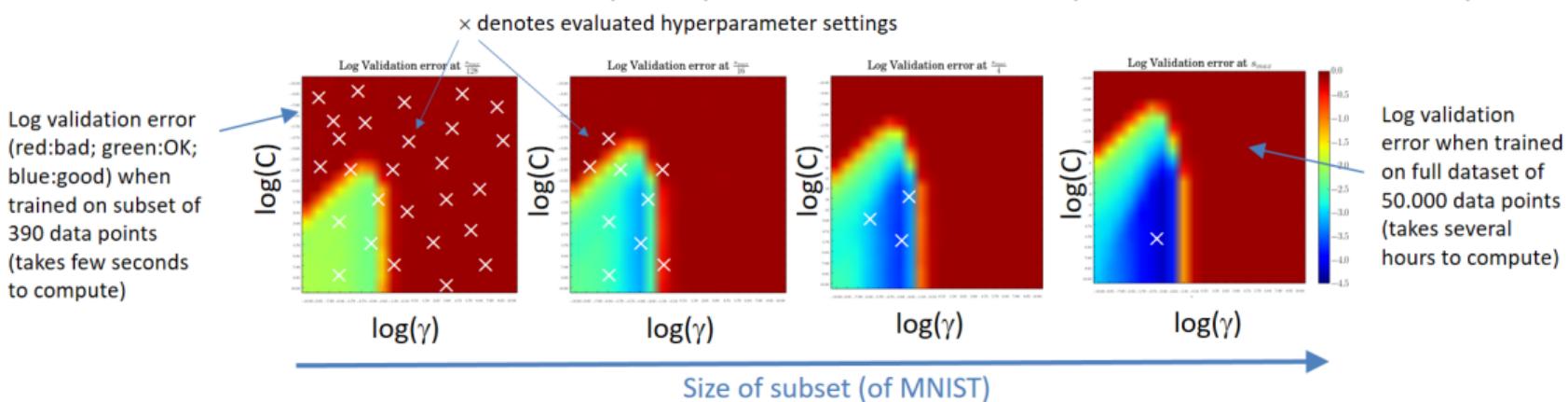


- E.g.: Support Vector Machines (SVM) on MNIST dataset (hyperparameters: C , γ)



Motivating Example

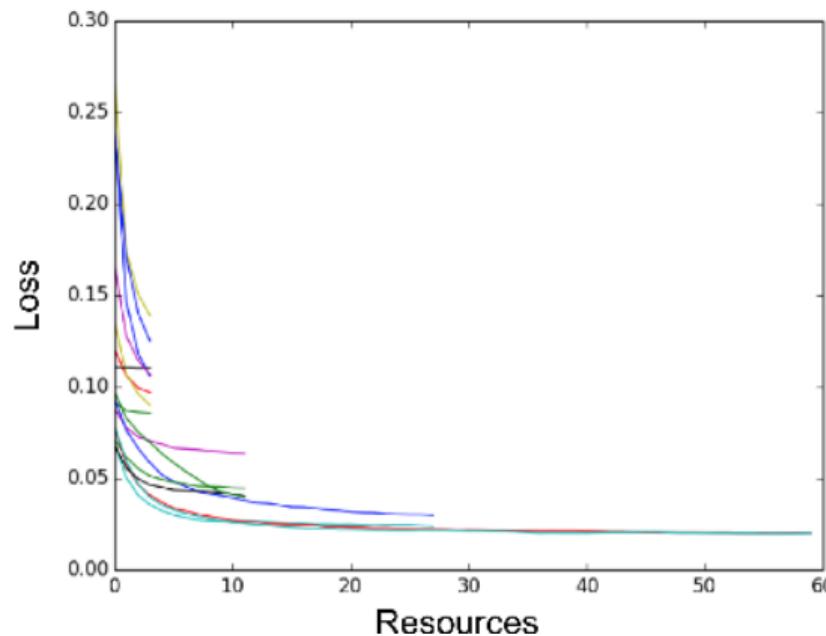
- One possible cheap approximation of an expensive function: use a data subset
 - ▶ Many cheap evaluations on small subsets
 - ▶ Few expensive evaluations on the full data
- E.g.: Support Vector Machines (SVM) on MNIST dataset (hyperparameters: C , γ)



→ up to 1000x speedups over blackbox optimization on full data [Klein et al, AISTATS 2017]

Motivating Example 2: Shorter Runs of Anytime Algorithms

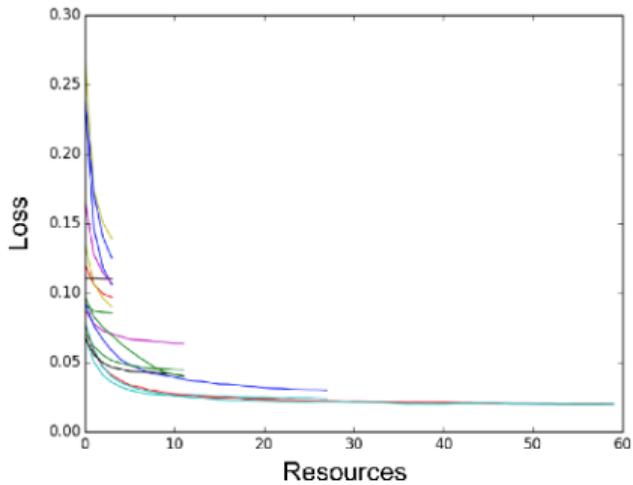
- Performance with shorter runs of an anytime algorithm (such as SGD):



Multi-Fidelity Optimization In General

Exploit cheap approximations of an expensive blackbox function → afford more configurations

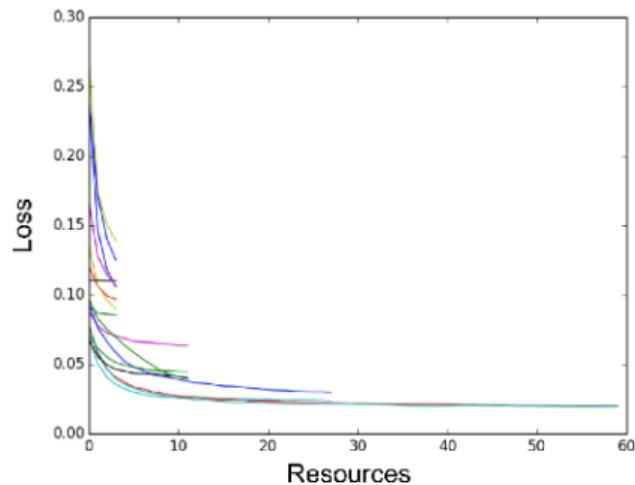
- Idea: eliminate poor configurations early, allocate more resources to promising ones.



Multi-Fidelity Optimization In General

Exploit cheap approximations of an expensive blackbox function → afford more configurations

- Idea: eliminate poor configurations early, allocate more resources to promising ones.
- Possible Resources:
 - ▶ Data subset size
 - ▶ Runtime / # epochs / # iterations

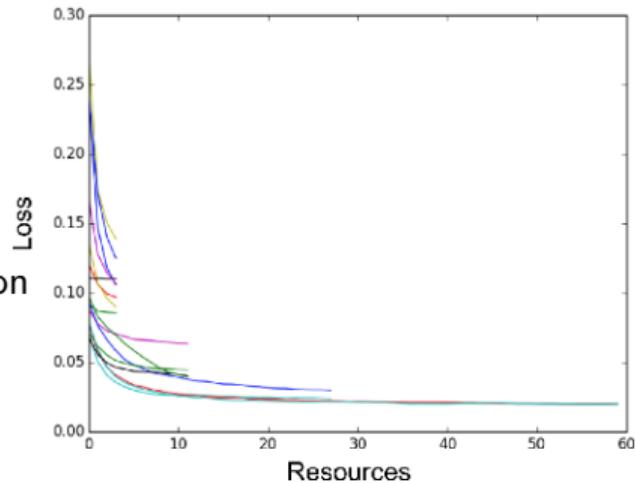


Multi-Fidelity Optimization In General

Exploit cheap approximations of an expensive blackbox function → afford more configurations

- Idea: eliminate poor configurations early, allocate more resources to promising ones.

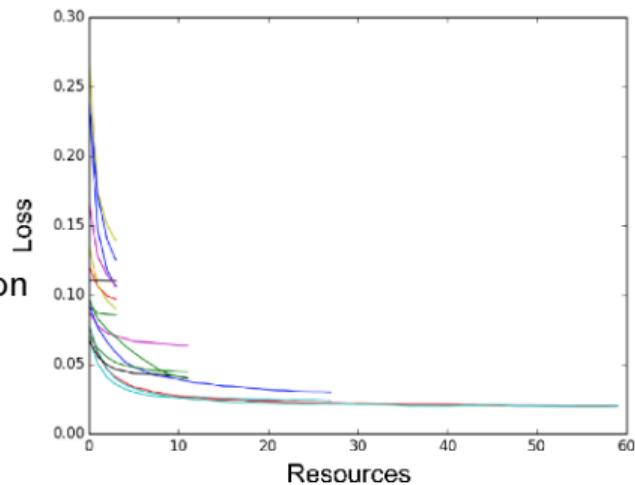
- Possible Resources:
 - ▶ Data subset size
 - ▶ Runtime / # epochs / # iterations
 - ▶ Downsampled size of images in object recognition
 - ▶ Depth / width of neural networks



Multi-Fidelity Optimization In General

Exploit cheap approximations of an expensive blackbox function → afford more configurations

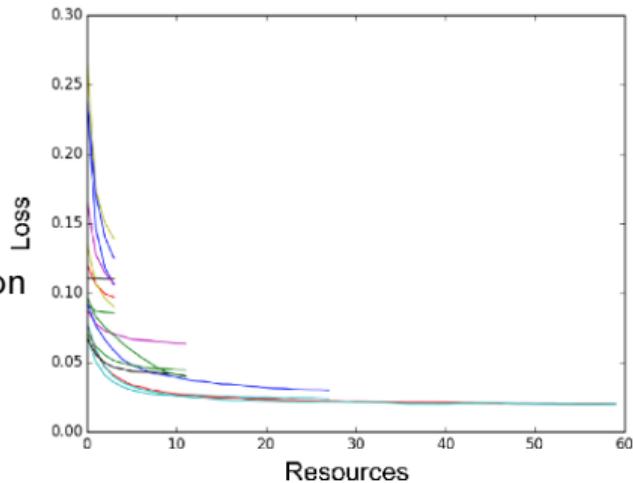
- Idea: eliminate poor configurations early, allocate more resources to promising ones.
- Possible Resources:
 - ▶ Data subset size
 - ▶ Runtime / # epochs / # iterations
 - ▶ Downsampled size of images in object recognition
 - ▶ Depth / width of neural networks
 - ▶ Number of trees
 - ▶ Number of features
 - ▶ Number of cross validation folds



Multi-Fidelity Optimization In General

Exploit cheap approximations of an expensive blackbox function → afford more configurations

- Idea: eliminate poor configurations early, allocate more resources to promising ones.
 - Possible Resources:
 - Data subset size
 - Runtime / # epochs / # iterations
 - Downsampled size of images in object recognition
 - Depth / width of neural networks
 - Number of trees
 - Number of features
 - Number of cross validation folds
 - General concept, applicable even in fields outside ML, e.g., fluid simulation:
 - Number of particles
 - Time scale of simulation



General Remarks on Multi-Fidelity Optimization

- Often, we have a choice which resources we use as budget

General Remarks on Multi-Fidelity Optimization

- Often, we have a choice which resources we use as budget
- For multi-fidelity optimization to be helpful, performance with low budgets should be informative about performance with high budgets

General Remarks on Multi-Fidelity Optimization

- Often, we have a choice which resources we use as budget
- For multi-fidelity optimization to be helpful, performance with low budgets should be informative about performance with high budgets
- In the simplest case: good with low resources \leftrightarrow good with high resources.
 - ▶ In practice, this is of course not always true

How Useful is the Cheap Approximation? The Rank Correlation

Given:

- A set of configurations $\Lambda = \{\lambda_1, \dots, \lambda_n\}$
- The performances $f(\lambda_1), \dots, f(\lambda_n)$ on the expensive black box
- The performances $g(\lambda_1), \dots, g(\lambda_n)$ on a cheap approximation of the black box

How Useful is the Cheap Approximation? The Rank Correlation

Given:

- A set of configurations $\Lambda = \{\lambda_1, \dots, \lambda_n\}$
- The performances $f(\lambda_1), \dots, f(\lambda_n)$ on the expensive black box
- The performances $g(\lambda_1), \dots, g(\lambda_n)$ on a cheap approximation of the black box

We compute the **Spearman rank correlation** between $[f(\lambda_1), \dots, f(\lambda_n)]$ and $[g(\lambda_1), \dots, g(\lambda_n)]$

- If this is high (in the extreme: 1), the relative ranking of the configurations is the same on f and g
 - ▶ In that case, we can optimize cheaply on g and also obtain an optimum for f
- If it is low (≈ 0), optimizing g does not tell us anything about f

How Useful is the Cheap Approximation? The Rank Correlation

Given:

- A set of configurations $\Lambda = \{\lambda_1, \dots, \lambda_n\}$
- The performances $f(\lambda_1), \dots, f(\lambda_n)$ on the expensive black box
- The performances $g(\lambda_1), \dots, g(\lambda_n)$ on a cheap approximation of the black box

We compute the **Spearman rank correlation** between $[f(\lambda_1), \dots, f(\lambda_n)]$ and $[g(\lambda_1), \dots, g(\lambda_n)]$

- If this is high (in the extreme: 1), the relative ranking of the configurations is the same on f and g
 - ▶ In that case, we can optimize cheaply on g and also obtain an optimum for f
- If it is low (≈ 0), optimizing g does not tell us anything about f

Goal: find approximations g that are very cheap but have high rank correlations with f

Questions to Answer for Yourself / Discuss with Friends

- **Repetition.** Which cheap approximation is better in this hypothetical case?
 - ▶ Downscaling images (5x cheaper, rank correlation of 0.8)
 - ▶ Less epoch of SGD (4x cheaper, rank correlation of 0.75)
- **Discussion.** Can you think of an application of your interest where you would likely have a good multi-fidelity approximation?

Speedup Techniques for Hyperparameter Optimization

Hyperband

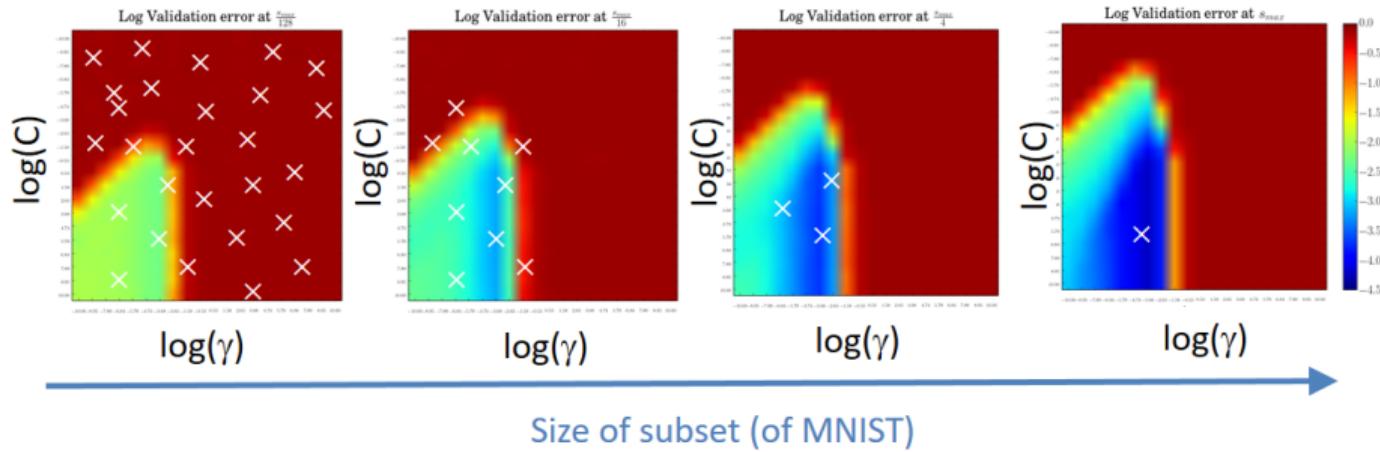
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

A Simple Multi-Fidelity Algorithms: Successive Halving (SH)

[Jamieson and Talwalkar, AISTATS 2016]

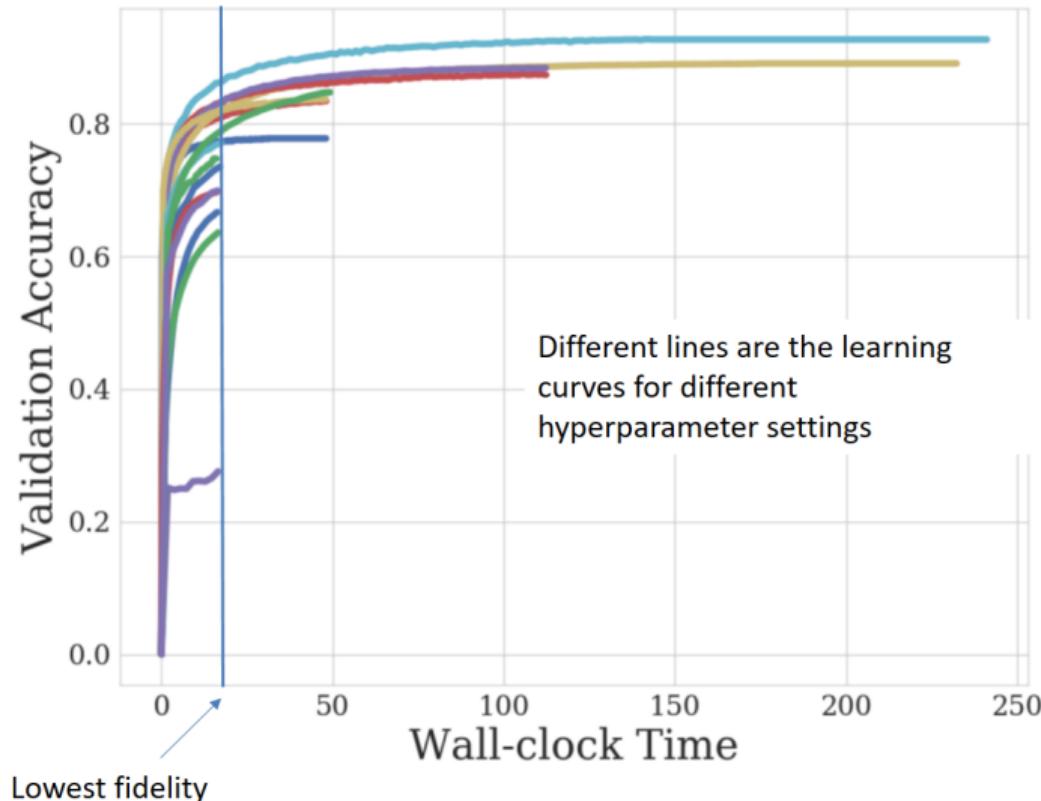
- A very simple algorithm:

- ▶ Sample N configurations uniformly at random & evaluate them on the cheapest fidelity
- ▶ Keep the best half (or third), move them to the next fidelity
- ▶ Iterate until the most expensive fidelity (= original expensive black box)



The Same SH Algorithm When the Fidelity is Runtime

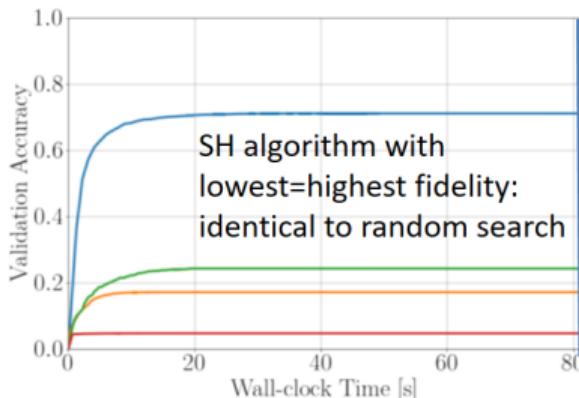
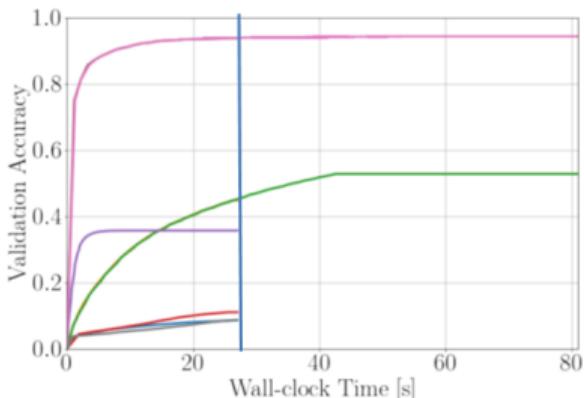
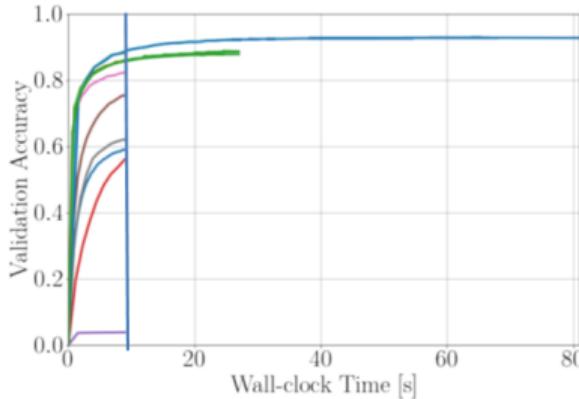
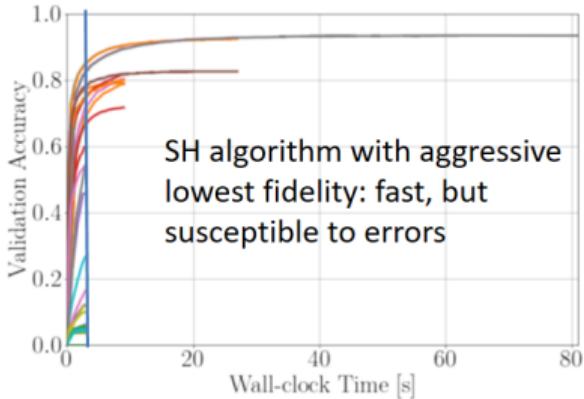
[Jamieson and Talwalkar, AISTATS 2016]



An Extension of SH with Theoretical Guarantees: Hyperband

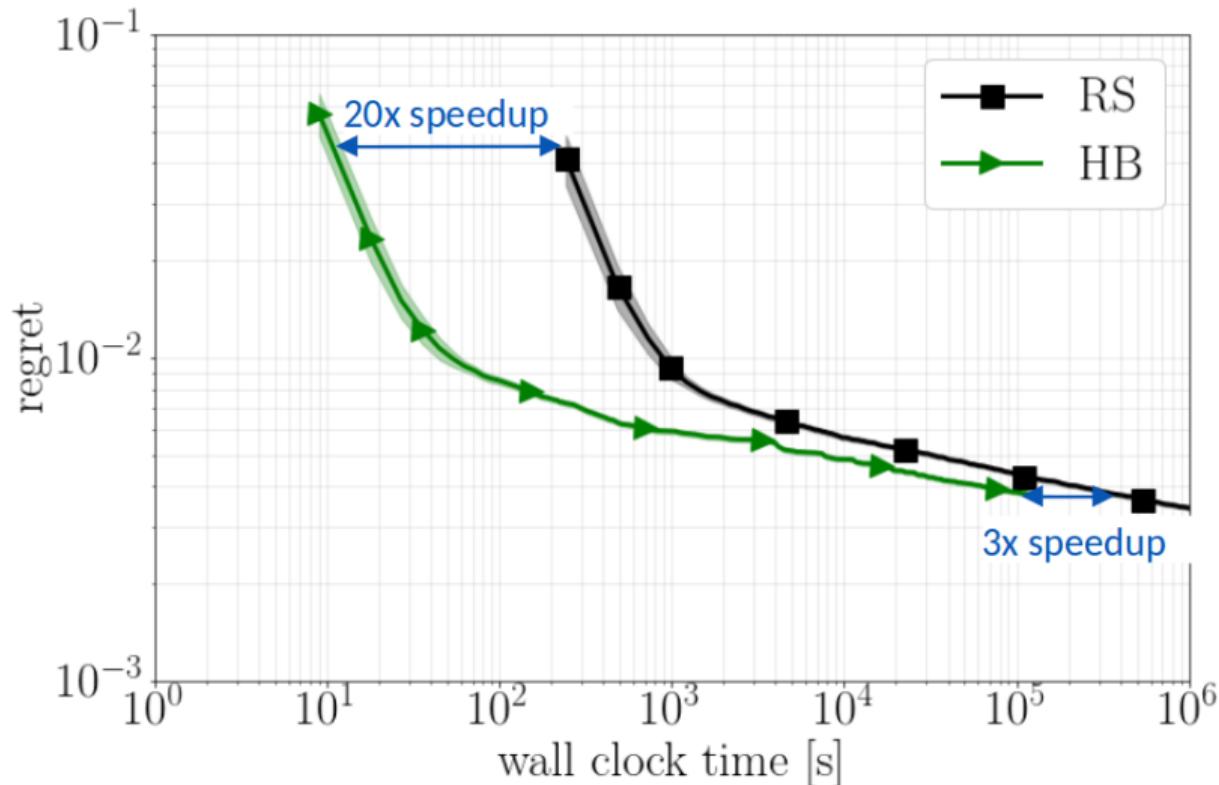
[Li et al., JMLR 2018]

- Main Idea:
hedge against
errors in cheap
approximations
- Algorithm:
run multiple copies
of SH in parallel,
starting at
different cheapest
fidelities



Empirical Evaluation: Hyperband vs. Random Search

[Falkner, Klein & Hutter, ICML 2018]



Questions to Answer for Yourself / Discuss with Friends

- Discussion. How do you think Hyperband would compare to successive halving using the most aggressive fidelity?
- Discussion. How slow is Hyperband in the worst case?

Speedup Techniques for Hyperparameter Optimization

BOHB

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Robust and Efficient Hyperparameter Optimization at Scale

Desiderata for a practical solution to the hyperparameter optimization problem:

- Strong Anytime Performance
- Strong Final Performance
- Scalability
- Robustness & Flexibility
- Computational Efficiency
- Effective Use of Parallel Resources
- Conceptual / Algorithmic Simplicity

Robust and Efficient Hyperparameter Optimization at Scale

Desiderata for a practical solution to the hyperparameter optimization problem:

- Strong Anytime Performance
- Strong Final Performance
- Scalability
- Robustness & Flexibility
- Computational Efficiency
- Effective Use of Parallel Resources
- Conceptual / Algorithmic Simplicity

To fulfill all of these desiderata, BOHB [Falkner, Klein and Hutter, ICML 2018] combines Bayesian Optimization with Hyperband

BOHB

- BOHB combines the advantages of Bayesian Optimization and Hyperband
 - ▶ Bayesian Optimization for choosing configurations to achieve strong final performance
 - ▶ Hyperband to choose the budgets for good anytime performance

BOHB

- BOHB combines the advantages of Bayesian Optimization and Hyperband
 - ▶ Bayesian Optimization for choosing configurations to achieve strong final performance
 - ▶ Hyperband to choose the budgets for good anytime performance
- BOHB replaces the random selection of configurations at the beginning of each HB iteration by a model-based search

BOHB

- BOHB combines the advantages of Bayesian Optimization and Hyperband
 - ▶ Bayesian Optimization for choosing configurations to achieve strong final performance
 - ▶ Hyperband to choose the budgets for good anytime performance
- BOHB replaces the random selection of configurations at the beginning of each HB iteration by a model-based search
- Details of the model
 - ▶ Variant of the Tree Parzen Estimator, with a product kernel
 - ▶ Models are fitted independently to the data for one budget at a time
 - ★ Specifically, always the highest budget that has enough data points

BOHB: Algorithm

Pseudocode for sampling in BOHB

Input : observations D , fraction of random runs ρ , percentile γ , number of samples N_s , minimum number of points N_{min} to build a model, and bandwidth factor b_w

Output: next configuration to evaluate

- 1 **if** $rand() < \rho$ **then return** *random configuration*
 - 2 $b = \arg \max\{D_b : |D_b| \geq N_{min} + 2\}$
 - 3 **if** $b = \emptyset$ **then return** *random configuration*
 - 4 Fit KDEs according to Equation ??
 - 5 Draw N_s samples according to $l'(\lambda)$
 - 6 **return** *sample with highest ratio $l(\lambda)/g(\lambda)$*
-

BOHB: Algorithm

Pseudocode for sampling in BOHB

Input : observations D , fraction of random runs ρ , percentile γ , number of samples N_s , minimum number of points N_{min} to build a model, and bandwidth factor b_w

Output: next configuration to evaluate

- 1 **if** $rand() < \rho$ **then return** *random configuration*
 - 2 $b = \arg \max\{D_b : |D_b| \geq N_{min} + 2\}$
 - 3 **if** $b = \emptyset$ **then return** *random configuration*
 - 4 Fit KDEs according to Equation ??
 - 5 Draw N_s samples according to $l'(\lambda)$
 - 6 **return** *sample with highest ratio $l(\lambda)/g(\lambda)$*
-

$$l(\lambda) = p(c(\lambda) \leq \gamma | D_b)$$

$$g(\lambda) = p(c(\lambda) > \gamma | D_b)$$

(1)

BOHB: Algorithm

Pseudocode for sampling in BOHB

Input : observations D , fraction of random runs ρ , percentile γ , number of samples N_s , minimum number of points N_{min} to build a model, and bandwidth factor b_w

Output: next configuration to evaluate

- 1 **if** $rand() < \rho$ **then return** *random configuration*
 - 2 $b = \arg \max\{D_b : |D_b| \geq N_{min} + 2\}$
 - 3 **if** $b = \emptyset$ **then return** *random configuration*
 - 4 Fit KDEs according to Equation ??
 - 5 Draw N_s samples according to $l'(\lambda)$
 - 6 **return** *sample with highest ratio $l(\lambda)/g(\lambda)$*
-

$$l(\lambda) = p(c(\lambda) \leq \gamma | D_b) \quad g(\lambda) = p(c(\lambda) > \gamma | D_b) \quad (1)$$

- Note: $l'(\lambda)$ is similar to $l(\lambda)$ but has larger bandwidths

BOHB: Empirical Evaluation

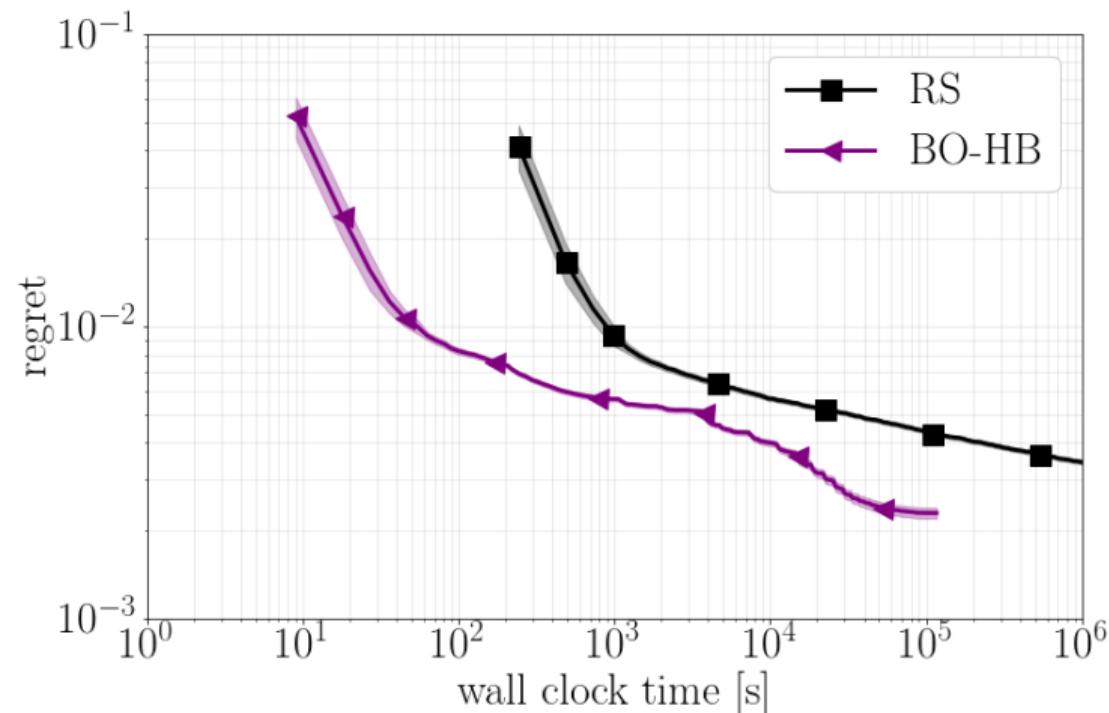


Figure: Performance of RS and BOHB on Auto-Net on dataset Adult

BOHB: Empirical Evaluation

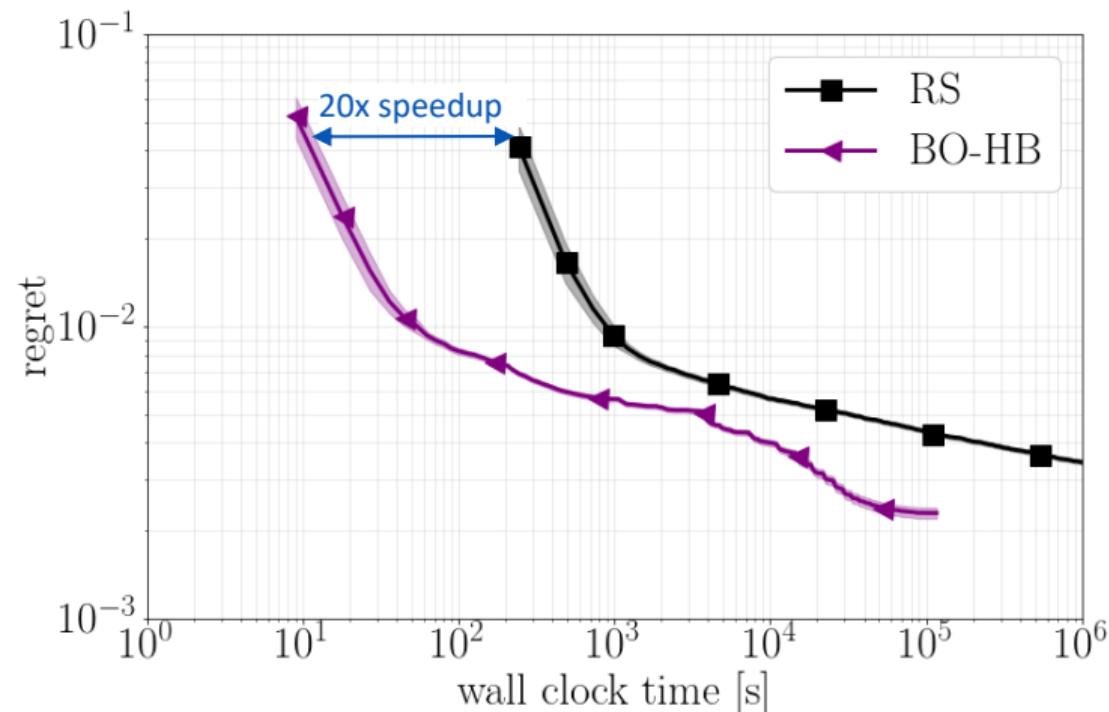


Figure: Performance of RS and BOHB on Auto-Net on dataset Adult

BOHB: Empirical Evaluation

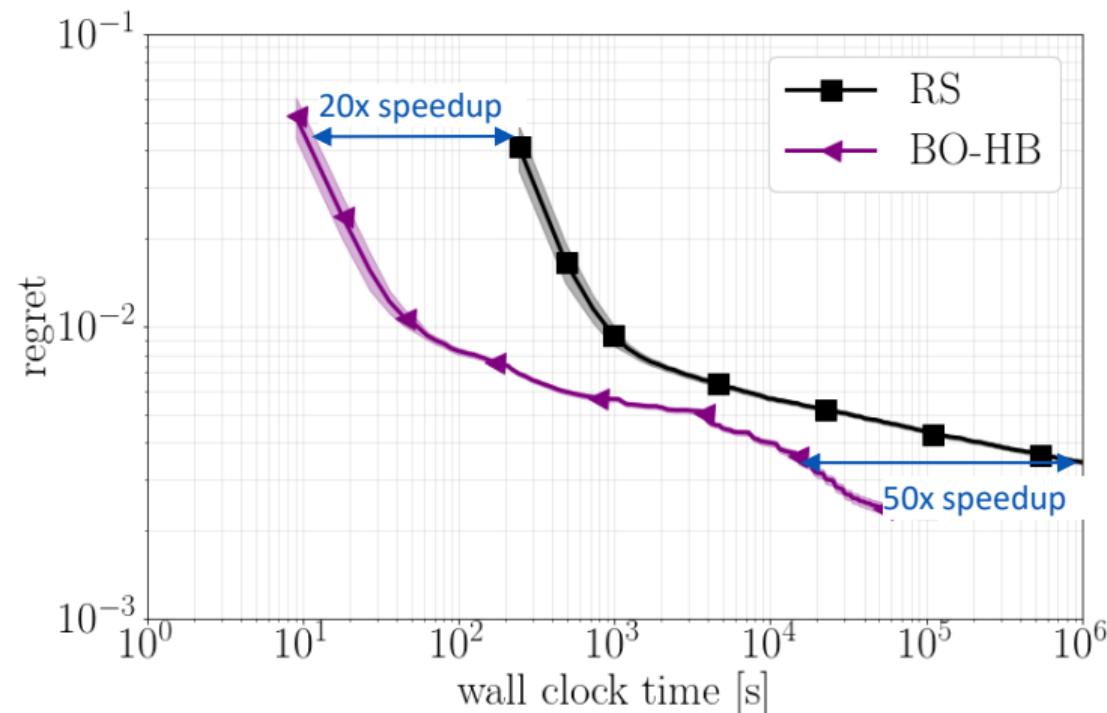


Figure: Performance of RS and BOHB on Auto-Net on dataset Adult

BOHB: Empirical Evaluation

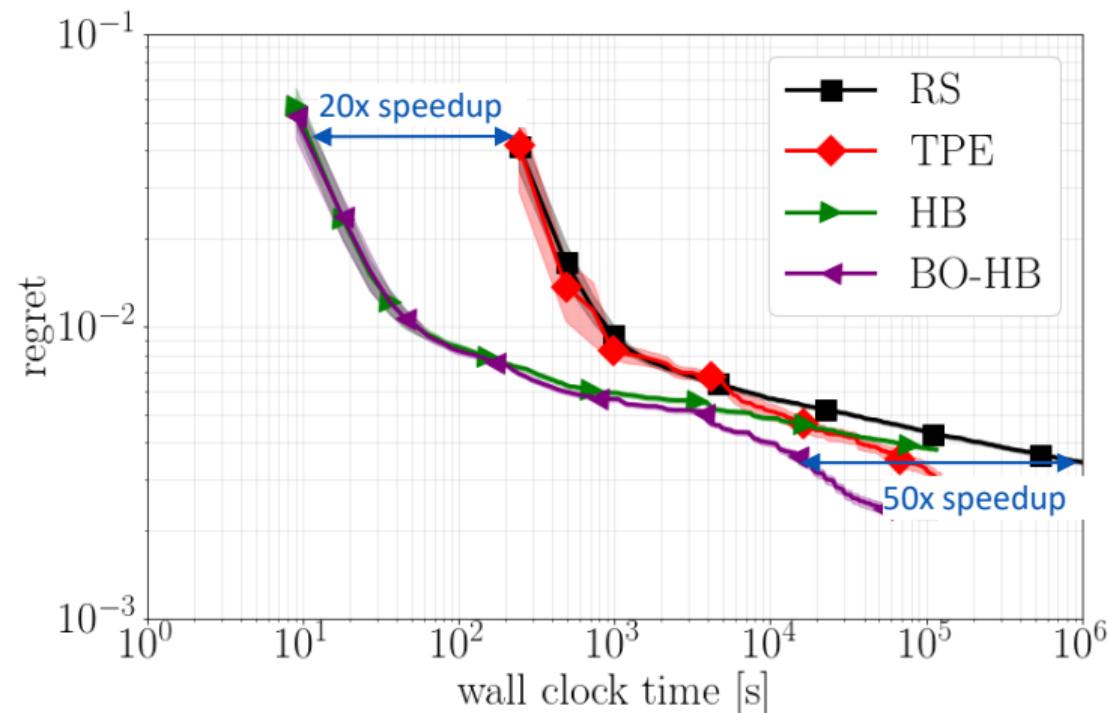


Figure: Performance of RS, TPE, HB and BOHB on Auto-Net on dataset Adult

BOHB: Different number of parallel workers

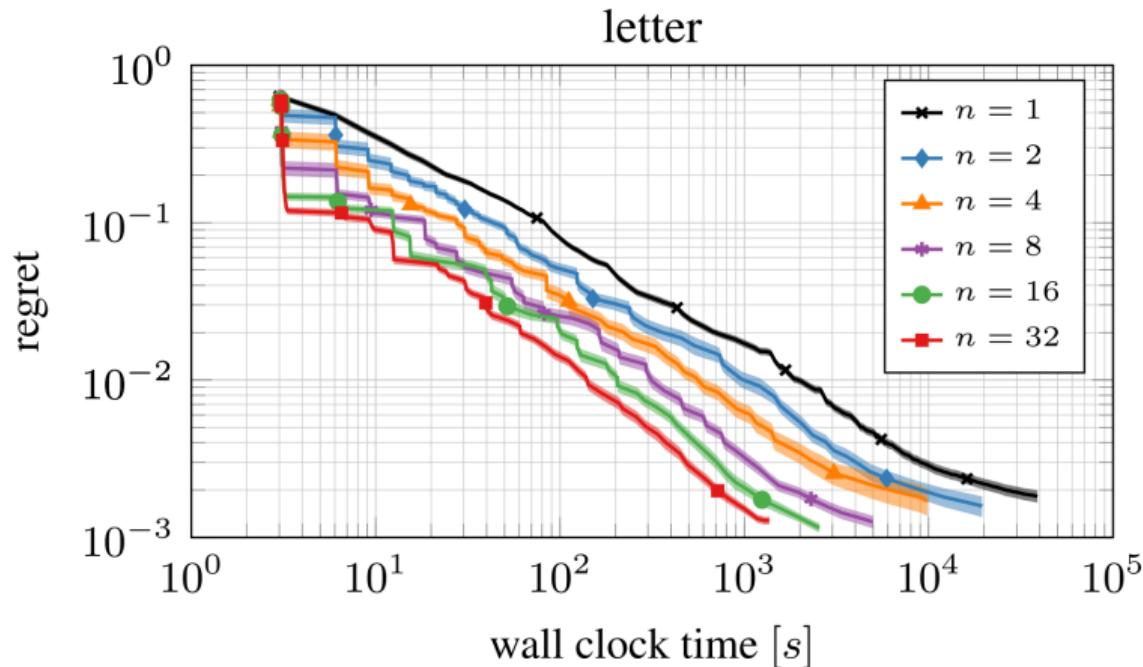


Figure: Performance of BOHB with different number of parallel workers on the letter surrogate benchmark for 128 iterations

BOHB: Optimization of a Bayesian Neural Network

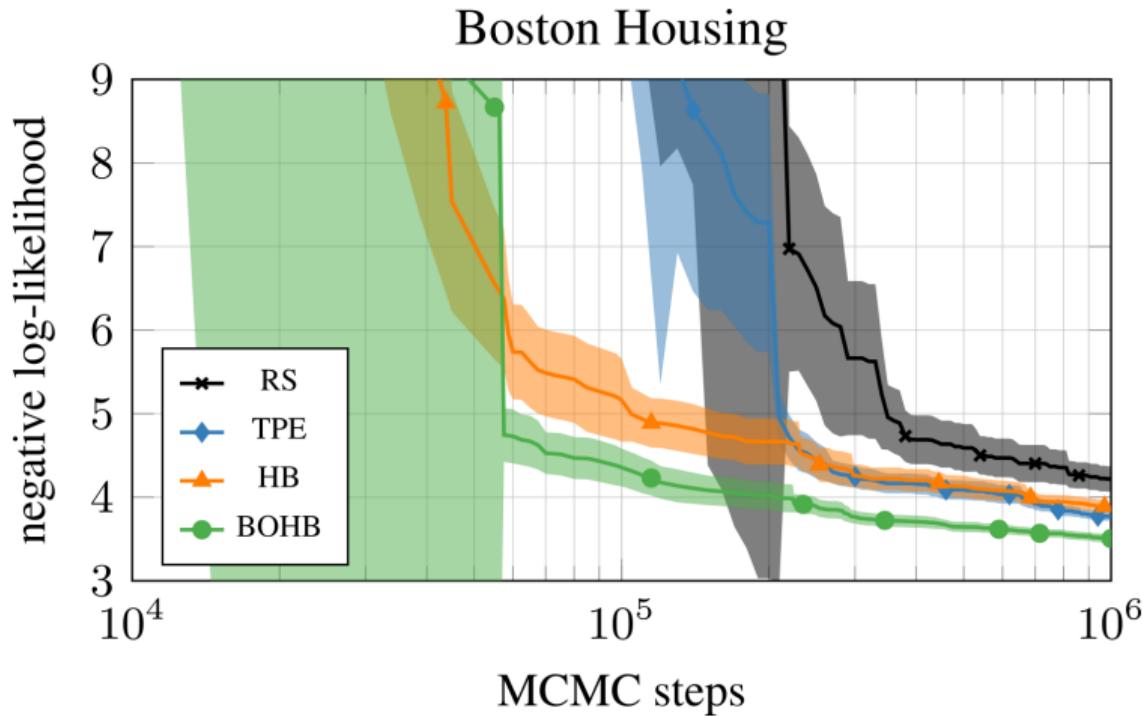


Figure: Optimization of 5 hyperparameters of a Bayesian neural network trained with SGHMC.

BOHB: Optimization of a Reinforcement Learning Agent

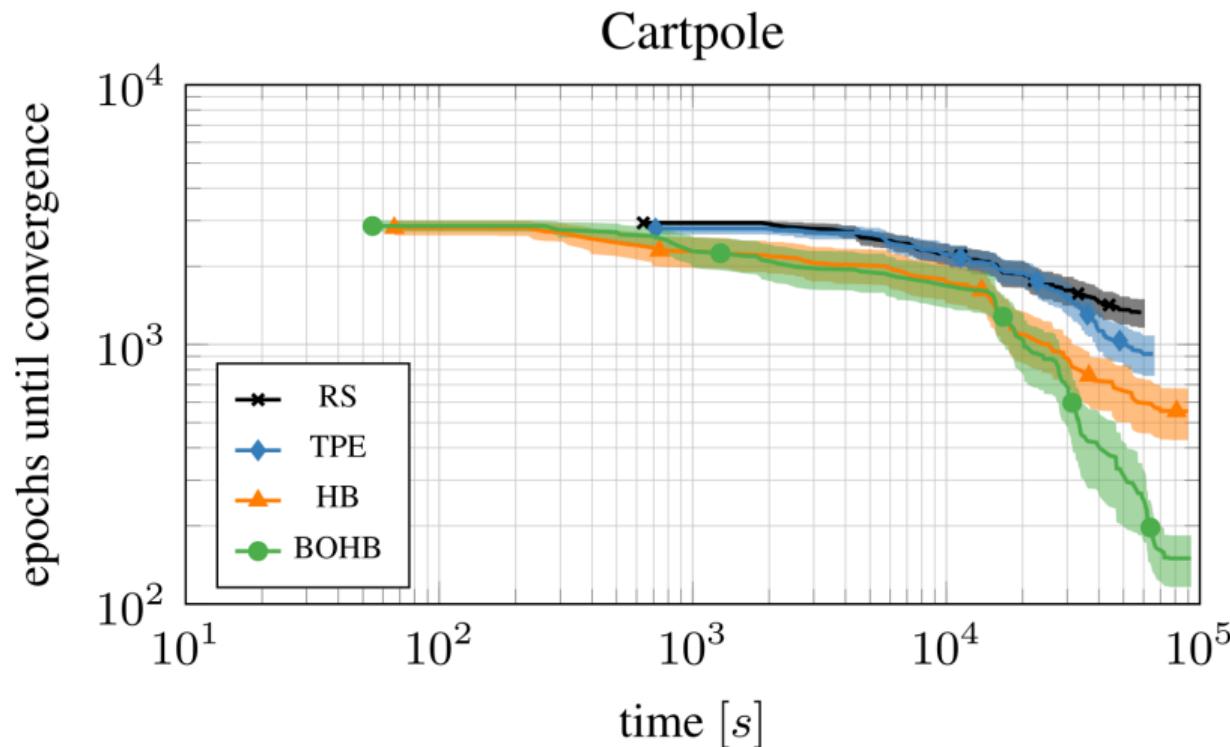


Figure: Hyperparameter optimization of 8 hyperparameters of PPO on the cartpole task.

BOHB: Optimization of an SVM

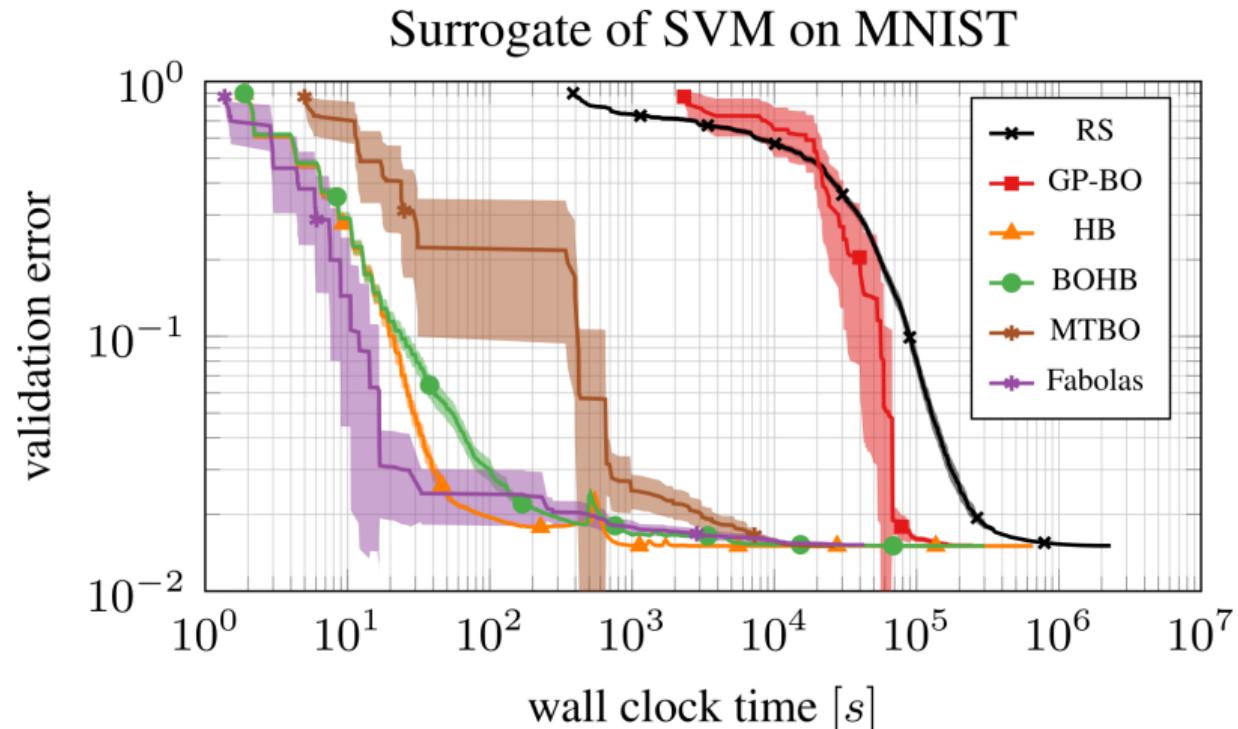


Figure: Comparison on the SVM on MNIST surrogate benchmark

BOHB: Counting Ones

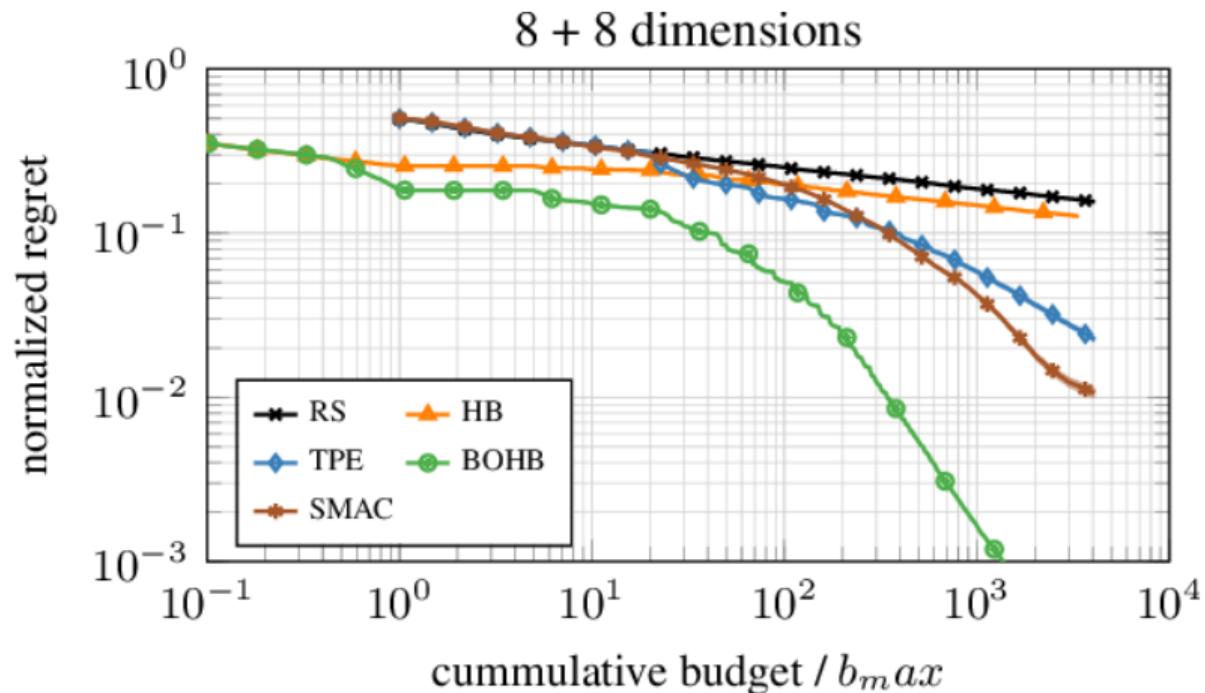


Figure: Results for the counting ones problem in 16 dimensional space with 8 categorical and 8 continuous hyperparameters.

BOHB: Optimization of a Feedforward Network

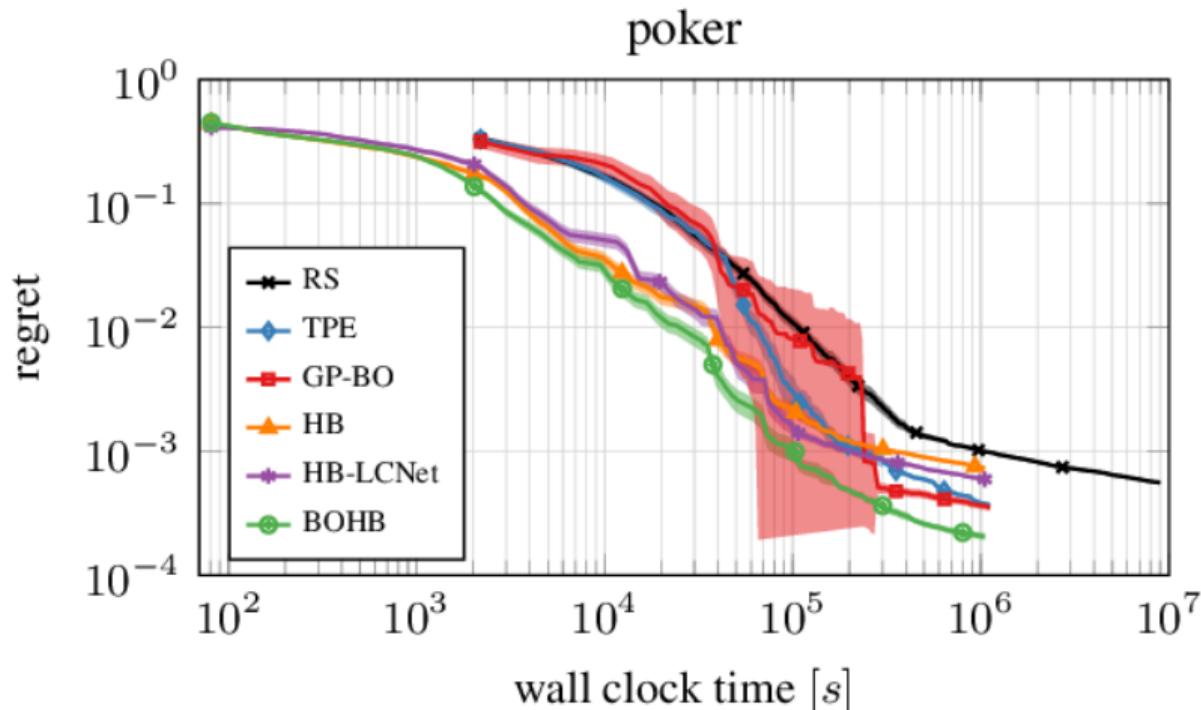


Figure: Optimizing six hyperparameters of a feed-forward neural network on featurized datasets; results are based on surrogate benchmarks.

Questions to Answer for Yourself / Discuss with Friends

- **Repetition.** Why does BOHB interleave randomly sampled configurations in the optimization process?
- **Discussion.** What are the advantages of the Parzen estimator model over more advanced models such as random forests or Gaussian processes?

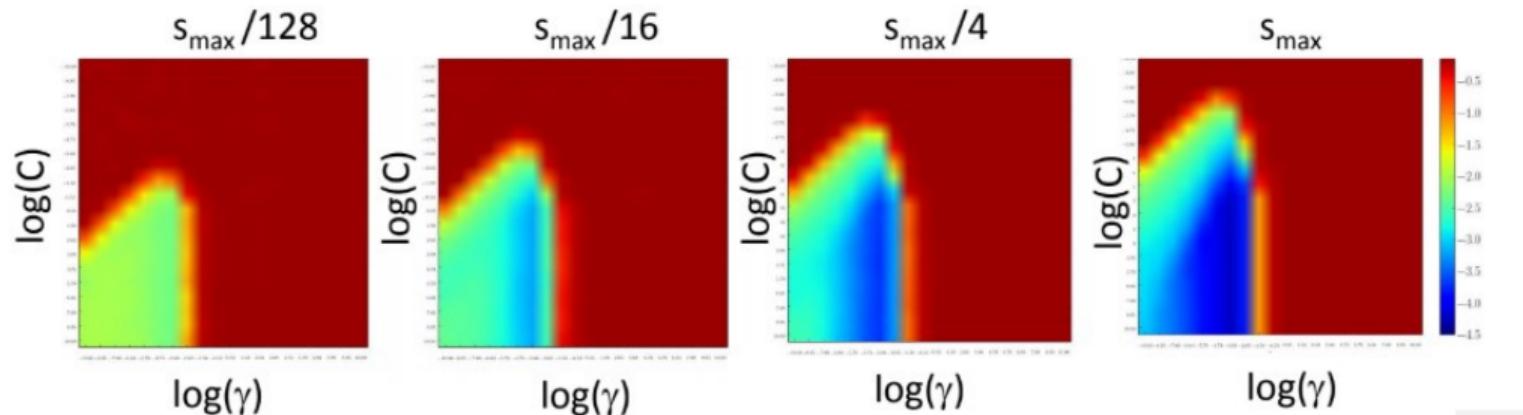
Speedup Techniques for Hyperparameter Optimization

Multi-fidelity Bayesian optimization

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Motivating example

- Performance of an SVM on MNIST and subsets of it:



- ▶ Computational cost grows quadratically in dataset size z
- ▶ Error shrinks smoothly with z
- Evaluations on the smallest subset (about 400 data points) cost $10\,000\times$ less than on the full data set

Idea of Multi-fidelity Bayesian optimization [Kandasamy et al. 2017; Klein et al. 2016]

- Recall: standard Bayesian optimization uses a model $\hat{c}(\lambda) \approx y$ to select the next λ

Idea of Multi-fidelity Bayesian optimization [Kandasamy et al. 2017; Klein et al. 2016]

- Recall: standard Bayesian optimization uses a model $\hat{c}(\lambda) \approx y$ to select the next λ
- Multi-fidelity Bayesian optimization uses a model $\hat{c}(\lambda, z) \approx y$ to select the next (λ, z)
 - ▶ Here, $z \in \mathcal{Z}$ is the fidelity; \mathcal{Z} is the fidelity space, e.g., $\mathcal{Z} = [1, N_\bullet] \times [1, T_\bullet]$

Idea of Multi-fidelity Bayesian optimization [Kandasamy et al. 2017; Klein et al. 2016]

- Recall: standard Bayesian optimization uses a model $\hat{c}(\boldsymbol{\lambda}) \approx y$ to select the next $\boldsymbol{\lambda}$
- Multi-fidelity Bayesian optimization uses a model $\hat{c}(\boldsymbol{\lambda}, z) \approx y$ to select the next $(\boldsymbol{\lambda}, z)$
 - ▶ Here, $z \in \mathcal{Z}$ is the fidelity; \mathcal{Z} is the fidelity space, e.g., $\mathcal{Z} = [1, N_{\bullet}] \times [1, T_{\bullet}]$
- Denoting z_{\bullet} as the maximum fidelity (e.g., $z_{\bullet} = [N_{\bullet}, T_{\bullet}]$), our goal is to find:

$$\boldsymbol{\lambda}^* = \arg \min_{\boldsymbol{\lambda} \in \Lambda} f(\boldsymbol{\lambda}) = \arg \min_{\boldsymbol{\lambda} \in \Lambda} \hat{c}(\boldsymbol{\lambda}, z_{\bullet})$$

Idea of Multi-fidelity Bayesian optimization [Kandasamy et al. 2017; Klein et al. 2016]

- Recall: standard Bayesian optimization uses a model $\hat{c}(\boldsymbol{\lambda}) \approx y$ to select the next $\boldsymbol{\lambda}$
- Multi-fidelity Bayesian optimization uses a model $\hat{c}(\boldsymbol{\lambda}, z) \approx y$ to select the next $(\boldsymbol{\lambda}, z)$
 - ▶ Here, $z \in \mathcal{Z}$ is the fidelity; \mathcal{Z} is the fidelity space, e.g., $\mathcal{Z} = [1, N_{\bullet}] \times [1, T_{\bullet}]$
- Denoting z_{\bullet} as the maximum fidelity (e.g., $z_{\bullet} = [N_{\bullet}, T_{\bullet}]$), our goal is to find:

$$\boldsymbol{\lambda}^* = \arg \min_{\boldsymbol{\lambda} \in \Lambda} f(\boldsymbol{\lambda}) = \arg \min_{\boldsymbol{\lambda} \in \Lambda} \hat{c}(\boldsymbol{\lambda}, z_{\bullet})$$

- Implications for Bayesian optimization
 - ▶ Model \hat{c} needs to be good at extrapolating from small to large z
 - ▶ Acquisition function now also needs to select z (i.e., take into account cost of evaluations)

Entropy Search: Reminder

- Define the p_{\min} distribution given data \mathcal{D} :

$$p_{\min}(\boldsymbol{\lambda}^* \mid \mathcal{D}) := p(\boldsymbol{\lambda}^* \in \arg \min_{\boldsymbol{\lambda} \in \Lambda} f(\boldsymbol{\lambda}) \mid \mathcal{D})$$

- Entropy search aims to minimize the entropy $\mathcal{H}[p_{\min}]$ [Hennig and Schuler. 2012]
 - ▶ It aims to be maximally certain about the location of $\boldsymbol{\lambda}^*$

Entropy Search: Reminder

- Define the p_{\min} distribution given data \mathcal{D} :

$$p_{\min}(\boldsymbol{\lambda}^* \mid \mathcal{D}) := p(\boldsymbol{\lambda}^* \in \arg \min_{\boldsymbol{\lambda} \in \Lambda} f(\boldsymbol{\lambda}) \mid \mathcal{D})$$

- Entropy search aims to minimize the entropy $\mathcal{H}[p_{\min}]$ [Hennig and Schuler. 2012]
 - ▶ It aims to be maximally certain about the location of $\boldsymbol{\lambda}^*$
- In a nutshell: select $\boldsymbol{\lambda}$ that maximizes the following acquisition function:

$$u_{ES}(\boldsymbol{\lambda}) := \mathcal{H}[p_{\min}(\cdot \mid \mathcal{D})] - \mathbb{E}_{p(\tilde{c} \mid \boldsymbol{\lambda}, \mathcal{D})} [\mathcal{H}[p_{\min}(\cdot \mid \mathcal{D} \cup \langle \boldsymbol{\lambda}, \tilde{c} \rangle)]]$$

- We now care about the p_{\min} distribution for the maximal budget z_\bullet :

$$p_{\min}(\boldsymbol{\lambda}^* \mid \mathcal{D}) := p(\boldsymbol{\lambda}^* \in \arg \min_{\boldsymbol{\lambda} \in \Lambda} f(\boldsymbol{\lambda}, z_\bullet) \mid \mathcal{D})$$

- We still want to minimize the entropy $\mathcal{H}[p_{\min}]$

- We now care about the p_{\min} distribution for the maximal budget z_\bullet :

$$p_{\min}(\boldsymbol{\lambda}^* \mid \mathcal{D}) := p(\boldsymbol{\lambda}^* \in \arg \min_{\boldsymbol{\lambda} \in \Lambda} f(\boldsymbol{\lambda}, z_\bullet) \mid \mathcal{D})$$

- We still want to minimize the entropy $\mathcal{H}[p_{\min}]$
- Now we aim for the biggest reduction in entropy per time spent
 - ▶ Now we don't model only f , but also the cost $c(\boldsymbol{\lambda}, z)$
 - ▶ We choose the next $(\boldsymbol{\lambda}, z)$ by maximizing:

$$u_{ES}(\boldsymbol{\lambda}, z \mid \mathcal{D}) := \mathbb{E}_{p(\tilde{c} \mid (\boldsymbol{\lambda}, z), \mathcal{D})} \left[\frac{\mathcal{H}[p_{\min}(\cdot \mid \mathcal{D})] - \mathcal{H}[p_{\min}(\cdot \mid \mathcal{D} \cup \langle (\boldsymbol{\lambda}, z), \tilde{c} \rangle)]}{c(\boldsymbol{\lambda}, z)} \right]$$

Entropy Search for Multi-Fidelity Optimization [Klein et al. 2017]

- The entire algorithm iterates the following 2 steps until time is up:

- Select (λ, z) by maximizing:

$$u_{ES}(\lambda, z \mid \mathcal{D}) := \mathbb{E}_{p(\tilde{c} \mid (\lambda, z), \mathcal{D})} \left[\frac{\mathcal{H}[p_{\min}(\cdot \mid \mathcal{D})] - \mathcal{H}[p_{\min}(\cdot \mid \mathcal{D} \cup \langle (\lambda, z), \tilde{c} \rangle)]}{c(\lambda, z)} \right]$$

- Observe performance $f(\lambda, z)$ and cost $c(\lambda, z)$ and update models for f and c

Entropy Search for Multi-Fidelity Optimization [Klein et al. 2017]

- The entire algorithm iterates the following 2 steps until time is up:

- Select (λ, z) by maximizing:

$$u_{ES}(\lambda, z \mid \mathcal{D}) := \mathbb{E}_{p(\tilde{c} \mid (\lambda, z), \mathcal{D})} \left[\frac{\mathcal{H}[p_{\min}(\cdot \mid \mathcal{D})] - \mathcal{H}[p_{\min}(\cdot \mid \mathcal{D} \cup \langle (\lambda, z), \tilde{c} \rangle)]}{c(\lambda, z)} \right]$$

- Observe performance $f(\lambda, z)$ and cost $c(\lambda, z)$ and update models for f and c
- The algorithm originally focussed on data subsets and is therefore dubbed **Fabolas**: “Fast Bayesian Optimization on Large Datasets”

Entropy Search for Multi-Fidelity Optimization [Klein et al. 2017]

- The entire algorithm iterates the following 2 steps until time is up:

- Select (λ, z) by maximizing:

$$u_{ES}(\lambda, z \mid \mathcal{D}) := \mathbb{E}_{p(\tilde{c} \mid (\lambda, z), \mathcal{D})} \left[\frac{\mathcal{H}[p_{\min}(\cdot \mid \mathcal{D})] - \mathcal{H}[p_{\min}(\cdot \mid \mathcal{D} \cup \langle (\lambda, z), \tilde{c} \rangle)]}{c(\lambda, z)} \right]$$

- Observe performance $f(\lambda, z)$ and cost $c(\lambda, z)$ and update models for f and c
- The algorithm originally focussed on data subsets and is therefore dubbed **Fabolas**: “Fast Bayesian Optimization on Large Datasets”

Advantages

- ▶ Conceptually beautiful
- ▶ 1 000-fold speedups for optimizing SVMs on MNIST

Entropy Search for Multi-Fidelity Optimization [Klein et al. 2017]

- The entire algorithm iterates the following 2 steps until time is up:
 - Select (λ, z) by maximizing:

$$u_{ES}(\lambda, z \mid \mathcal{D}) := \mathbb{E}_{p(\tilde{c} \mid (\lambda, z), \mathcal{D})} \left[\frac{\mathcal{H}[p_{\min}(\cdot \mid \mathcal{D})] - \mathcal{H}[p_{\min}(\cdot \mid \mathcal{D} \cup \langle (\lambda, z), \tilde{c} \rangle)]}{c(\lambda, z)} \right]$$

- Observe performance $f(\lambda, z)$ and cost $c(\lambda, z)$ and update models for f and c
- The algorithm originally focussed on data subsets and is therefore dubbed **Fabolas**: “Fast Bayesian Optimization on Large Datasets”

Advantages

- Conceptually beautiful
- 1 000-fold speedups for optimizing SVMs on MNIST

Disadvantages

- Scalability of GPs is a big problem (limits size of initial design)
- Limited applicability of Gaussian processes

Questions to Answer for Yourself / Discuss with Friends

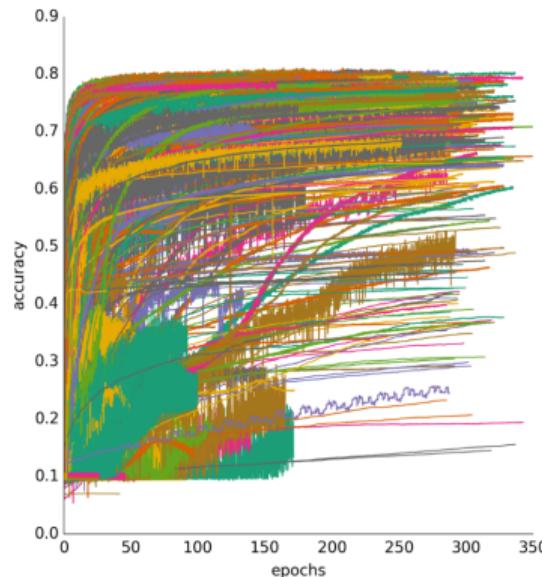
- Discussion. What kind of cost model would you use in Fabolas?
- Discussion. Could one use an acquisition function other than entropy search for Fabolas?

Speedup Techniques for Hyperparameter Optimization

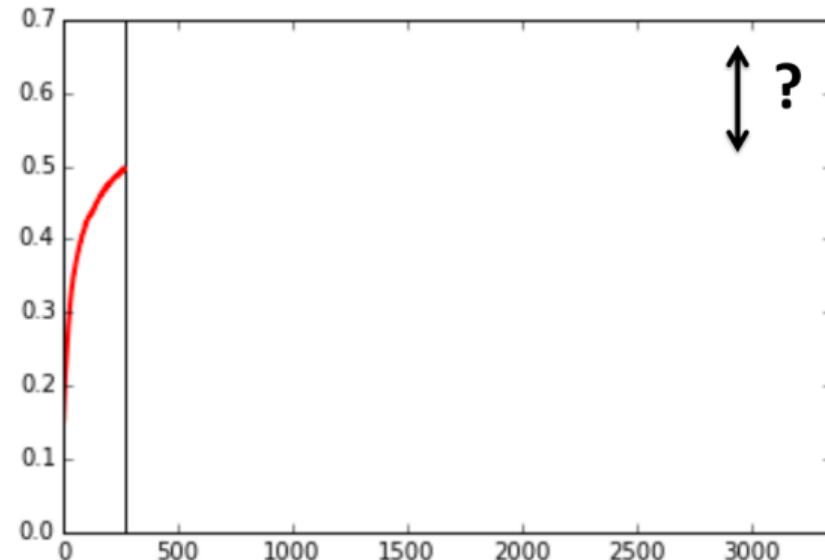
Predicting Learning Curves

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Learning Curves

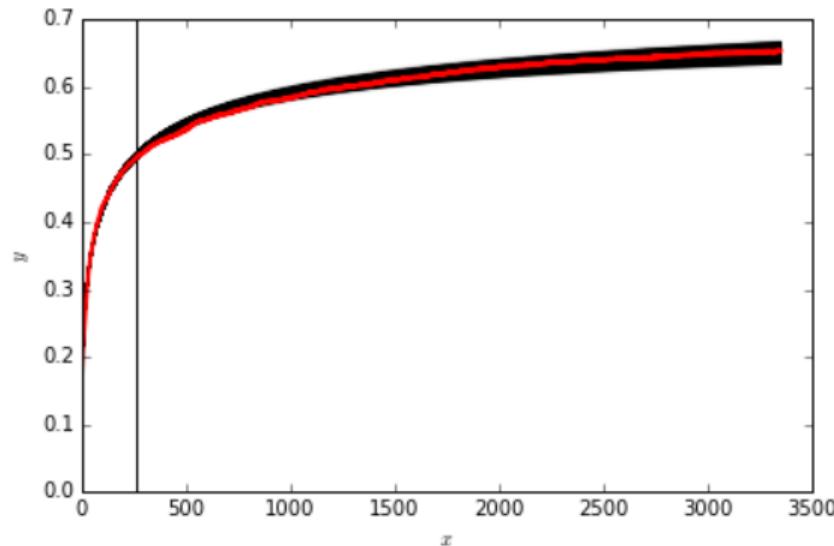


Learning Curve Predictions



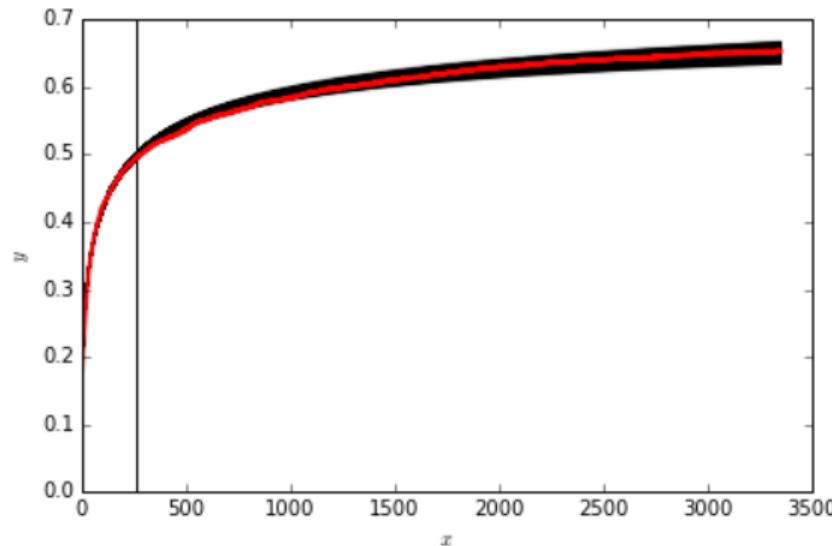
- ① Observe learning curve for the first n steps (here $n = 250$)

Learning Curve Predictions



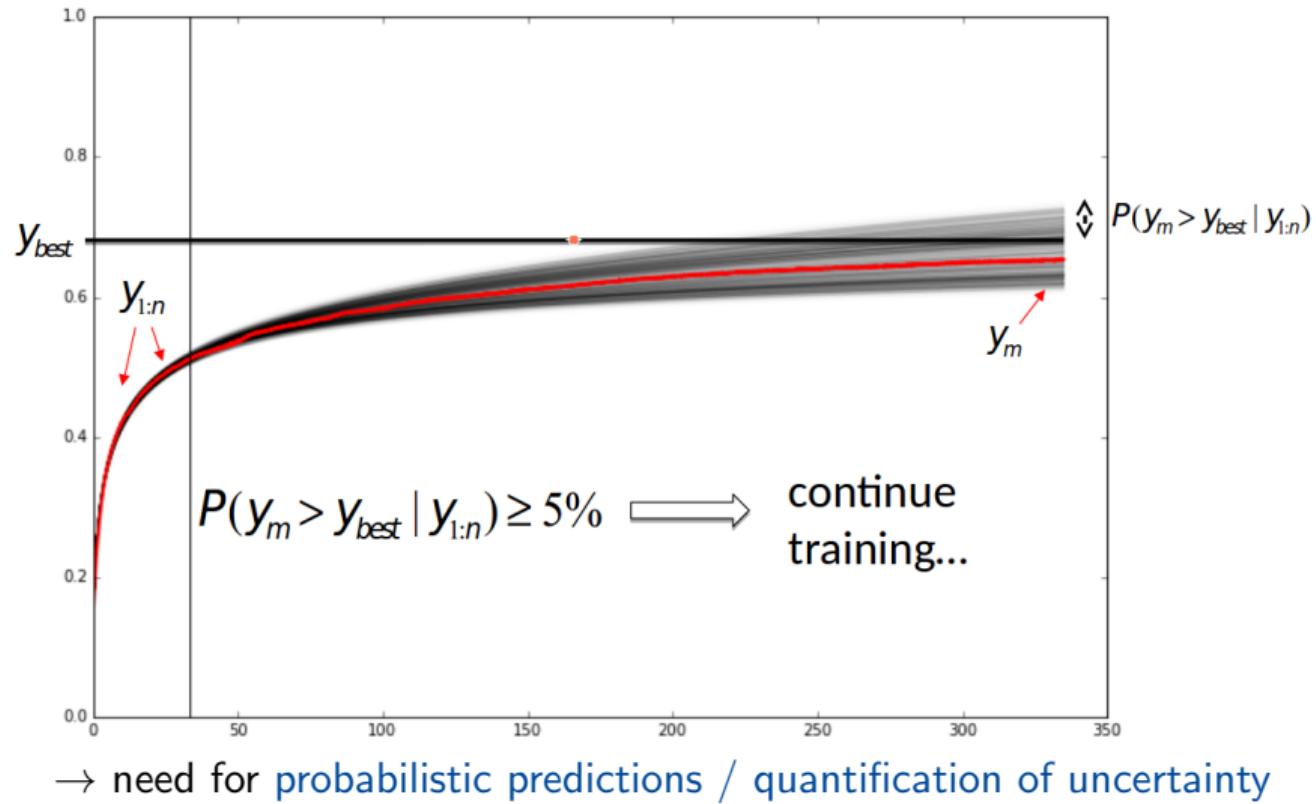
- ① Observe learning curve for the first n steps (here $n = 250$)
- ② **Extrapolation:** fit parametric model on partial learning curve to predict remaining learning curve

Learning Curve Predictions

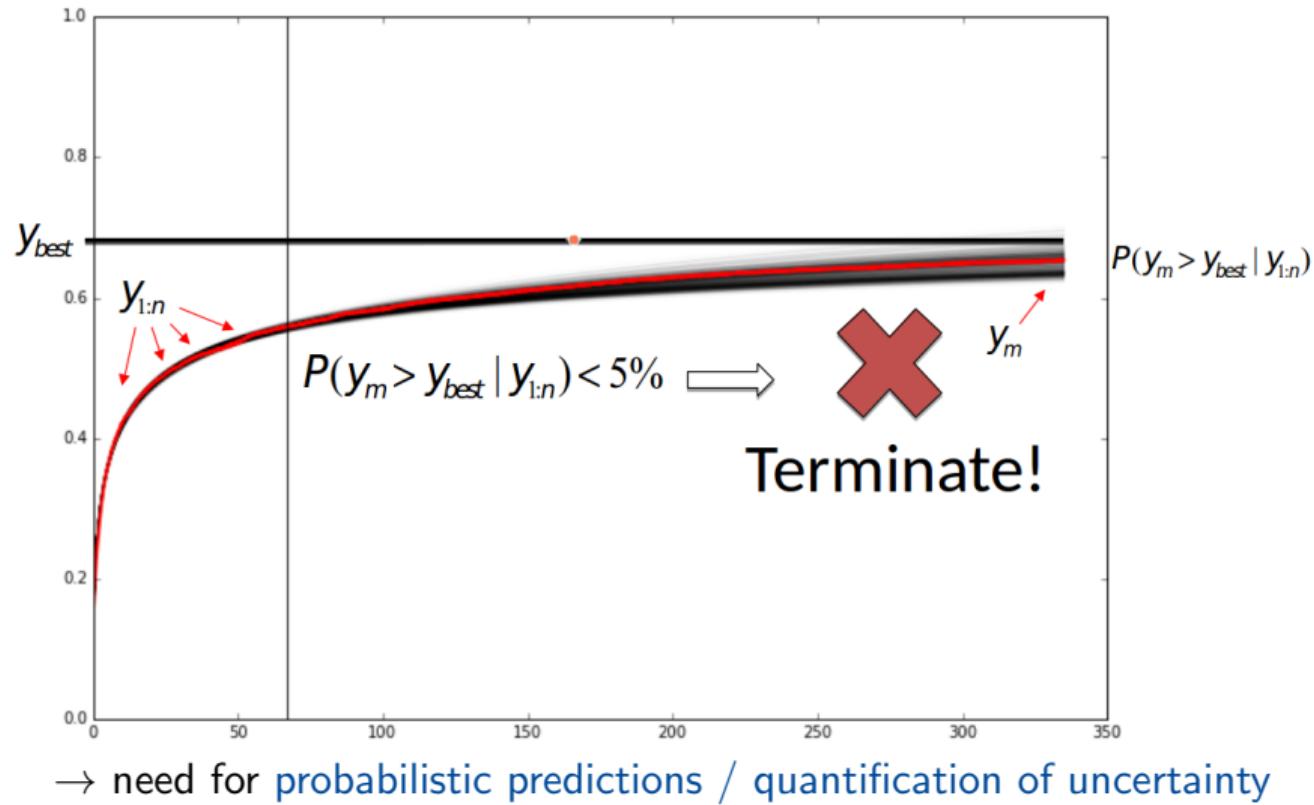


- ① Observe learning curve for the first n steps (here $n = 250$)
- ② **Extrapolation:** fit parametric model on partial learning curve to predict remaining learning curve
 - ▶ Various models can be used (see following slides)

Learning Curves: Early Termination



Learning Curves: Early Termination



Parametric Learning Curves [Domhan et al. 2015]

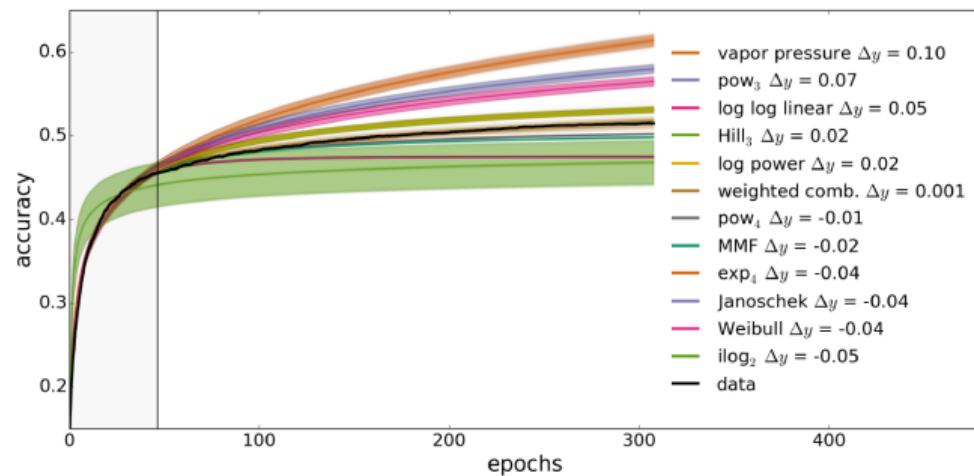
- Use a parametric model f_k with parameters θ to model performance at step t as:
 $y_t = f_k(t|\theta) + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

Parametric Learning Curves [Domhan et al. 2015]

- Use a parametric model f_k with parameters θ to model performance at step t as:
 $y_t = f_k(t|\theta) + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.
- Linear combination of $K = 11$ parametric types of models:
 $f_{comb}(t|\xi) = \sum_{k=1}^K w_k f_k(t|\theta_k)$, where $\xi = (w_1, \dots, w_K, \theta_1, \dots, \theta_K, \sigma^2)$

Parametric Learning Curves [Domhan et al. 2015]

- Use a parametric model f_k with parameters θ to model performance at step t as:
 $y_t = f_k(t|\theta) + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.
- Linear combination of $K = 11$ parametric types of models:
 $f_{comb}(t|\xi) = \sum_{k=1}^K w_k f_k(t|\theta_k)$, where $\xi = (w_1, \dots, w_K, \theta_1, \dots, \theta_K, \sigma^2)$

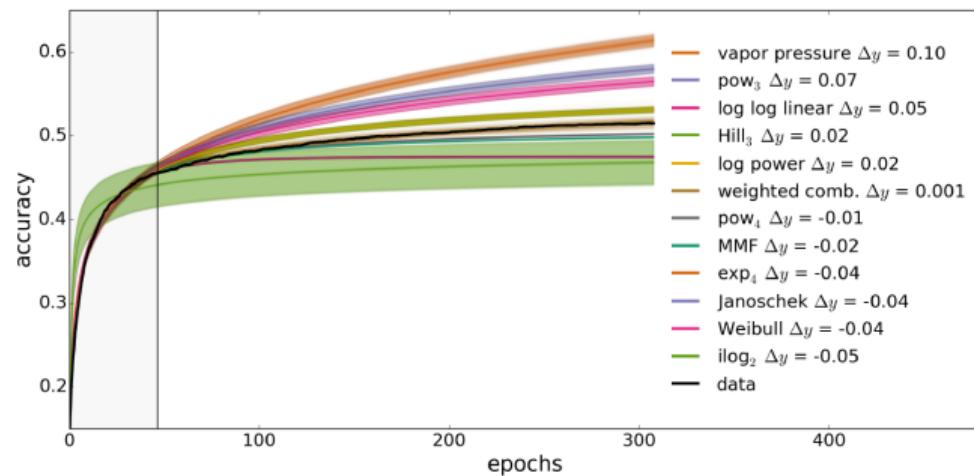


Reference name	Formula
vapor pressure	$\exp(a + \frac{b}{x} + c \log(x))$
pow ₃	$c - ax^{-\alpha}$
log log linear	$\log(a \log(x) + b)$
Hill ₃	$\frac{y_{\max} x^{\eta}}{\kappa^{\eta} + x^{\eta}}$
log power	$\frac{a}{1 + (\frac{x}{e^b})^c}$
pow ₄	$c - (ax + b)^{-\alpha}$
MMF	$\alpha - \frac{\alpha - \beta}{1 + (\kappa x)^{\delta}}$
exp ₄	$c - e^{-ax^{\alpha} + b}$
Janoschek	$\alpha - (\alpha - \beta)e^{-\kappa x^{\delta}}$
Weibull	$\alpha - (\alpha - \beta)e^{-(\kappa x)^{\delta}}$
ilog ₂	$c - \frac{a}{\log x}$

$K = 11$ parametric families for modelling learning curves

Parametric Learning Curves [Domhan et al. 2015]

- Use a parametric model f_k with parameters θ to model performance at step t as:
 $y_t = f_k(t|\theta) + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2)$.
- Linear combination of $K = 11$ parametric types of models:
 $f_{comb}(t|\xi) = \sum_{k=1}^K w_k f_k(t|\theta_k)$, where $\xi = (w_1, \dots, w_K, \theta_1, \dots, \theta_K, \sigma^2)$

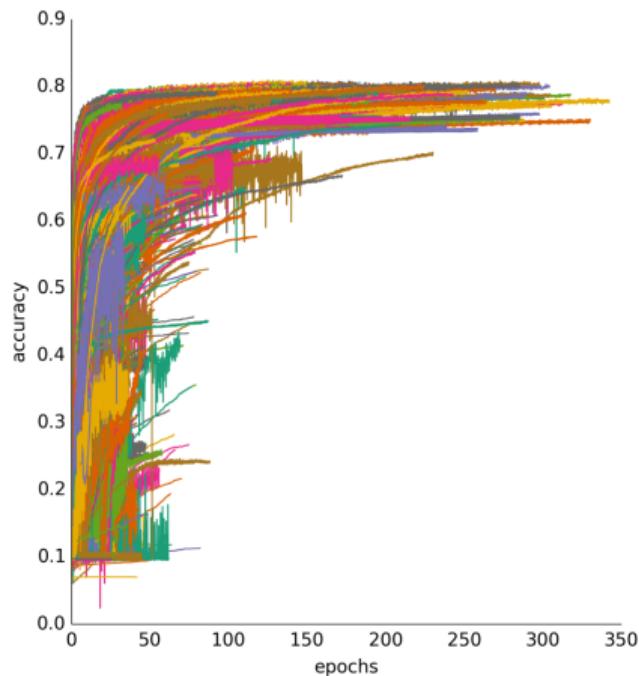
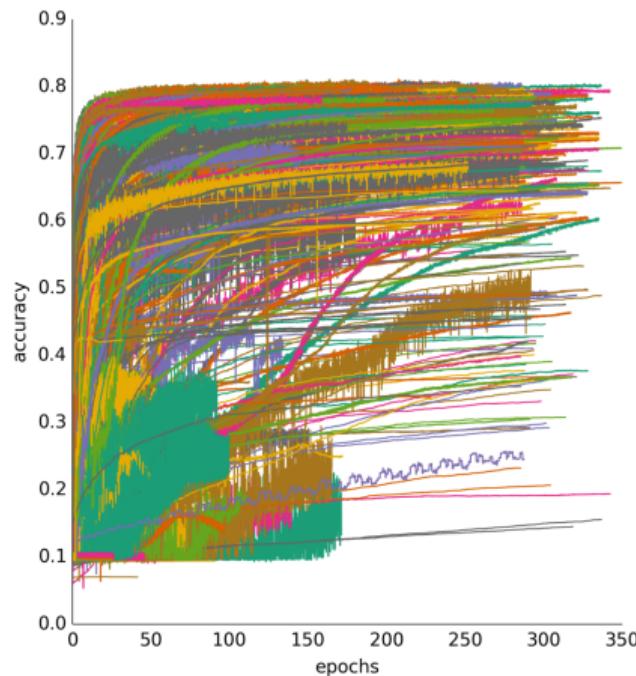


Reference name	Formula
vapor pressure	$\exp(a + \frac{b}{x} + c \log(x))$
pow ₃	$c - ax^{-\alpha}$
log log linear	$\log(a \log(x) + b)$
Hill ₃	$\frac{y_{\max} x^{\eta}}{\kappa^{\eta} + x^{\eta}}$
log power	$\frac{a}{1 + (\frac{x}{e^b})^c}$
pow ₄	$c - (ax + b)^{-\alpha}$
MMF	$\alpha - \frac{\alpha - \beta}{1 + (\kappa x)^{\delta}}$
exp ₄	$c - e^{-ax^{\alpha} + b}$
Janoschek	$\alpha - (\alpha - \beta)e^{-\kappa x^{\delta}}$
Weibull	$\alpha - (\alpha - \beta)e^{-(\kappa x)^{\delta}}$
ilog ₂	$c - \frac{a}{\log x}$

$K = 11$ parametric families for modelling learning curves

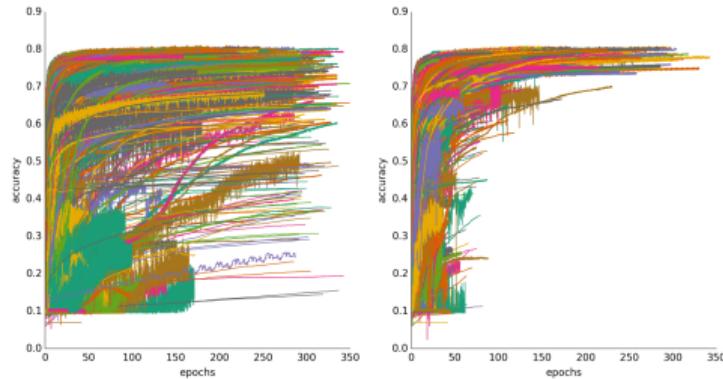
- Use Markov Chain Monte Carlo sampling of ξ to obtain uncertainties

Predictive Termination



All learning curves vs. learning curves with early termination

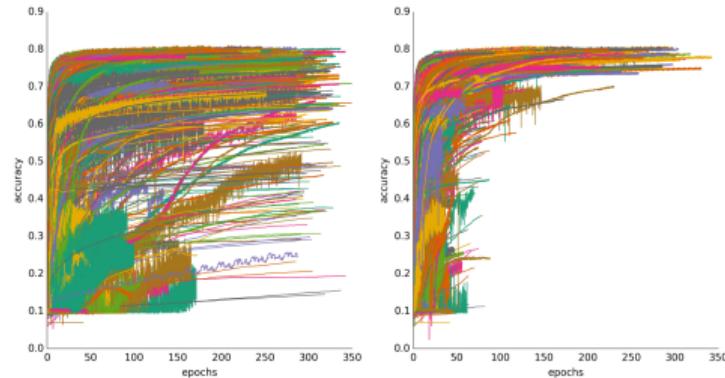
Predictive Termination



All learning curves vs. learning curves with early termination

- Disadvantages of this model?

Predictive Termination

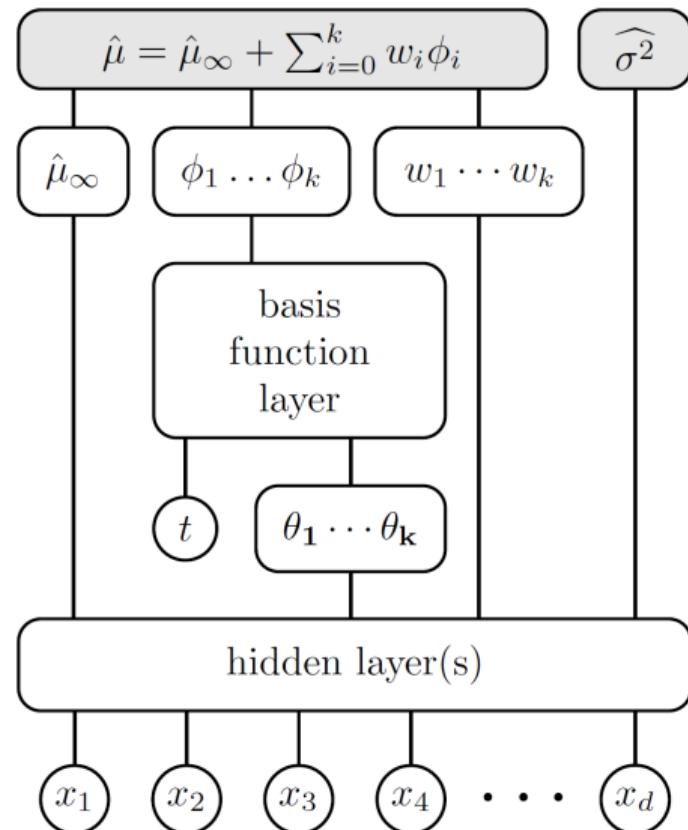


All learning curves vs. learning curves with early termination

- Disadvantages of this model?
 - ▶ Relies on manually-selected parametric families of curves
 - ▶ Does not take into account hyperparameters used
 - can't learn across hyperparameters

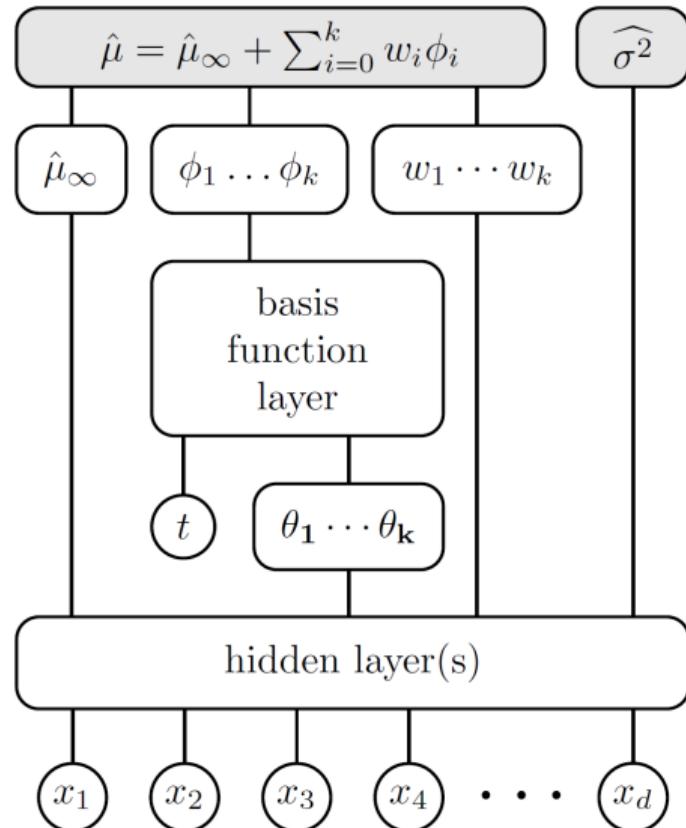
LC-Net [Klein et al. 2017]

- Make a layer out of the parametric learning curves by Domhan et al.
- Also support hyperparameters as inputs (in the figure denoted by x_1, \dots, x_d)



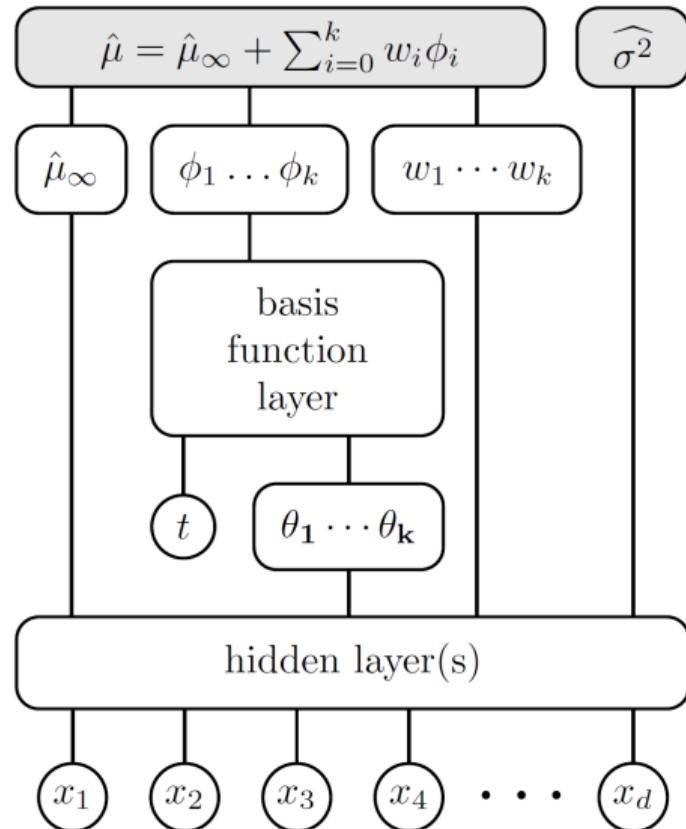
LC-Net [Klein et al. 2017]

- Make a layer out of the parametric learning curves by Domhan et al.
- Also support hyperparameters as inputs (in the figure denoted by x_1, \dots, x_d)
- Disadvantages of this model?



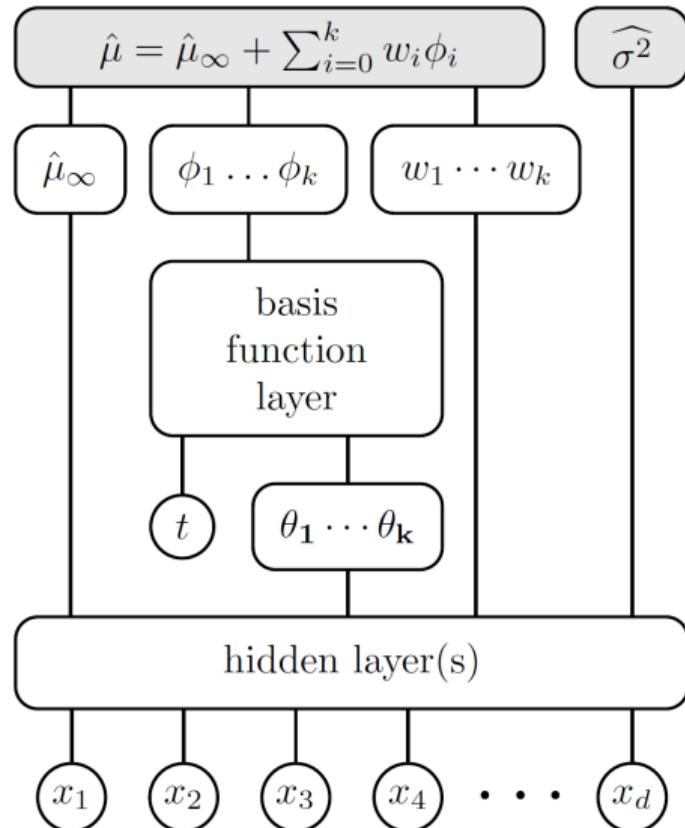
LC-Net [Klein et al. 2017]

- Make a layer out of the parametric learning curves by Domhan et al.
- Also support hyperparameters as inputs (in the figure denoted by x_1, \dots, x_d)
- Disadvantages of this model?
 - ▶ Relies on manually-selected parametric families of curves
 - ▶ Cannot quickly integrate new information from extending the current curve (or from new runs)



LC-Net [Klein et al. 2017]

- Make a layer out of the parametric learning curves by Domhan et al.
- Also support hyperparameters as inputs (in the figure denoted by x_1, \dots, x_d)
- Disadvantages of this model?
 - ▶ Relies on manually-selected parametric families of curves
 - ▶ Cannot quickly integrate new information from extending the current curve (or from new runs)
 - ▶ Also, the model is very hard to train

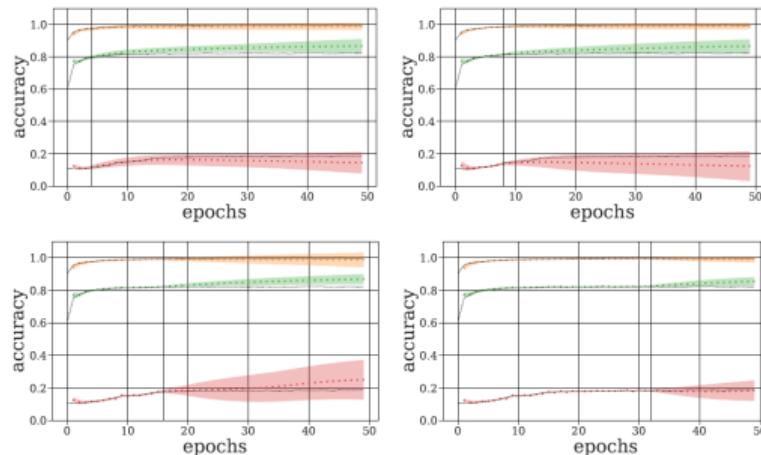


Sequence Models (e.g., Bayesian RNN) [Gargiani et al. 2019]

- Learning curves are **sequences**
 - ▶ Previous models don't treat them like this
 - ▶ We can use an RNN (in particular, an LSTM) to predict the next value from a given sequence
 - ▶ We can use variational dropout to obtain uncertainty estimates:

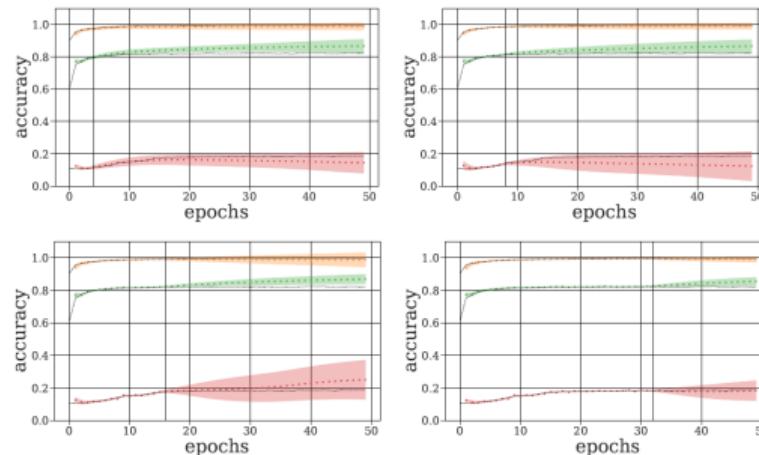
Sequence Models (e.g., Bayesian RNN) [Gargiani et al. 2019]

- Learning curves are **sequences**
 - ▶ Previous models don't treat them like this
 - ▶ We can use an RNN (in particular, an LSTM) to predict the next value from a given sequence
 - ▶ We can use variational dropout to obtain uncertainty estimates:



Sequence Models (e.g., Bayesian RNN) [Gargiani et al. 2019]

- Learning curves are **sequences**
 - ▶ Previous models don't treat them like this
 - ▶ We can use an RNN (in particular, an LSTM) to predict the next value from a given sequence
 - ▶ We can use variational dropout to obtain uncertainty estimates:



Note: we can also use a simpler model

- E.g., a random forest to map from a fixed-size window to the next value

Compare: Baker et al, 2017 [Baker et al. 2018]

- Idea: map from configurations (including architectural hyperparameters) and partial learning curves to the final performance
- Advantages
 - ▶ Much simpler idea than all the approaches just discussed:
no need to model the entire learning curve
 - ▶ Much easier to implement
- Disadvantage?

Compare: Baker et al, 2017 [Baker et al. 2018]

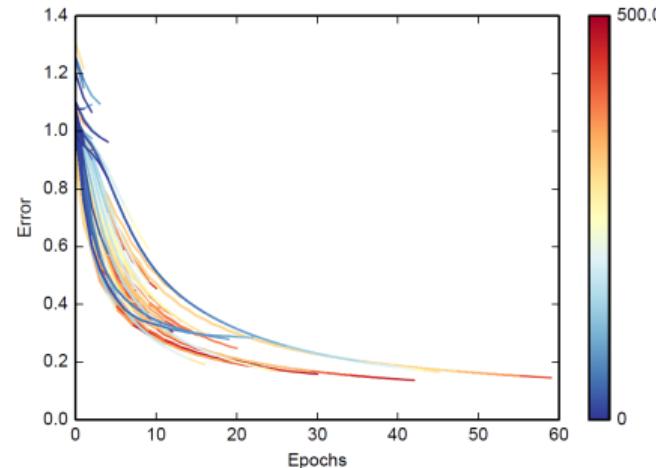
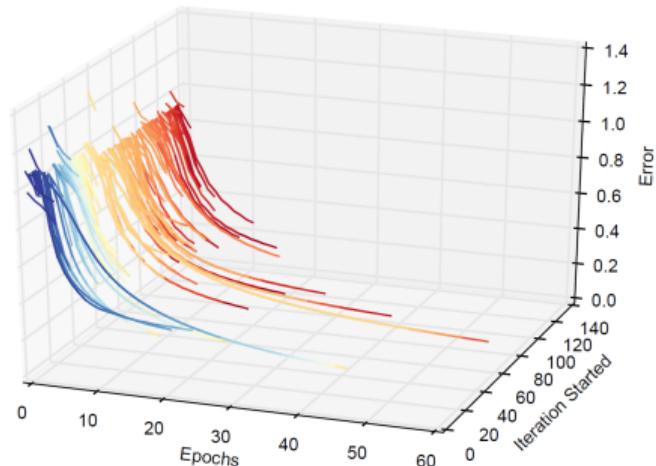
- Idea: map from configurations (including architectural hyperparameters) and partial learning curves to the final performance
- Advantages
 - ▶ Much simpler idea than all the approaches just discussed:
no need to model the entire learning curve
 - ▶ Much easier to implement
- Disadvantage? → requires many (e.g., 100) fully-evaluated learning curves as training data
 - ▶ After 100 full function evaluations we want to be pretty much converged in practice
 - ▶ But definitely helpful for speeding up RL

Freeze-Thaw Bayesian Optimization [Swersky et al. 2014]

- Use a Gaussian process with inputs λ and t ; special kernel for t
- For N configurations and T epochs each: $O(N^3t^3) \rightarrow$ approximation
- Iteratively: either extend existing configuration or try new one

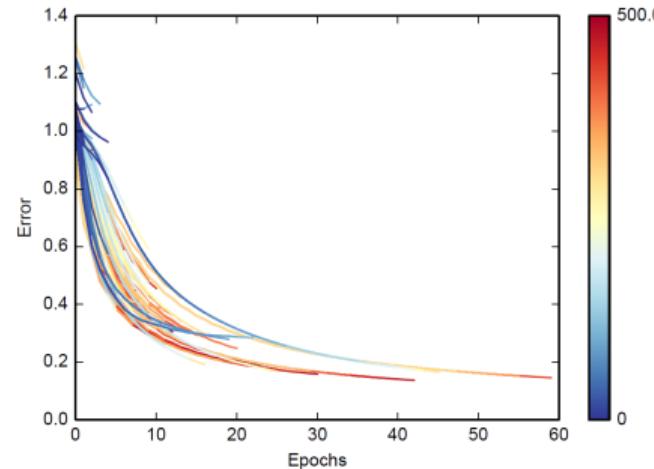
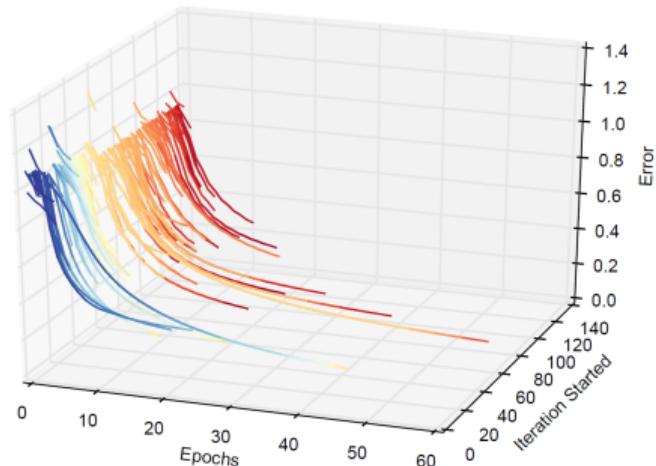
Freeze-Thaw Bayesian Optimization [Swersky et al. 2014]

- Use a Gaussian process with inputs λ and t ; special kernel for t
- For N configurations and T epochs each: $O(N^3t^3) \rightarrow$ approximation
- Iteratively: either extend existing configuration or try new one
- Result for probabilistic matrix factorization:



Freeze-Thaw Bayesian Optimization [Swersky et al. 2014]

- Use a Gaussian process with inputs λ and t ; special kernel for t
- For N configurations and T epochs each: $O(N^3t^3) \rightarrow$ approximation
- Iteratively: either extend existing configuration or try new one
- Result for probabilistic matrix factorization:



- Unfortunately, no results for DNNs; no code available

Questions to Answer for Yourself / Discuss with Friends

- **Repetition.** List all learning curve prediction methods you recall, along with their pros and cons.
- **Discussion.** Could predictive termination cut off evaluations early that would turn out to be the best?
- **Discussion.** How would you determine a learning curve prediction method's own hyperparameters (such as the 5% for early learning curve termination), in practice?
- **Discussion.** How could we exploit additional side information we gain about the learning curve, such as, e.g., statistics for the size of the gradients and activations over time?

Speedup Techniques for Hyperparameter Optimization

Success Stories and Practical Recommendations

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Large-scale Meta-Learning for HPO in Industry (Facebook)

- Facebook has an internal self-service machine learning (ML) system
 - ▶ Non-ML departments can integrate highly optimized ML models into their workflow
 - ▶ Hyperparameters of the ML models are optimized with Bayesian optimization

Large-scale Meta-Learning for HPO in Industry (Facebook)

- Facebook has an internal self-service machine learning (ML) system
 - ▶ Non-ML departments can integrate highly optimized ML models into their workflow
 - ▶ Hyperparameters of the ML models are optimized with Bayesian optimization
- Training data for the models changes over time
 - ▶ Hyperparameters are constantly re-optimized
 - ▶ For efficiency: meta-learning Bayesian optimization, as described in [Feurer et al. 2018]

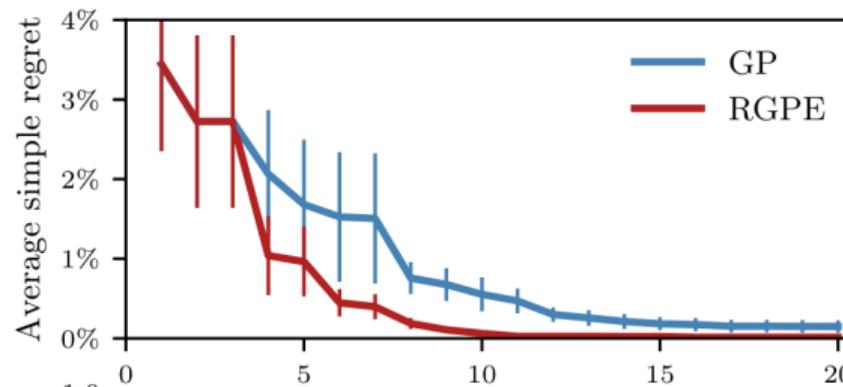
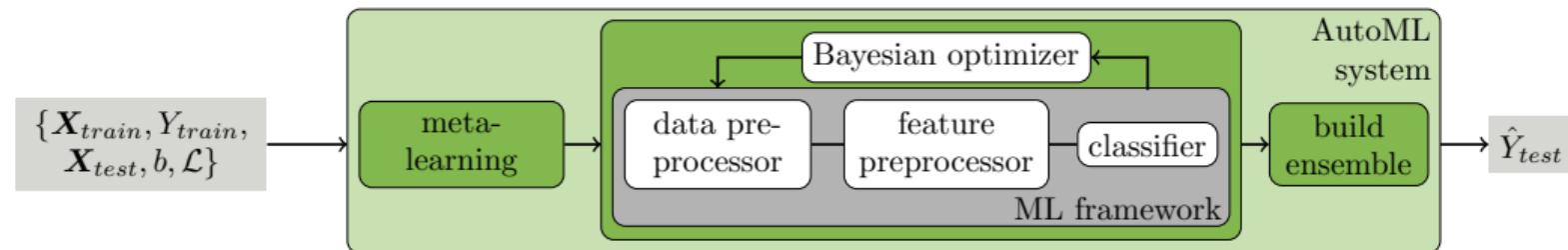


Figure: Bayesian optimization with meta-learning (RGPE) vs. vanilla Bayesian optimization (GP)

Auto-sklearn [Feurer et al. 2015]

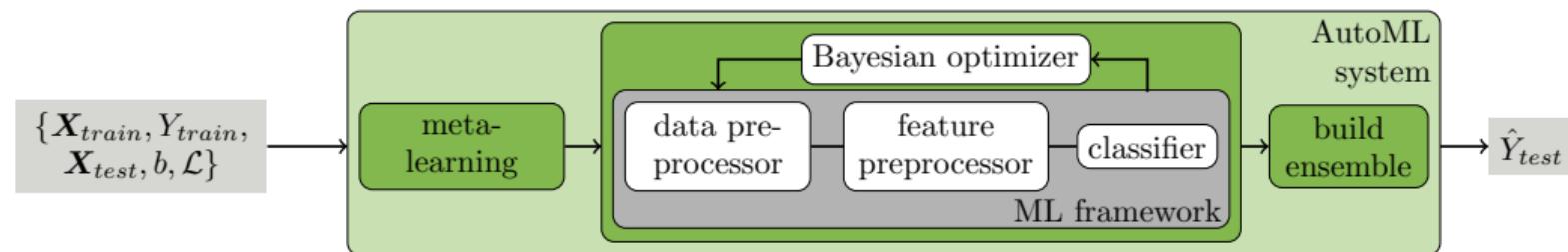
Extension of Auto-WEKA with focus on speed improvements and robustness:



- Uses meta-learning to warmstart Bayesian optimization
- Won the 1st AutoML challenge

Auto-sklearn [Feurer et al. 2015]

Extension of Auto-WEKA with focus on speed improvements and robustness:



- Uses meta-learning to warmstart Bayesian optimization
- Won the 1st AutoML challenge
- Open source (BSD) and trivial to use



```
>>> import autosklearn.classification  
>>> cls = autosklearn.classification.AutoSklearnClassifier()  
>>> cls.fit(X_train, y_train)  
>>> predictions = cls.predict(X_test)
```

Available at <https://automl.github.io/auto-sklearn>; frequently used in industry

BOHB [Falkner et al. 2018]

- Robust and efficient
- Only published in 2018, adopted by the community very quickly

Cited by 129



Scholar articles

[BOHB: Robust and efficient hyperparameter optimization at scale](#)
S Falkner, A Klein, F Hutter - arXiv preprint arXiv:1807.01774, 2018
[Cited by 129](#) [Related articles](#) [All 8 versions](#)

- Available at <https://github.com/automl/HpBandSter>



Unwatch ▾

23



Star

371

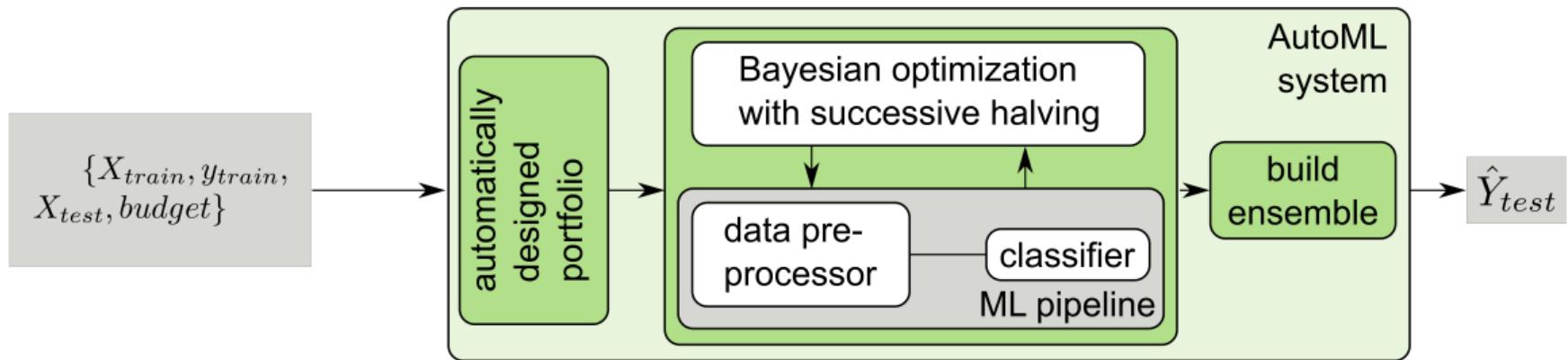


Fork

80

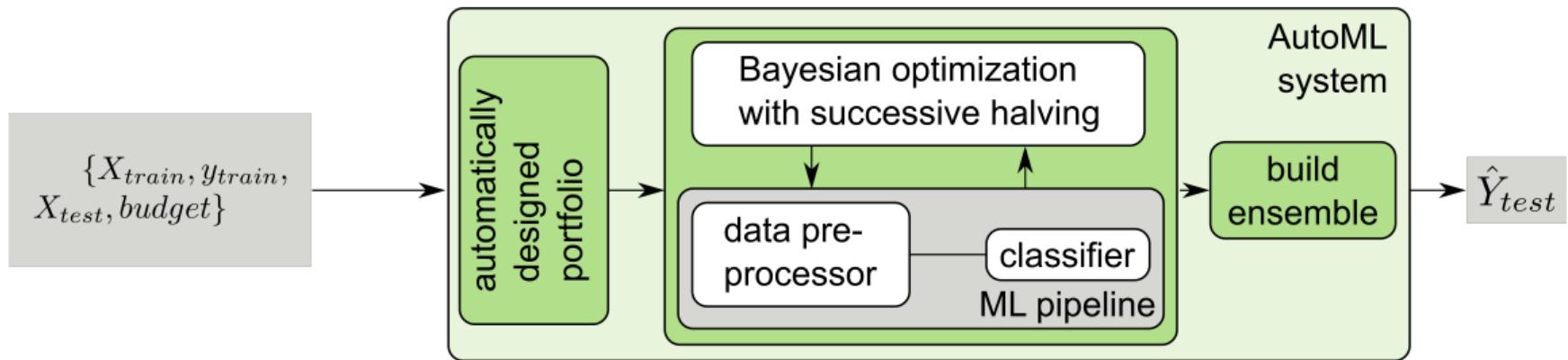
PoSH-Auto-sklearn [Feurer et al. 2018]

Idea: integrate warmstarting and a BOHB-like approach for Auto-sklearn



PoSH-Auto-sklearn [Feurer et al. 2018]

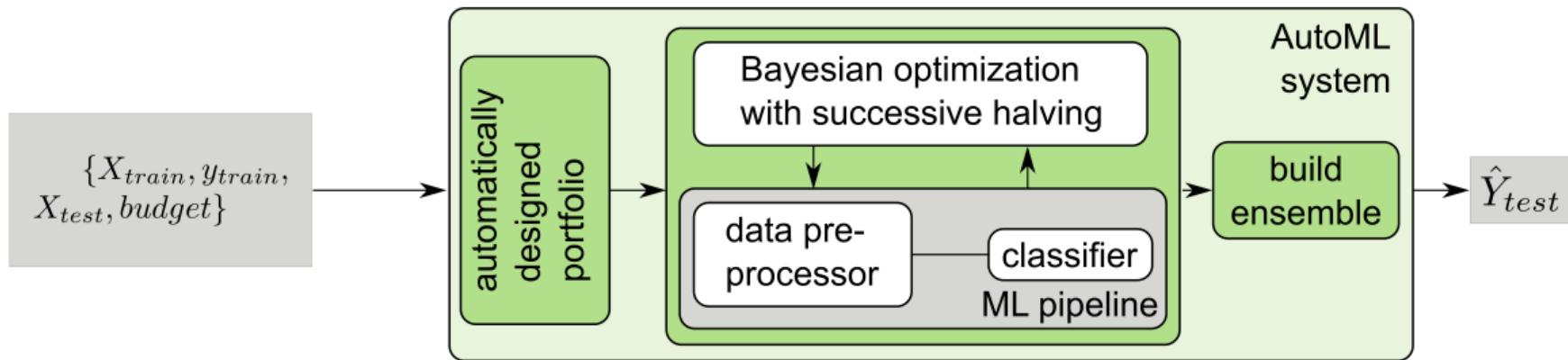
Idea: integrate warmstarting and a BOHB-like approach for Auto-sklearn



- Uses task-independent meta-learning to warmstart Bayesian optimization
 - ▶ Therefore, no need for (potentially unreliable) meta-features

PoSH-Auto-sklearn [Feurer et al. 2018]

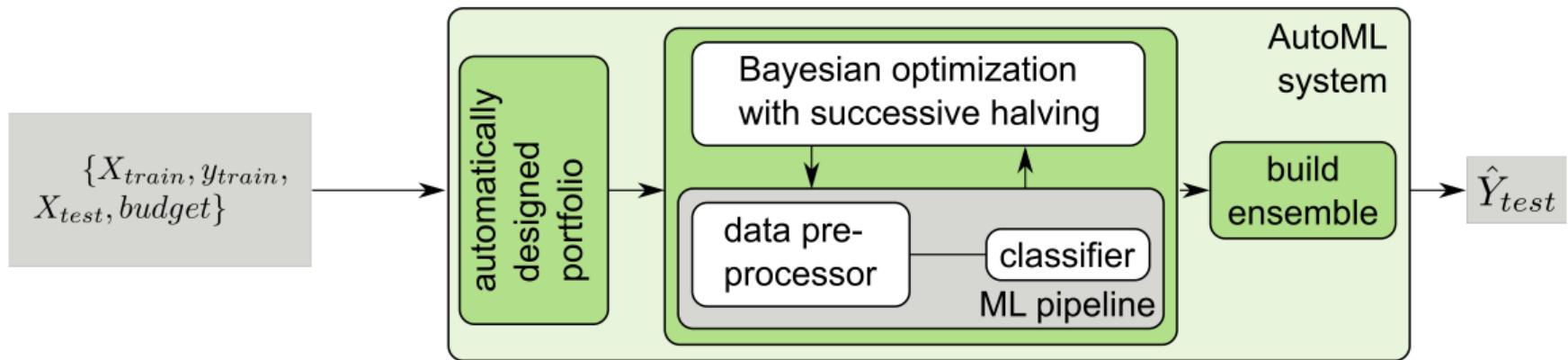
Idea: integrate warmstarting and a BOHB-like approach for Auto-sklearn



- Uses task-independent meta-learning to warmstart Bayesian optimization
 - ▶ Therefore, no need for (potentially unreliable) meta-features
- Uses successive halving to quickly go through proposed configurations
 - ▶ Therefore, scales better to larger datasets

PoSH-Auto-sklearn [Feurer et al. 2018]

Idea: integrate warmstarting and a BOHB-like approach for Auto-sklearn



- Uses task-independent meta-learning to warmstart Bayesian optimization
 - ▶ Therefore, no need for (potentially unreliable) meta-features
- Uses successive halving to quickly go through proposed configurations
 - ▶ Therefore, scales better to larger datasets
- Followed by BOHB-like approach (uses successive halving instead of Hyperband)
- Won the 2nd AutoML challenge

Auto-sklearn 2.0

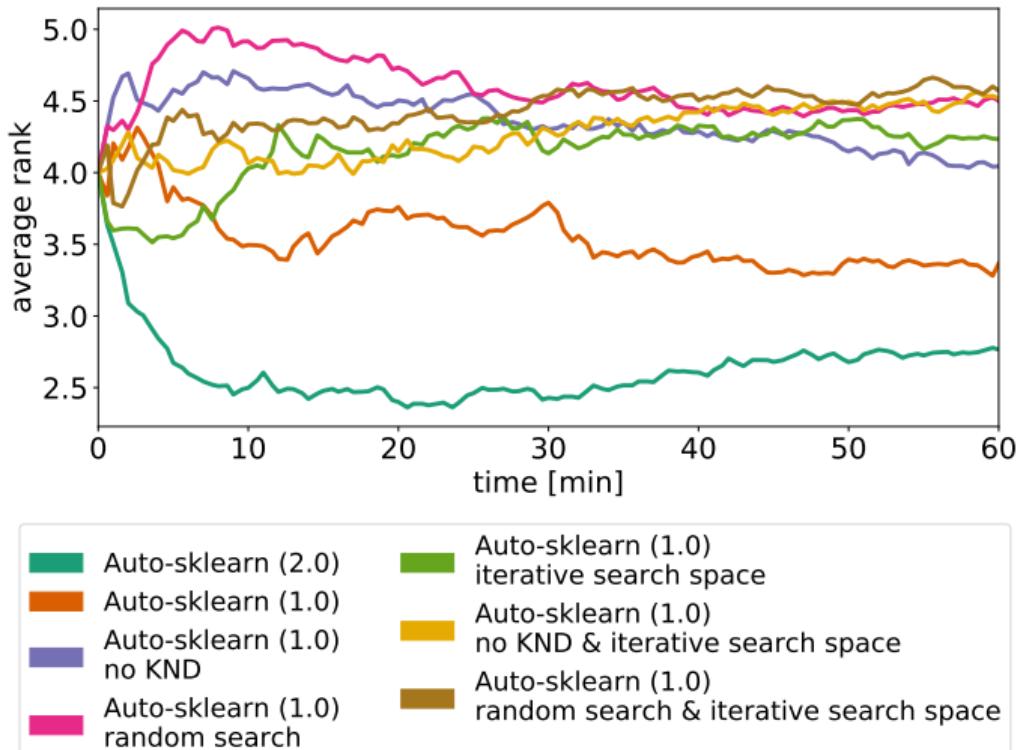
- Idea: automatically choose on a per-dataset basis
 - ▶ holdout or cross-validation
 - ▶ optimization on the full budget or optimization with successive halving

Auto-sklearn 2.0

- Idea: automatically choose on a per-dataset basis
 - ▶ holdout or cross-validation
 - ▶ optimization on the full budget or optimization with successive halving
- Can be done based on algorithm selection

Auto-sklearn 2.0

- Idea: automatically choose on a per-dataset basis
 - ▶ holdout or cross-validation
 - ▶ optimization on the full budget or optimization with successive halving
- Can be done based on algorithm selection
- Substantial improvements over Auto-sklearn 1.0
 - ▶ 5× reduction of average error
 - ▶ 6× speedup (same performance in 10 minutes as Auto-sklearn 1.0 in 1 hour)



Practical Recommendations Which HPO Method to Use [Feurer and Hutter. 2019]

- If multiple fidelities available: BOHB [Falkner et al. 2018]
- Otherwise
 - ▶ Low-dimensional continuous parameter space:
 - ★ GP-based BO, e.g., Spearmint [Snoek et al. 2012]
 - ▶ High-dimensional discrete parameter space:
 - ★ RF-based BO, e.g., SMAC [Hutter et al. 2011]
 - ▶ Purely continuous, cheap function evaluations:
 - ★ CMA-ES [Hansen et al. 2001]; evaluated for HPO by [Loshchilov and Hutter. 2016]

Practical Recommendations Which HPO Method to Use [Feurer and Hutter. 2019]

- If multiple fidelities available: BOHB [Falkner et al. 2018]
- Otherwise
 - ▶ Low-dimensional continuous parameter space:
 - ★ GP-based BO, e.g., Spearmint [Snoek et al. 2012]
 - ▶ High-dimensional discrete parameter space:
 - ★ RF-based BO, e.g., SMAC [Hutter et al. 2011]
 - ▶ Purely continuous, cheap function evaluations:
 - ★ CMA-ES [Hansen et al. 2001]; evaluated for HPO by [Loshchilov and Hutter. 2016]
- Just submitted: **DEHB** combines differential evolution and Hyperband and largely dominates BOHB. Especially good for high dimensions.

Questions to Answer for Yourself / Discuss with Friends

- Repetition. Discuss several success stories of speeding up Bayesian optimization.
- Repetition. What differs between Auto-sklearn 1.0 and Auto-sklearn 2.0?

Speedup Techniques for Hyperparameter Optimization

Further Reading

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Further Reading

Survey on hyperparameter optimization: [Feurer and Hutter 2019]

Multi-criteria Optimization

Introduction

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Introductory example I

Often we want to solve optimization problems concerning several goals.

General applications:

- Medicine: maximum effect, but minimum side effect of a drug.
- Finances: maximum return, but minimum risk of an equity portfolio.
- Production planning: maximum revenue, but minimum costs.
- Booking a hotel: maximum rating, but minimum costs.

In machine learning:

- Sparse models: maximum predictive performance, but minimal number of features.
- Fast models: maximum predictive performance, but short prediction time.
- ...

Introductory example II

Example:

Choose the best hotel to stay at by maximizing ratings subject to a maximum price per night.

Problems:

- The result depends on how we select the maximum price and usually returns different solutions for different maximum price values.
- We could also choose a minimum rating and optimize the price per night.
- The more objectives we optimize, the more difficult such a definition becomes.

Goal:

Find a more general approach to solve multi-criteria problems.

Introductory example III



Maritim Hotel München

4,0 ★★★★☆ (899)

WiFi Kostenlose WLAN

76 €



Marriott Hotel München

4,3 ★★★★★ (1.030)

28 % Rabatt

407 €

77 €



H+ Hotel München

4,2 ★★★★☆ (660)

WiFi Kostenlose WLAN

84 €



Hotel Vier Jahreszeiten

Kempinski Munich

4,6 ★★★★★ (1.025)

WiFi Kostenlose WLAN

278 €

When booking a hotel: find the hotel with

- minimum price per night (**costs**) and
- maximum user rating (**performance**).

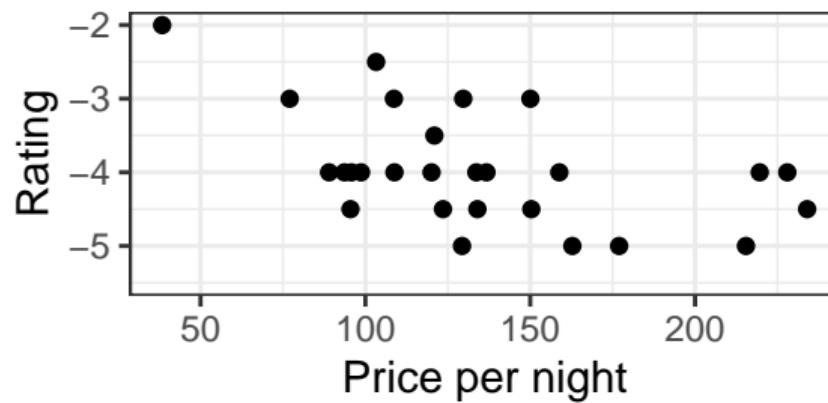
Since our standard is to minimize objectives, we minimize negative ratings.

Introductory example IV

The objectives often conflict with each other:

- Lower price → usually lower hotel rating.
- Better rating → usually higher price.

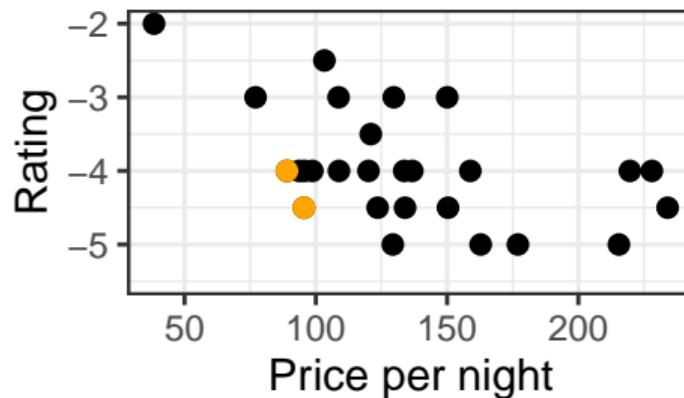
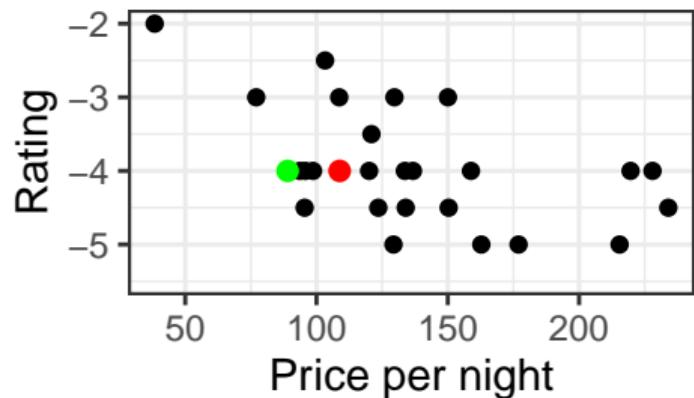
Example: (negative) average rating by hotel guests (1 - 5) vs. average price per night (excerpt).



Introductory example V

Often, objectives are not directly comparable as they are measured on different scales:

- Left: A hotel with rating 4 for 89 Euro ($c^{(1)} = (89, -4.0)$) would be preferred to a hotel for 108 Euro with the same rating ($c^{(2)} = (108, -4.0)$).
- Right: How to decide if $c^{(1)} = (89, -4.0)$ or $c^{(1)} = (95, -4.5)$ is preferred?
- How much is one *rating point* worth?



Definition: multi-criteria optimization problem

A **multi-criteria optimization problem** is defined by

$$\min_{\lambda \in \Lambda} c(\lambda) \Leftrightarrow \min_{\lambda \in \Lambda} (c_1(\lambda), c_2(\lambda), \dots, c_m(\lambda)),$$

with $\Lambda \subset \mathbb{R}^n$ and multi-criteria objective function $c : \Lambda \rightarrow \mathbb{R}^m$, $m \geq 2$.

- **Goal:** minimize multiple target functions simultaneously.
- $(c_1(\lambda), \dots, c_m(\lambda))^\top$ maps each candidate λ into the objective space \mathbb{R}^m .
- Often no clear best solution, as objectives are usually conflicting and we cannot totally order in \mathbb{R}^m .
- W.l.o.g. we always minimize.
- Alternative names: multi-criteria optimization, multi-objective optimization, Pareto optimization.

Pareto sets and Pareto optimality

Definition:

Given a multi-criteria optimization problem

$$\min_{\lambda \in \Lambda} (c_1(\lambda), \dots, c_m(\lambda)), \quad c_i : \Lambda \rightarrow \mathbb{R}.$$

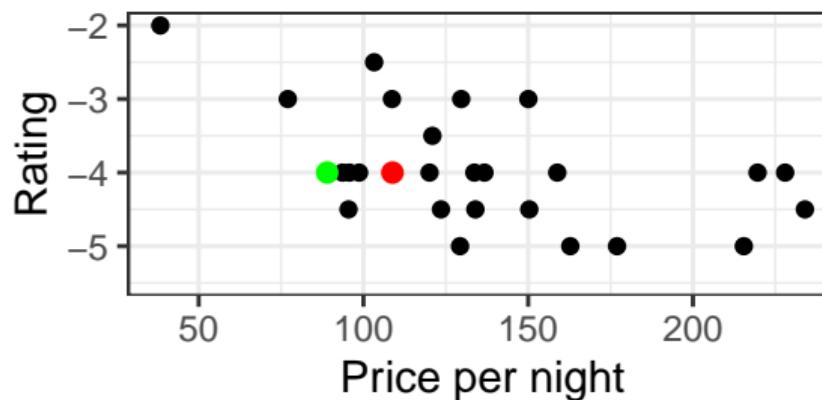
- A candidate $\lambda^{(1)}$ (**Pareto-**) **dominates** $\lambda^{(2)}$, if $c(\lambda^{(1)}) \prec c(\lambda^{(2)})$, i.e.
 - ① $c_i(\lambda^{(1)}) \leq c_i(\lambda^{(2)})$ for all $i \in \{1, 2, \dots, m\}$ and
 - ② $c_j(\lambda^{(1)}) < c_j(\lambda^{(2)})$ for at least one $j \in \{1, 2, \dots, m\}$
- A candidate λ^* that is not dominated by any other candidate is called **Pareto optimal**.
- The set of all Pareto optimal candidates is called **Pareto set**
$$\mathcal{P} := \{\lambda \in \Lambda \mid \nexists \tilde{\lambda} \text{ with } c(\tilde{\lambda}) \prec c(\lambda)\}$$
- $\mathcal{F} = c(\mathcal{P}) = \{c(\lambda) \mid \lambda \in \mathcal{P}\}$ is called **Pareto front**.

How to define optimality? I

Let $c = (\text{price}, -\text{rating})$. For some cases it is *clear* which point is the better one:

- The candidate $c^{(1)} = (89, -4.0)$ dominates $c^{(2)} = (108, -4.0)$: $c^{(1)}$ is not worse in any dimension and is better in one dimension. Therefore, $c^{(2)}$ gets **dominated** by $c^{(1)}$

$$c^{(2)} \prec c^{(1)}.$$

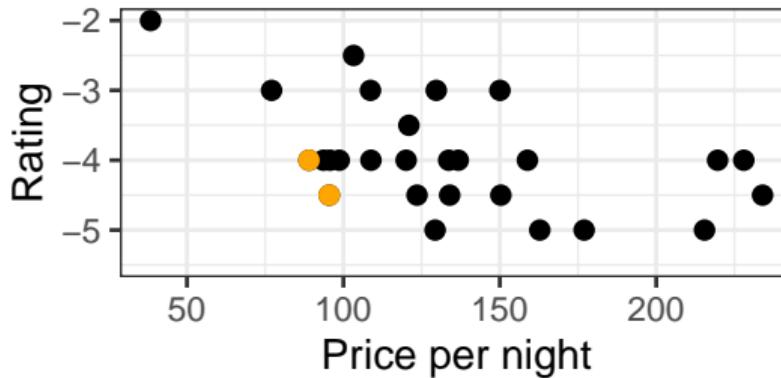


How to define optimality? II

For the points $c^{(1)} = (89, -4.0)$ and $c^{(2)} = (95, -4.5)$ we cannot say which one is better.

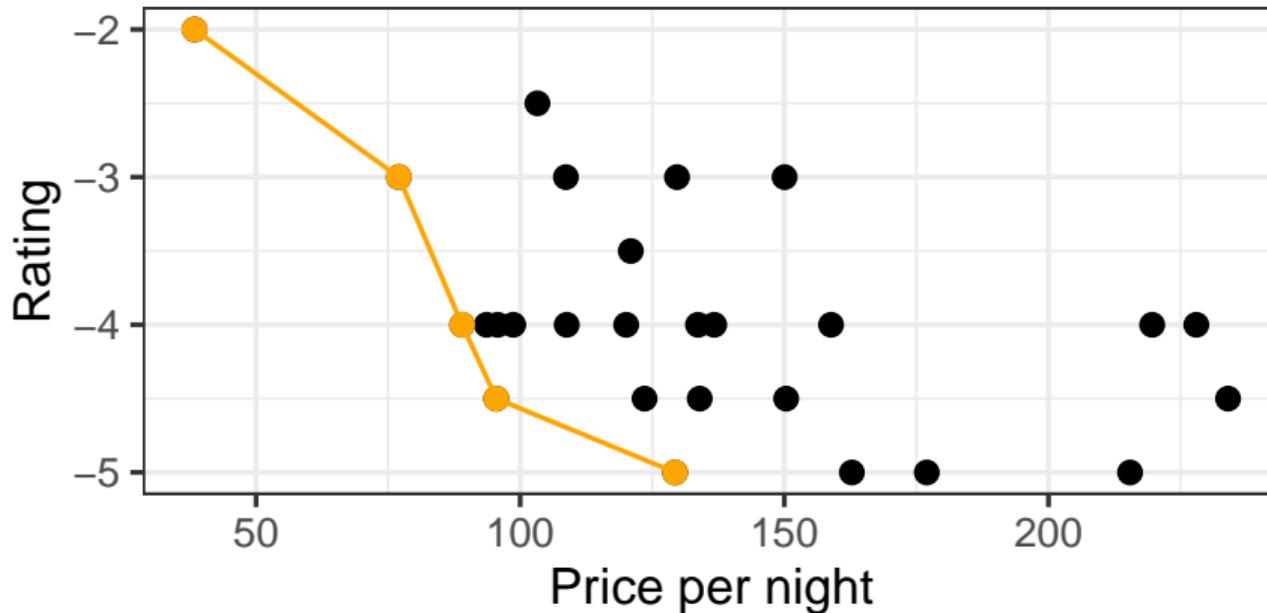
- We define the points as **equivalent** and write

$$c^{(1)} \not\prec c^{(2)} \text{ and } c^{(2)} \not\prec c^{(1)}.$$



How to define optimality? III

- The set of all equivalent points that are not dominated by another point is called the **Pareto front**.

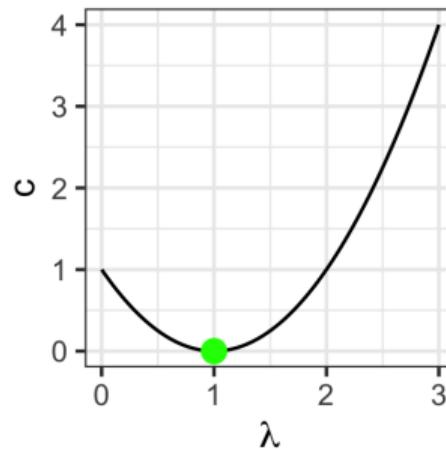


Example: One objective function

We consider the minimization problem

$$\min_{\lambda} c(\lambda) = (\lambda - 1)^2, \quad 0 \leq \lambda \leq 3.$$

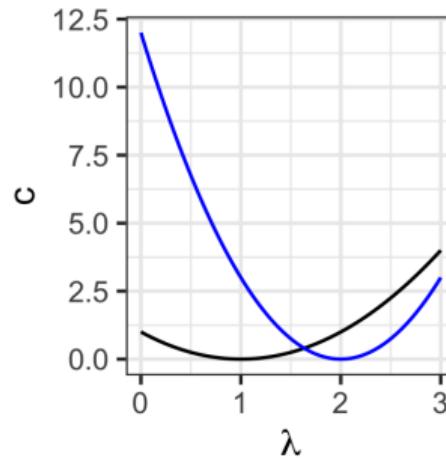
The optimum is at $\lambda^* = 1$.



Example: Two target functions I

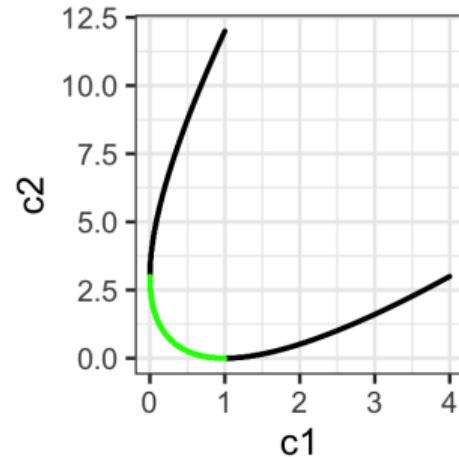
We extend the above problem to two objective functions $c_1(\lambda) = (\lambda - 1)^2$ and $c_2(\lambda) = 3(\lambda - 2)^2$, thus

$$\min_{\lambda} c(\lambda) = (c_1(\lambda), c_2(\lambda)), \quad 0 \leq \lambda \leq 3.$$



Example: Two target functions II

We consider the functions in the objective function space $c(\Lambda)$ by drawing the objective function values $(c_1(\lambda), c_2(\lambda))$ for all $0 \leq \lambda \leq 3$.



The Pareto front is shown in green. The Pareto front cannot be *left* without getting worse in at least one objective function.

A-priori vs. A-posteriori

- The Pareto set is a set of equally optimal solutions.
- In many applications one is often interested in a **single** optimal solution.
- Without further information no unambiguous optimal solution can be determined.
→ The decision must be based on other criteria.

There are two possible approaches:

- **A-priori approach:** User preferences are considered **before** the optimization process
- **A-posteriori approach:** User preferences are considered **after** the optimization process

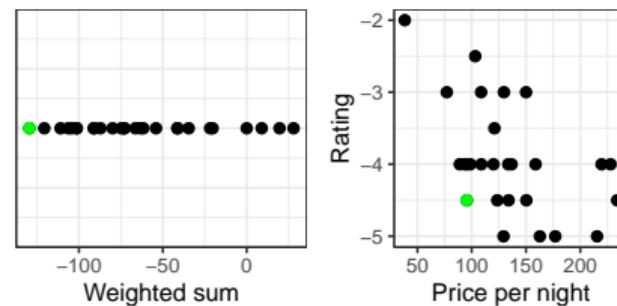
A-priori procedure I

Example: Weighted total

Prior knowledge: One rating point is worth 50 Euro to a customer.

→ We optimize the weighted sum:

$$\min_{\text{Hotel}} (\text{Price} / \text{Night}) - 50 \cdot \text{Rating}$$



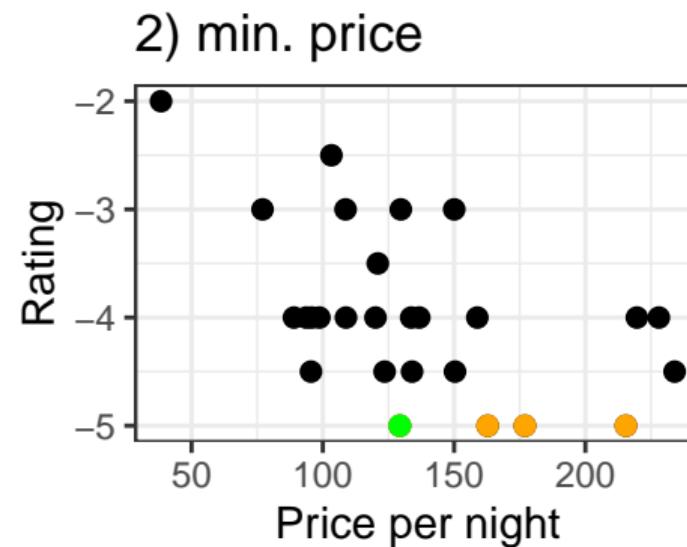
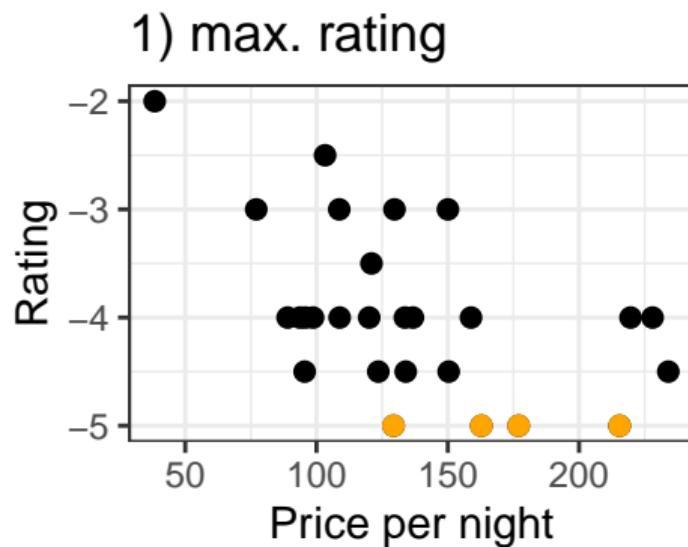
Alternative a weighted sum: $\min_{\lambda \in \Lambda} \sum_{i=1}^m w_i c_i(\lambda)$ with $w_i \geq 0$

A-priori procedure II

Example: Lexicographic method

Prior knowledge: Customer prioritizes rating over price.

→ Optimize target functions one after the other.



A-priori procedure III

A-priori approach: Lexicographic method

$$c_1^* = \min_{\lambda \in \Lambda} c_1(\lambda)$$

$$c_2^* = \min_{\lambda \in \{\lambda \mid c_1(\lambda) = c_1^*\}} c_2(\lambda)$$

$$c_3^* = \min_{\lambda \in \{\lambda \mid c_1(\lambda) = c_1^* \wedge c_2(\lambda) = c_2^*\}} c_3(\lambda)$$

⋮

But: Different sequences provide different solutions.

A-priori procedure IV

Summary a-priori approach:

- Implicit assumption: Single-objective optimization is easy.
- Only one solution is obtained, which depends on a-priori weights, order, etc.
- Several solutions can be obtained if weights, order, etc. are systematically varied.
- Usually not all non-dominated candidates can be found by these methods.

A-posteriori procedure I

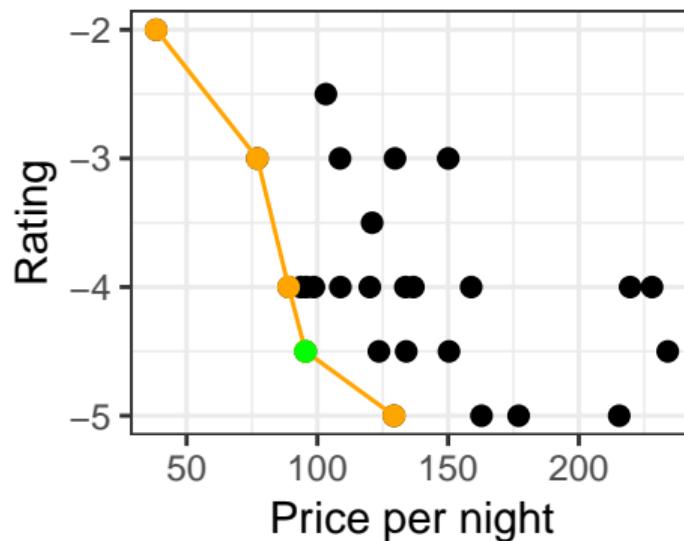
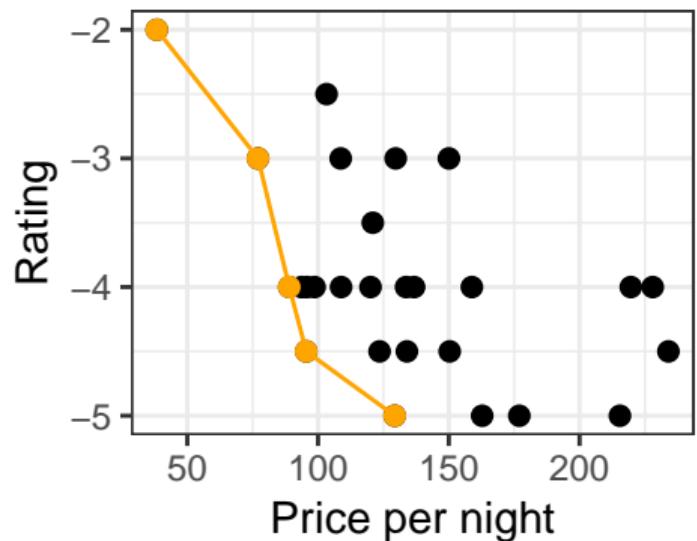
A-posteriori methods try to

- find the set of **all** optimal candidates (the Pareto set),
- select (if necessary) an optimal candidate based on prior knowledge or individual preferences.
- Implicit assumption: Specifying your hidden preferences / making a selection from a pool of candidates is easier, if you see the non-dominated solutions.

A-posteriori methods are therefore the more generic approach to solving a multi-criteria optimization problem.

A-posteriori procedure II

Example: A user is displayed all Pareto optimal hotels (left) and chooses an optimal candidate (right) based on his hidden preferences or additional criteria (e.g. location of the hotel).

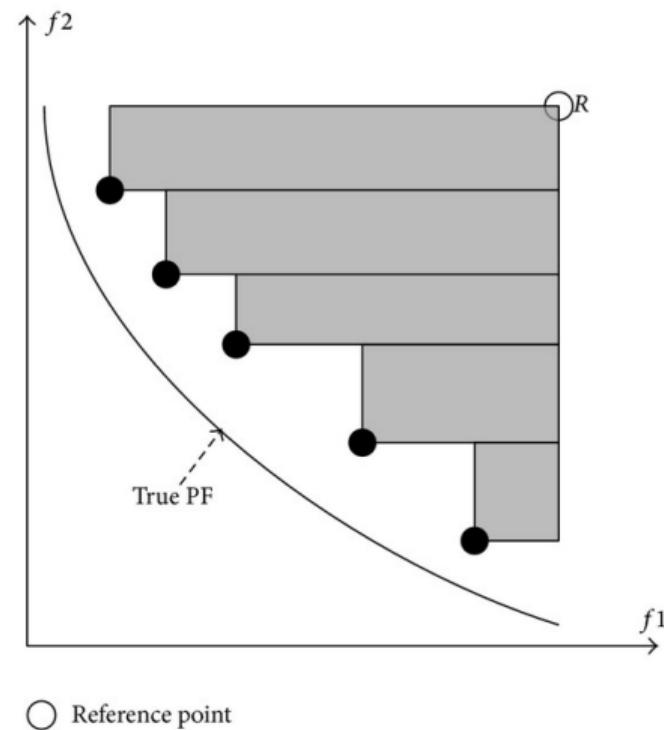


Evaluation of solutions I

A common metric for evaluating the performance of a set of candidates $\mathcal{P} \subset \Lambda$ is the **dominated hypervolume**

$$S(\mathcal{P}, R) = \Lambda \left(\bigcup_{\tilde{\lambda} \in \mathcal{P}} \left\{ \lambda | \tilde{\lambda} \prec \lambda \prec R \right\} \right),$$

where Λ is the Lebesgue measure.



Evaluation of solutions II

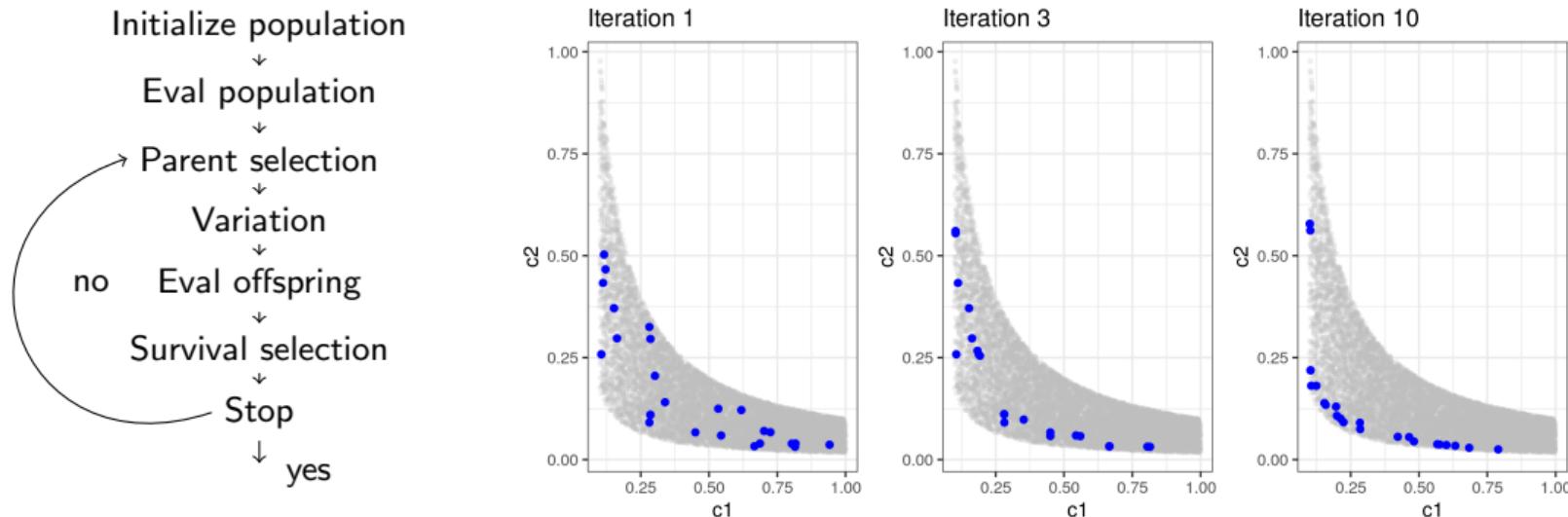
- HV is calculated w.r.t the reference point R , which often reflects in each component the natural maximum of the respective objective – if possible
- The dominated hypervolume is also often called **S-Metric**.
- Computation of HV scales exponentially in the number of objective functions $\mathcal{O}(n^{m-1})$.
- Fast approximations exist for small values of m and especially for machine learning applications we rarely optimize $m > 3$ objectives.

Multi-criteria Optimization Evolutionary Approaches

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

A-posteriori methods and evolutionary algorithms I

Evolutionary multi-objective algorithms (EMOAs) evolve a diverse population over time to approximate the Pareto front.



A-posteriori methods and evolutionary algorithms II

Algorithm 1 Basic EA template loop

```
1: Init and eval population  $\mathcal{P}_0 \subset \Lambda$  with  $|\mathcal{P}| = \mu$ 
2:  $t \leftarrow 0$ 
3: repeat
4:   Select parents and generate offspring  $\mathcal{Q}_t$  with  $|\mathcal{Q}_t| = \lambda$ 
5:   Select  $\mu$  survivors  $\mathcal{P}_{t+1}$ 
6:    $t \leftarrow t + 1$ 
7: until Stop criterion fulfilled
```

- Note that (as in the EA lecture unit) we are using somewhat non-standard notation here.
- Nearly all steps in the above template work also for EMOAs but both parent and survival selection are now less obvious. How do we rank under multiple objectives?

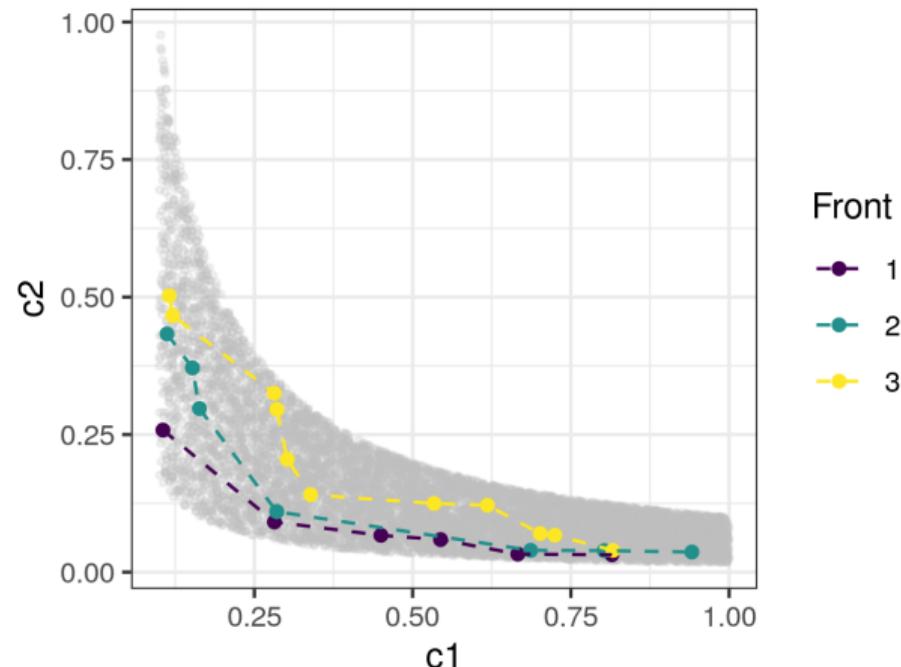
The **non-dominated sorting genetic algorithm (NSGA-II)** was published by [Dep et al. 2002].

- Follows a $(\mu + \lambda)$ strategy.
- All previously discussed variation strategies can be used; the original paper uses tournament selection, polynomial mutation and simulated binary crossover.
- Parent and survival selection rank candidates by
 - ① **Non-dominated sorting** as main criterion
 - ② **Crowding distance assignment** as tie breaker

NSGA-II: non-dominated sorting I

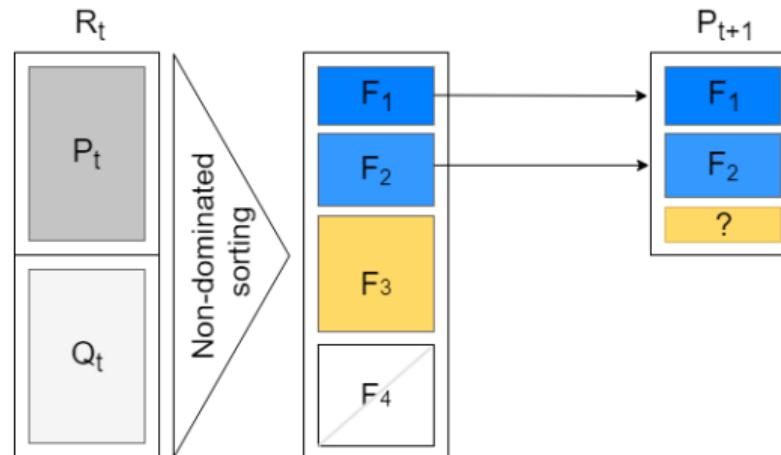
NDS partitions an objective space set into fronts $\mathcal{F}_1 \prec \mathcal{F}_2 \prec \mathcal{F}_3 \prec \dots$

- \mathcal{F}_1 is non-dominated, each $\lambda \in \mathcal{F}_2$ is dominated, but only by points in \mathcal{F}_1 , each $\lambda \in \mathcal{F}_3$ is dominated, but only by points in \mathcal{F}_1 and \mathcal{F}_2 , and so on.
- We can easily compute the partitioning by computing all non-dominated points \mathcal{F}_1 , removing them, then computing the next layer of non-dominated points \mathcal{F}_2 , and so on.



NSGA-II: non-dominated sorting II

How does survival selection now work? We fill μ places one by one with $\mathcal{F}_1, \mathcal{F}_2, \dots$ until a front can no longer **fully** survive (here: \mathcal{F}_3).

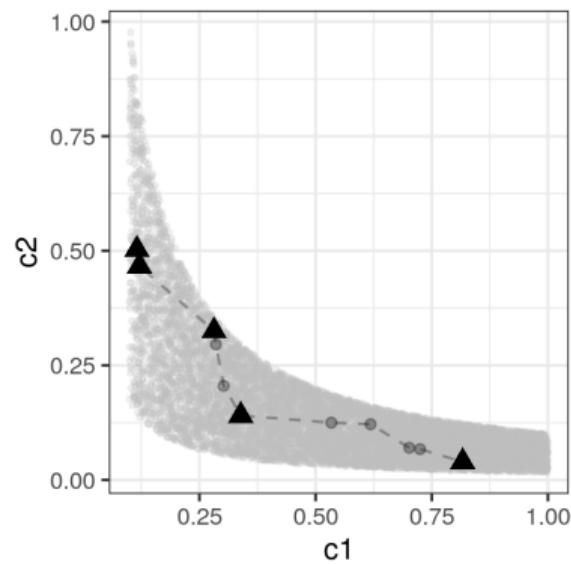
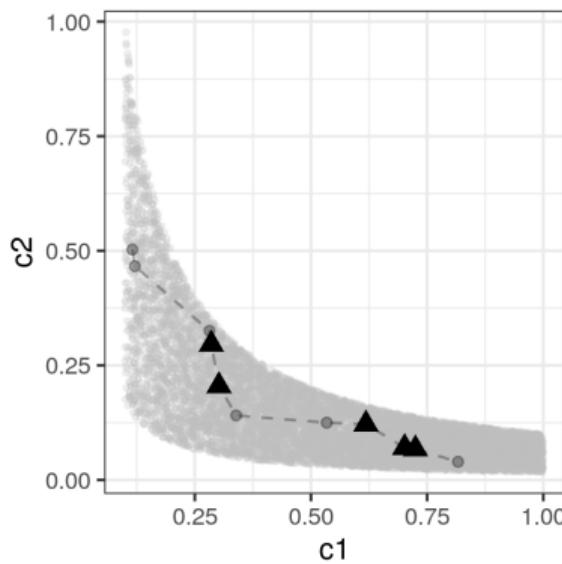


Which individuals survive from \mathcal{F}_3 ? → **crowding sort**

NB: the same principle to rank individuals is applied in tournament selection in parent selection.

NSGA-II: crowding distance I

Idea: Add *good* representatives of front \mathcal{F}_3 , define this as points of "low density" in c-space.



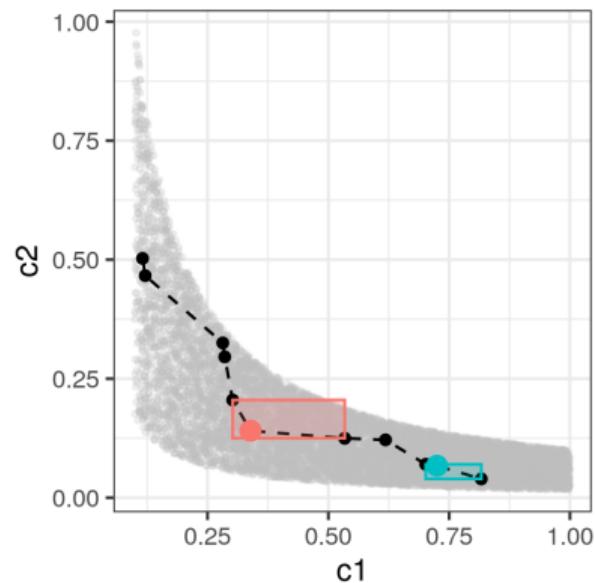
Left: Not good, points very close together. Right: better.

NSGA-II: crowding distance II

For each objective c_j

- Sort points by c_j
- Normalize scores to $[0,1]$
- Assign border points (which have score 0 or 1) a CD of ∞ (they should always be selected, if possible)
- Each point gets a distance score, which is the distance between its 2 next-neighbors w.r.t. the sorting of c_j

For each point, all of its m distance scores are summed up (or averaged) and points are ranked w.r.t. to this overall score.



Red: Point with high CD. Blue: Low CD.

Selection criteria: contribution to the hypervolume I

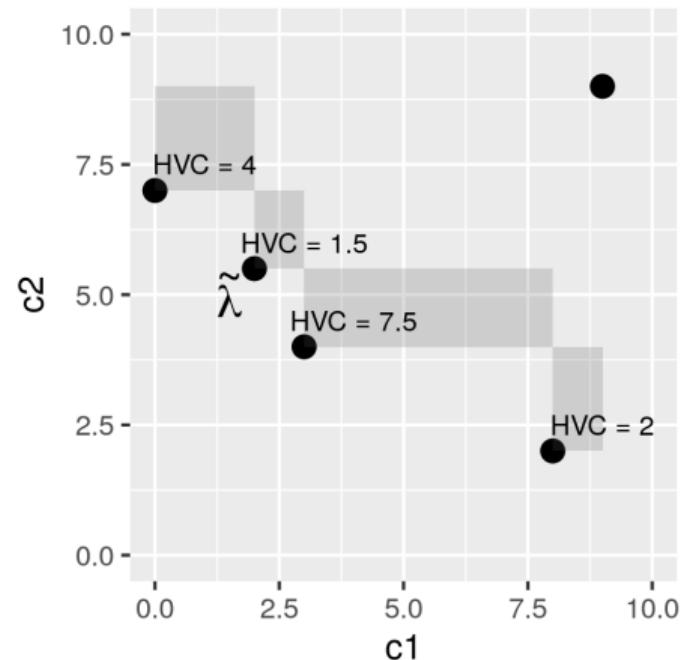
SMS-EMOA

(S-Metric-Selection-EMOA) [Beume et al. 2007]

is a $(\mu + 1)$ EMOA and evaluates fitness of an individual $\lambda \in \mathcal{P} \subset \Lambda$ based on its contribution to the dominated HV:

$$\Delta s(\lambda, \mathcal{P}) = S(\mathcal{P}, R) - S(\mathcal{P} \setminus \{\lambda\}, R).$$

- Dark rectangles: HV contribution of dots.
- Grey point: reference point.
- The HVC contribution is the volume of space that is dominated only by λ , and nothing else.
- $\tilde{\lambda}$ has lowest S-metric contribution.



SMS-EMOA algorithm I

Algorithm 2 SMS-EMOA

- 1: Generate start population \mathcal{P}_0 of size μ
- 2: $t \leftarrow 0$
- 3: **repeat**
- 4: Generate **one** individual \mathbf{q} by recombination and mutation of \mathcal{P}_t
- 5: $\{\mathcal{F}_1, \dots, \mathcal{F}_k\} \leftarrow \text{NDS}(\mathcal{P}_t \cup \{\mathbf{q}\})$
- 6: $\boldsymbol{\lambda} \leftarrow \operatorname{argmin}_{\boldsymbol{\lambda} \in \mathcal{F}_k} \Delta s(\boldsymbol{\lambda}, \mathcal{F}_k)$
- 7: $\mathcal{P}_{t+1} \leftarrow (\mathcal{P}_t \cup \{\mathbf{q}\}) \setminus \{\tilde{\boldsymbol{\lambda}}\}$
- 8: $t \leftarrow t + 1$
- 9: **until** Termination criterion fulfilled

- L5: the set of temporary $(\mu + 1)$ individuals is partitioned by NDS into k fronts $\mathcal{F}_1, \dots, \mathcal{F}_k$.
- L6-7: In last front, find $\tilde{\boldsymbol{\lambda}} \in \mathcal{F}_k$ with smallest HV contribution - and kill it.
- Fitness of an individual is mainly the rank of its front and HV contribution as tie-breaker.

Multi-criteria Optimization

Bayesian Optimization

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Recap: Bayesian Optimization I

Advantages of BO

- Sample efficient
- Can handle noise
- Native incorporation of priors
- Does not require gradients
- Theoretical guarantees

We will now extend BO to multiple cost functions.

Recap: Bayesian Optimization II

Bayesian optimization loop

Require: Search space Λ , cost function c , acquisition function u , predictive model \hat{c} , maximal number of function evaluations T

Result : Best configuration $\hat{\lambda}$ (according to \mathcal{D} or \hat{c})

- 1 Initialize data $\mathcal{D}^{(0)}$ with initial observations
 - 2 **for** $t = 1$ **to** T **do**
 - 3 Fit predictive model $\hat{c}^{(t)}$ on $\mathcal{D}^{(t-1)}$
 - 4 Select next query point: $\lambda^{(t)} \in \arg \max_{\lambda \in \Lambda} u(\lambda; \mathcal{D}^{(t-1)}, \hat{c}^{(t)})$
 - 5 Query $c(\lambda^{(t)})$
 - 6 Update data: $\mathcal{D}^{(t)} \leftarrow \mathcal{D}^{(t-1)} \cup \{\langle \lambda^{(t)}, c(\lambda^{(t)}) \rangle\}$
-

Multi-Criteria Bayesian Optimization

Goal: Extend Bayesian optimization to multiple cost functions

$$\min_{\lambda \in \Lambda} c(\lambda) \Leftrightarrow \min_{\lambda \in \Lambda} (c_1(\lambda), c_2(\lambda), \dots, c_m(\lambda)).$$

There are two basic approaches:

- ① Simplify the problem by scalarizing the cost functions, or
- ② define acquisition functions for multiple cost functions.

Scalarization

Idea: Aggregate all cost functions

$$\min_{\lambda \in \Lambda} \sum_{i=1}^m w_i c_i(\lambda) \quad \text{with} \quad w_i \geq 0$$

- **Obvious problem:** How to choose w_1, \dots, w_m ?
 - ▶ Expert knowledge?
 - ▶ Systematic variation?
 - ▶ Random variation?
- If expert knowledge is not available a-priori, we need to ensure that different trade-offs between cost functions are explored.
- Simplifies multi-criteria optimization problem to single-objective
→ Bayesian optimization can be used without adaption of the general algorithm.

Scalarization: ParEGO [Knowles. 2006]

Scalarize the cost functions using the augmented Tchebycheff norm / achievement function

$$c = \max_{i=1,\dots,m} (w_i c_i(\boldsymbol{\lambda})) + \rho \sum_{i=1}^m w_i c_i(\boldsymbol{\lambda}),$$

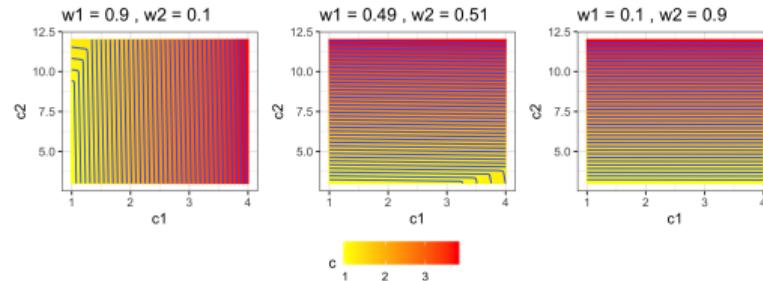
- The weights $w \in W$ are drawn from

$$W = \left\{ w = (w_1, \dots, w_m) \mid \sum_{i=1}^m w_i = 1, w_i = \frac{l}{s} \wedge l \in 0, \dots, s \right\},$$

with $|W| = \binom{s+m-1}{k-1}$.

- New weights are drawn in every BO iteration.
- ρ is a small parameter suggested to be set to 0.05.
- s selects the number of different weights to draw from.

Why the Tchebycheff norm?



$$c = \max_{i=1,\dots,m} (w_i c_i(\lambda)) + \rho \sum_{i=1}^m w_i c_i(\lambda),$$

- The norm consists of two components:
 - ▶ $\max_{i=1,\dots,m} (w_i c_i(\lambda))$ takes only the maximum weighted cost into account.
 - ▶ $\sum_{i=1}^m w_i c_i(\lambda)$ is the weighted sum of all cost functions.
- ρ describes the trade-off between these components.
- By the randomized weights in each iteration and the usually small value of $\rho = 0.05$, this allows exploration of extreme points of single cost functions.
- One can prove: **Every solution of the scalarized problem is pareto-optimal!**

ParEGO Algorithm

ParEGO loop

Require: Search space Λ , cost function c , acquisition function u , predictive model \hat{c} , maximal number of function evaluations T , ρ , l , s

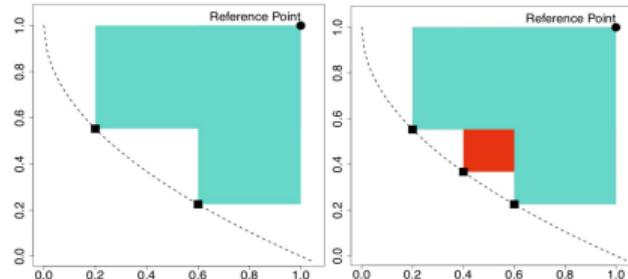
Result : Best configuration $\hat{\lambda}$ (according to \mathcal{D} or \hat{c})

- 1 Initialize data $\mathcal{D}^{(0)}$ with initial observations
 - 2 **for** $t = 1$ **to** T **do**
 - 3 Sample w from $\{w = (w_1, \dots, w_m) \mid \sum_{i=1}^m w_i = 1, w_i = \frac{l}{s} \wedge l \in 0, \dots, s\}$;
 - 4 Compute scalarization $c^{(t)} = \max_{i=1, \dots, m} (w_i c_i(\lambda)) + \rho \sum_{i=1}^m w_i c_i(\lambda)$;
 - 5 Fit predictive model $\hat{c}^{(t)}$ on $\mathcal{D}^{(t-1)}$
 - 6 Select next query point: $\lambda^{(t)} \in \arg \max_{\lambda \in \Lambda} u(\lambda; \mathcal{D}^{(t-1)}, \hat{c}^{(t)})$
 - 7 Query $c(\lambda^{(t)})$
 - 8 Update data: $\mathcal{D}^{(t)} \leftarrow \mathcal{D}^{(t-1)} \cup \{\langle \lambda^{(t)}, c(\lambda^{(t)}) \rangle\}$
-

Hypervolume based Acquisition Functions

Idea: Define acquisition function that directly models contribution to dominated HV.

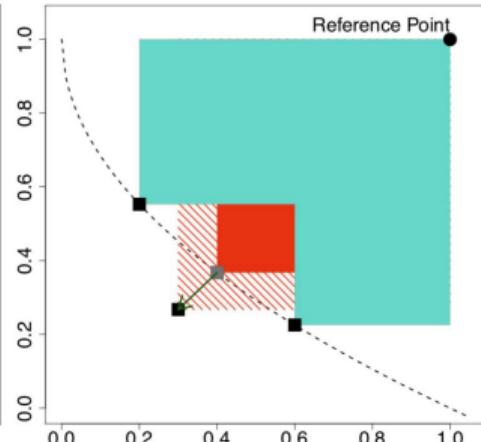
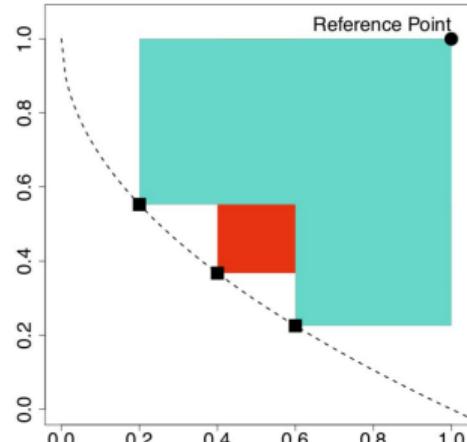
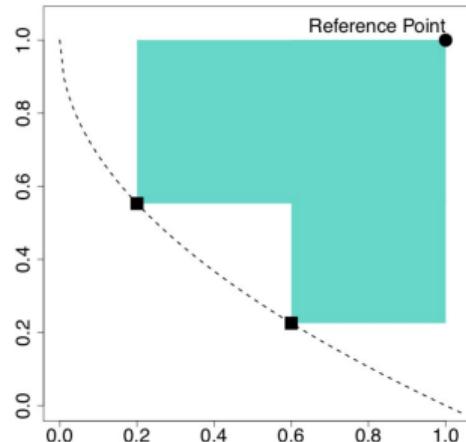
$$\max(0, S(\mathcal{P} \cup \lambda, R) - S(\mathcal{P}, R))$$



- Fit m single-objective surrogate models $\hat{c}_1, \dots, \hat{c}_m$
- Acquisition function takes all surrogate models into account.
- Single-criteria optimization of acquisition function.

S-Metric Selection-based EGO I

Using the Lower Confidence bound $u_{LCB,1}(\lambda), \dots, u_{LCB,m}(\lambda)$, an optimistic estimate of hypervolume contribution can be calculated.



S-Metric Selection-based EGO II

Problem: Based on the way the hypervolume contribution is measured large plateaus of zero improvement are present.

- These make optimization much harder.
- An adaptive penalty is added to regions in which the lower confidence bound is dominated.

This method is referred to as SMS-EGO [Ponweiser et al. 2008].

Further Hypervolume based Acquisition Functions

Expected Hypervolume Improvement (EHI) [Yang et al. 2019]

$$u_{EI,\mathcal{H}}(\boldsymbol{\lambda}) = \int_{-\infty}^{\infty} p(c \mid \boldsymbol{\lambda}) \times \mathcal{H}(\boldsymbol{\lambda}) \ dc,$$

with $\mathcal{H}(\boldsymbol{\lambda}) = S(\mathcal{P} \cup \boldsymbol{\lambda}, R) - S(\mathcal{P}, R)$.

- Direct extension of u_{EI} to the hypervolume.
- $p(c \mid \boldsymbol{\lambda})$ is the joint density of the surrogate model predictions at $\boldsymbol{\lambda}$.
- As the surrogates are GPs and modeled independently of each other, this is just an integral over m univariate normal distributions.
- Efficient computations for $m \leq 3$ exist, beyond that expensive simulation-based computation is required.

Further hypervolume based acquisition functions:

- **Stepwise Uncertainty Reduction** (SUR) based on the probability of improvement.
- **Expected Maximin Improvement** (EMI) based on the ϵ -indicator.

Hypervolume based BO Algorithm

Hypervolume based Bayesian optimization loop

Require: Search space Λ , cost function c , acquisition function u , predictive model \hat{c} , maximal number of function evaluations T

Result : Best configuration $\hat{\lambda}$ (according to \mathcal{D} or \hat{c})

- 1 Initialize data $\mathcal{D}^{(0)}$ with initial observations
 - 2 **for** $t = 1$ **to** T **do**
 - 3 Fit predictive models $\hat{c}_1^{(t)}, \dots, \hat{c}_m^{(t)}$ on $\mathcal{D}^{(t-1)}$
 - 4 Select next query point: $\lambda^{(t)} \in \arg \max_{\lambda \in \Lambda} u(\lambda; \mathcal{D}^{(t-1)}, \hat{c}_1^{(t)}, \dots, \hat{c}_m^{(t)})$
 - 5 Query $c(\lambda^{(t)})$
 - 6 Update data: $\mathcal{D}^{(t)} \leftarrow \mathcal{D}^{(t-1)} \cup \{\langle \lambda^{(t)}, c(\lambda^{(t)}) \rangle\}$
-

Multi-criteria Optimization

Practical Applications

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Practical Applications in Machine Learning I

ROC Optimization: Balance *true positive* and *false positive* rates

- Typically unbalanced classification tasks with unspecified costs.
- Could also use other ROC metrics, e.g., *positive predicted value* or *false discovery rate*.

Efficient Models: Balance *predictive performance* with *prediction time*, *energy consumption* and/or *model size*.

- Time: Models in production need to predict fast.
- Size / Energy consumption: Models should be deployed on a mobile/edge device and not use much power.

Sparse Models: Balance *predictive performance* and *number of used features*, either for cost efficiency, but often also for interpretability.

Fair Models: Balance *predictive performance* and *fairness*.

- Model has to be fair regarding subgroups in the data, e.g. gender.
- Many different approaches to quantify fairness exist.

ROC Optimization - Setup

Again, we want to train a *spam detector* on the popular Spam dataset¹.

- Learning algorithm: SVM with RBF kernel.
- Hyperparameters to optimize:
 - cost $[2^{-15}, 2^{15}]$
 - γ $[2^{-15}, 2^{15}]$
 - Threshold t $[0, 1]$
- Objective: *minimize* false positive rate (FPR) and *maximize* true positive rate (TPR), evaluated through 5-fold CV
- Optimizer: Multi-criteria Bayesian optimization:
 - ▶ ParEGO with $\rho = 0.05$, $s = 100000$.
 - ▶ Acquisition function u : *Confidence Bound* with $\alpha = 2$.
 - ▶ Budget: 100 evaluations
- Tuning is conducted on a training holdout and all hyperparameter configurations on the estimated Pareto front are validated on an outer validation set.

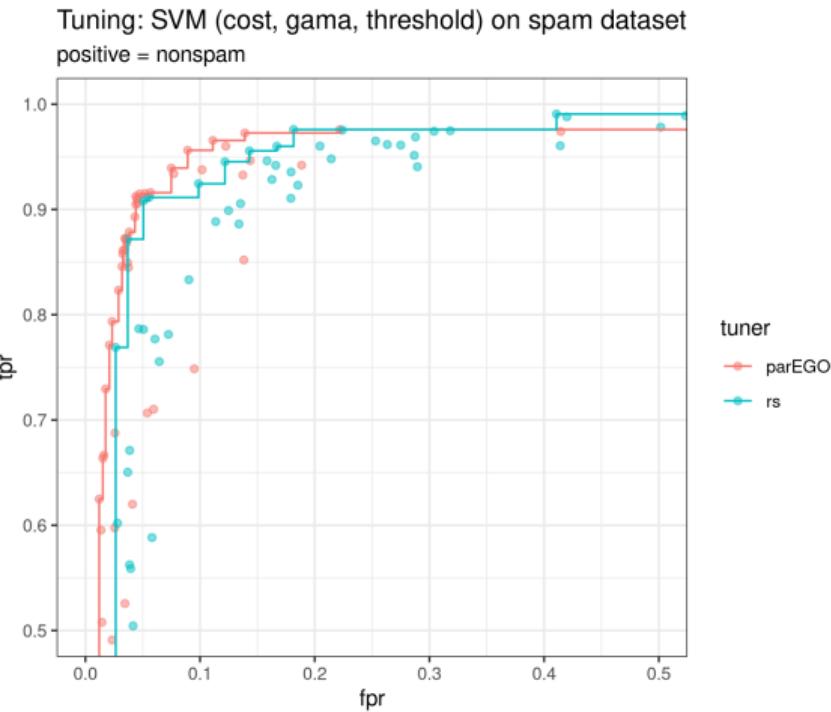
The threshold t could be separately optimized post-hoc.

¹<https://archive.ics.uci.edu/ml/datasets/spambase>

ROC Optimization - Result I

We notice:

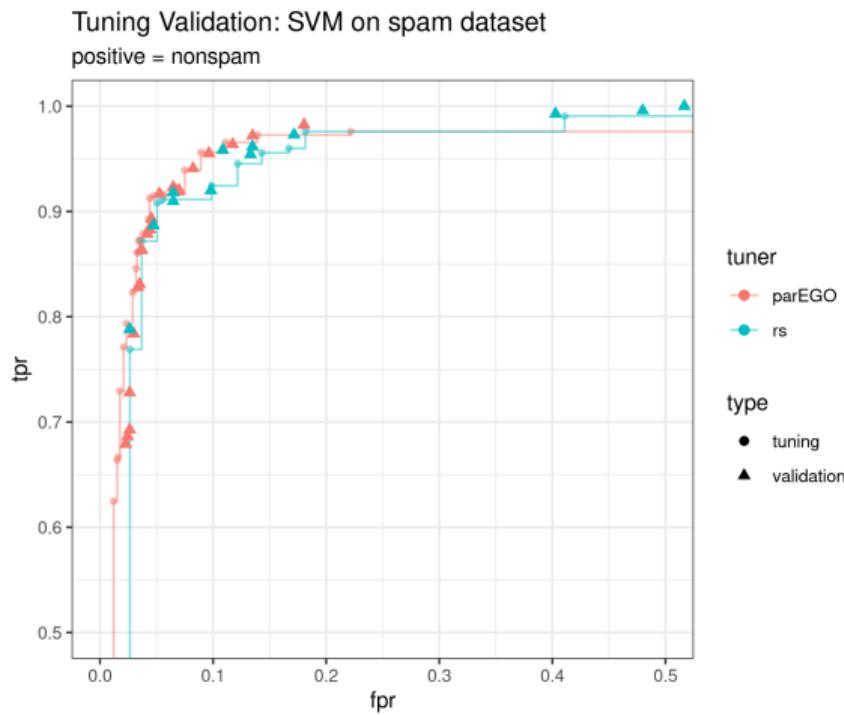
- Compared to *random search*: Many *ParEGO* evaluations are on the Pareto front.
- The Pareto front of *ParEGO* dominates most points from the *random search*.
- The dominated hypervolume to the reference point $(0, 1)$ is:
 - ParEGO*: 0.965
 - random search*: 0.959



ROC Optimization - Result II

We validate the configurations on the estimated Pareto front on a holdout:

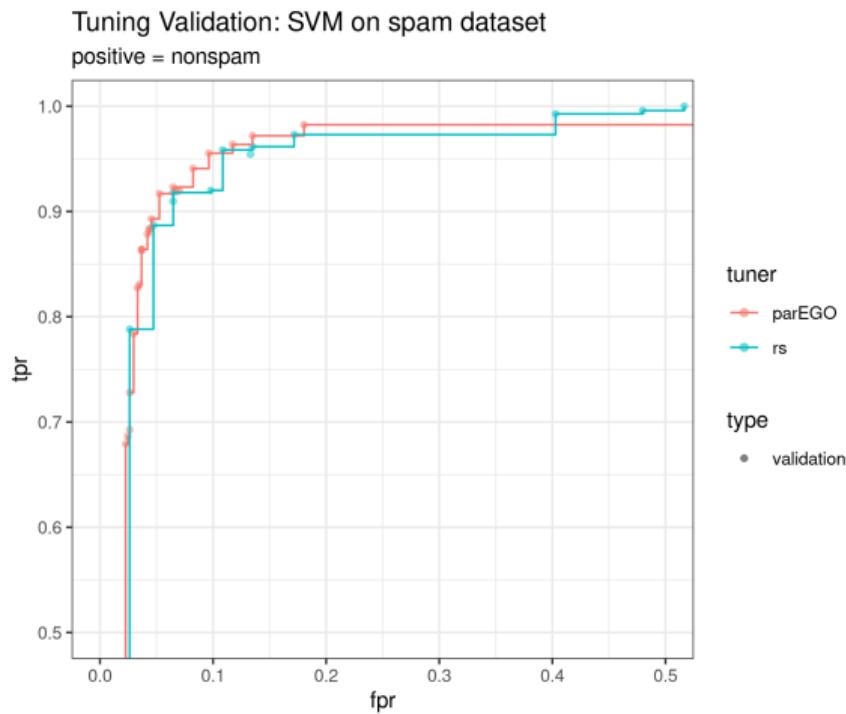
- The performance on the validation set varies slightly.
- The TPR got slightly better but the FPR got slightly worse.
- On the validation set, some configurations get dominated by others.



ROC Optimization - Result II

We validate the configurations on the estimated Pareto front on a holdout:

- The performance on the validation set varies slightly.
- The TPR got slightly better but the FPR got slightly worse.
- On the validation set, some configurations get dominated by others.
- The dominated hypervolume of the validation set is:
ParEGO: 0.960
random search: 0.961



Efficient Models - Overview

- "Efficiency" can be:
 - ▶ Memory consumption of the model
 - ▶ Training or prediction time
 - ▶ Number of features needed
 - ▶ Energy consumption for prediction
 - ▶ ...
- Some hyperparameters have a strong impact on the efficiency of a model, e.g.,
 - ▶ Number of trees in *random forests* or *gradient tree boosting*,
 - ▶ Number, size and type of layers in *neural networks*,
 - ▶ L1 regularization penalties,
 - ▶ ...
- Other hyperparameters might have no influence on efficiency.
- Typical scenario: Optimize jointly over multiple algorithms of varying efficiency.

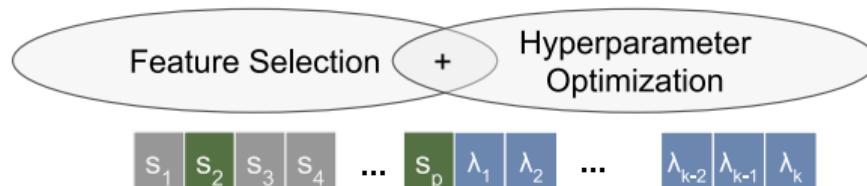
Efficient Models - Example: Feature Selection I

Goal of *feature selection*: Identify an informative feature subset with only a small drop in predictive performance compared to all features.

Find optimal hyperparameter setting λ and minimal feature subset s

$$\min_{\lambda \in \Lambda, s \in \{0,1\}^p} \left(\widehat{GE} (\mathcal{I}(\mathcal{D}, \lambda, s)), \frac{1}{p} \sum_{i=1}^p s_i \right)$$

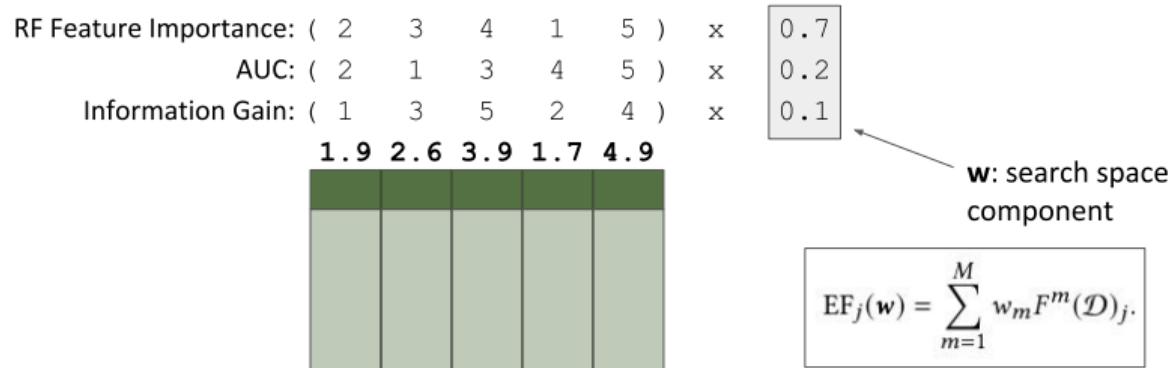
- Problem: Feature selection and hyperparameter tuning are usually two separate steps.
- Solution: Identify an informative subset of features and a good hyperparameter configuration **simultaneously**.



Efficient Models - Example: Feature Selection II

Idea: *Multi-Objective Hyperparameter Tuning and Feature Selection using Filter Ensembles* [Binder et al. 2020]:

- Pre-calculate multiple ranked *feature filter values*.

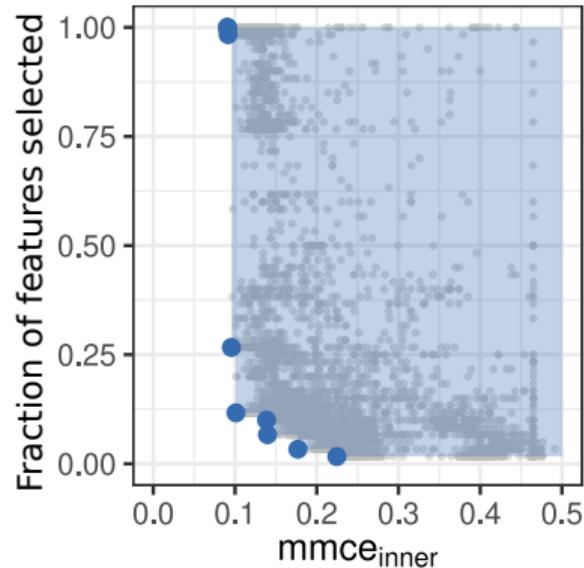


- New joint hyperparameter vector: $\boldsymbol{\lambda} = (\tilde{\boldsymbol{\lambda}}, w_1, \dots, w_p, \tau)$
 - Hyperparameters of learner: $\tilde{\boldsymbol{\lambda}}$
 - Weight of each *feature filter value* vector: (w_1, \dots, w_p)
 - Fraction of features to keep τ

Efficient Models - Example: Feature Selection III

Combined feature selection and hyperparameter optimization on Sonar dataset².

- Learning algorithm: SVM with RBF kernel.
- Hyperparameters to optimize:
 - cost $[2^{-10}, 2^{10}]$
 - γ $[2^{-10}, 2^{10}]$
 - (w_1, \dots, w_p) $[0, 1]^p$
 - τ $[0, 1]^p$
- Objective: minimize *misclassification* and *fraction of features selected*
- Optimizer: *ParEGO* with *random forest* surrogate, LCB acquisition function, 15 batch proposals, budget: 2000 evaluations



²Only the tuning error is shown here

Efficient Models - Example: FLOPS

Goal: Optimize prediction accuracy and number of floating point operations (FLOPs)
[Wang et al. 2019].

Data: Image Classification on CIFAR-10.

Learner: *DenseNet* - Densely Connected Convolutional Network [Huang et al. 2018].

- Composed out of 4 *dense blocks*.
- A dense block consists of multiple convolutional layers where the inputs for each layer are all feature maps of all preceding layers in the block.
- Dense blocks are connected via convolutional and max pooling layer.

Training: 300 Epochs with a batch size of 128 and initial learning rate of 0.1.

Efficient Models - Example: FLOPS

- Objective: *accuracy* vs. *FLOPS* (floating point operations, per observation)

- Search Space:

growth rate (k) [8, 32]

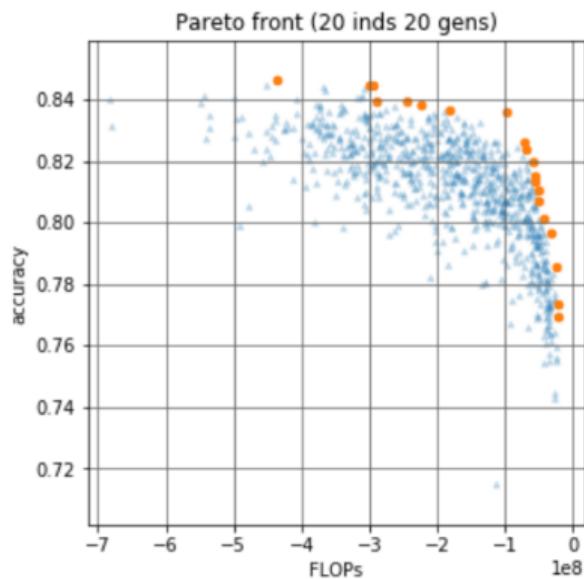
layers in first block [4, 6]

layers in second block [4, 12]

layers in third block [4, 24]

layers in fourth block [4, 16]

- Tuner: *Particle Swarm Optimization* with a population size of 20 and 400 evaluations.

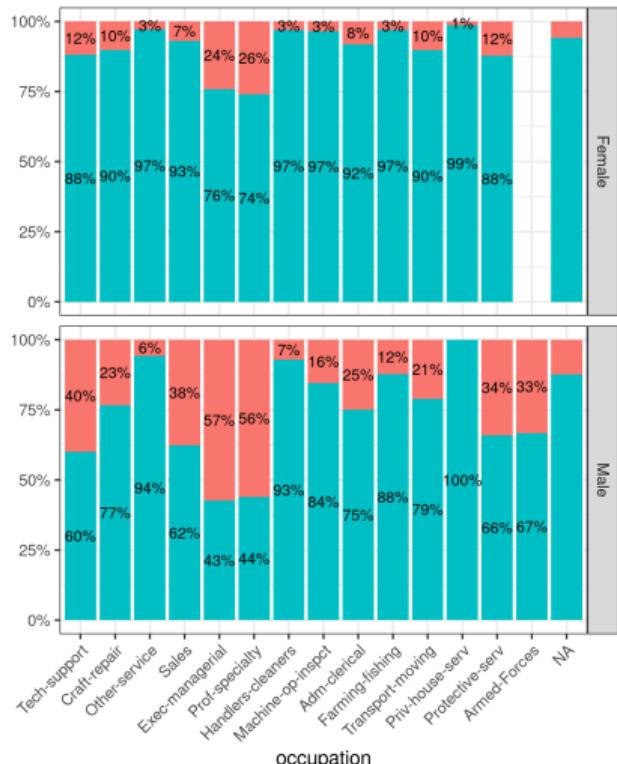
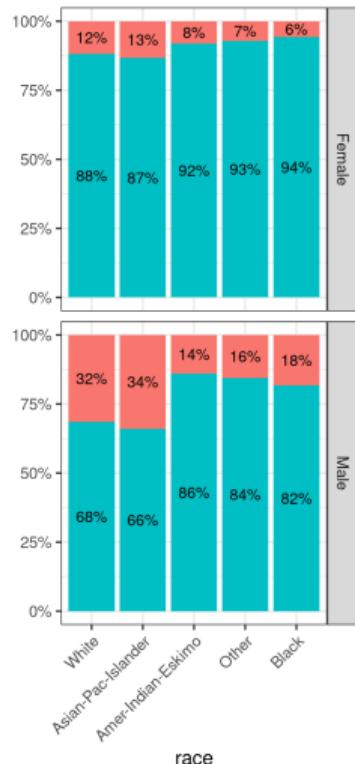
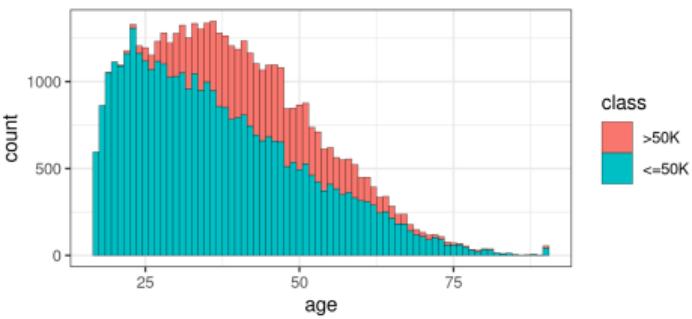


The growth rate is the number of output feature maps in each layer of a block

Fair Models - The Adult dataset

Dataset: Adult

- Source: US Census database, 1994, <https://www.openml.org/d/1590>.
- 48842 observations
- Target: binary, income above 50k
- 14 features: age, education, hours.per.week, marital.status, native.country, occupation, race, relationship, sex, ...



Fair Models - Setup I

A fair model for income prediction on binarized target.

- Learner: *eXtreme Gradient Boosting*
- Hyperparameters to optimize:

eta	[0.01, 0.2]
gamma	$[2^{-7}, 2^6]$
max_depth	{2, ..., 20}
colsample_bytree	[0.5, 1]
colsample_bylevel	[0.5, 1]
lambda	$[2^{-10}, 2^{10}]$
alpha	$[2^{-10}, 2^{10}]$
subsample	[0.5, 1]

- Objective: minimize *misclassification error* and *unfairness*

Fair Models - Setup II

- Careful: Usually this data would be used to model the relation between person characteristics and income, then to **discuss and study** by careful inference - to **figure out if** something like e.g. a "gender pay gap" exists.
- Here, in our toy example we **pretend** now that we would like to create a automatic "assignment algorithm" for salary - maybe not totally unrealistic nowadays? In **such** a scenario, biasing the prediction by **incorporating fairness** might be of interest.
- Here, a simplified proxy for fairness is defined as the absolute difference in F1-Scores between female (f) and male (m) population (low is good):

$$L_{\text{fair}} := |L_{\text{F1}}(y_f, \hat{f}(\mathbf{x}_f)) - L_{\text{F1}}(y_m, \hat{f}(\mathbf{x}_m))|$$

Fair Models - Results

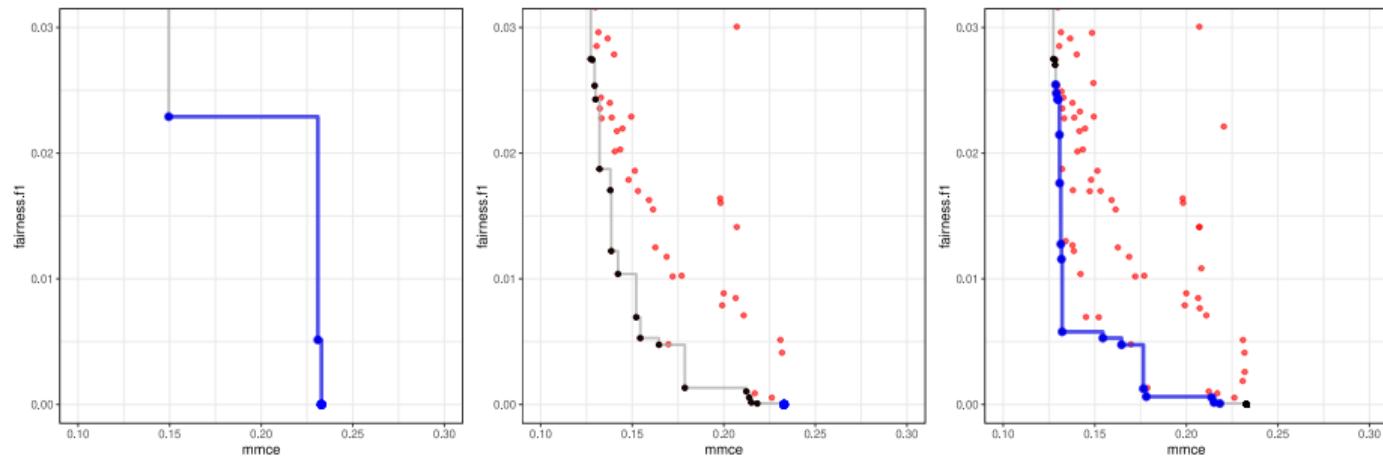


Figure: Pareto fronts after 20, 70 and 120 tuning iterations.

- Optimizer: ParEGO with random forest surrogate and restricted range of projections to $[0.1, 0.9]$ (No interest in very unfair or bad configurations).
- Here, the hyperparameters actually have an effect on the defined *fairness measure*.
- However, this is often not the case or not enough to ensure a fair model.

AutoML: Neural Architecture Search (NAS)

Overview

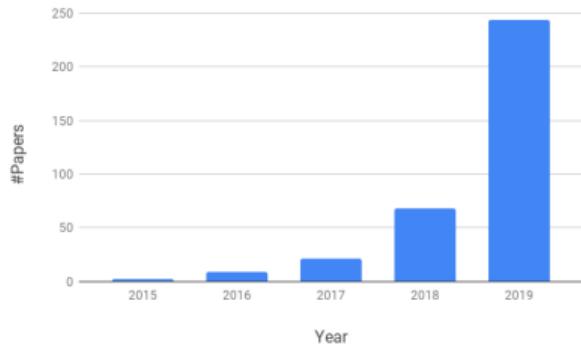
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Neural Architecture Search (NAS)

- Goal: automatically find neural architectures with strong performance
 - ▶ Optionally, subject to a resource constraint

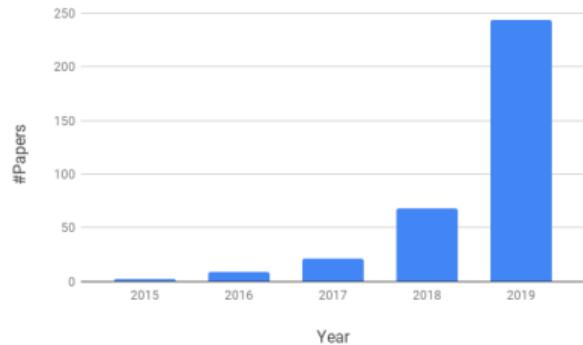
Neural Architecture Search (NAS)

- Goal: automatically find neural architectures with strong performance
 - ▶ Optionally, subject to a resource constraint
- A decade-old problem, but main stream since 2017 and now intensely researched
- One of the main problems AutoML is known for



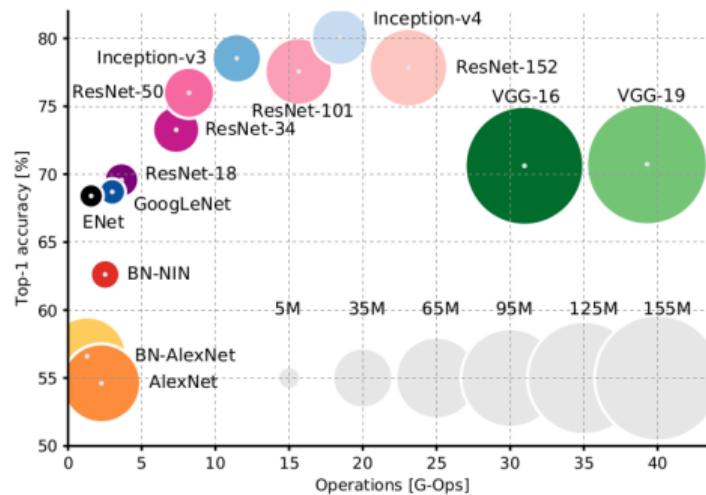
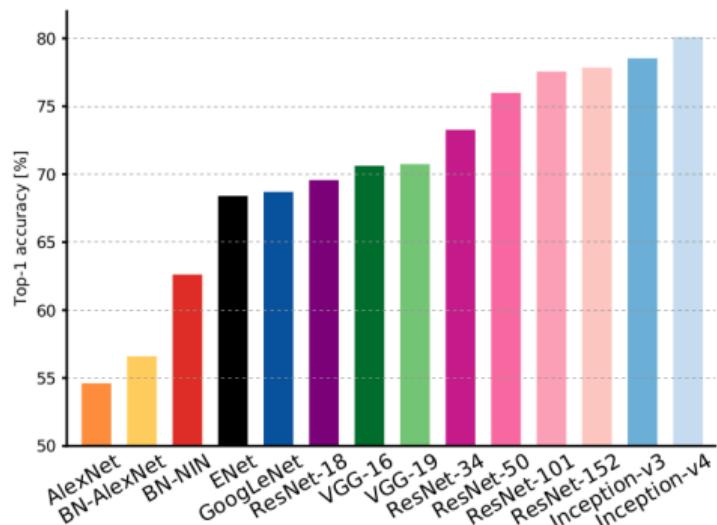
Neural Architecture Search (NAS)

- Goal: automatically find neural architectures with strong performance
 - ▶ Optionally, subject to a resource constraint
- A decade-old problem, but main stream since 2017 and now intensely researched
- One of the main problems AutoML is known for
- Initially extremely expensive
- By now several methods promise low overhead over a single model training



Motivation for NAS

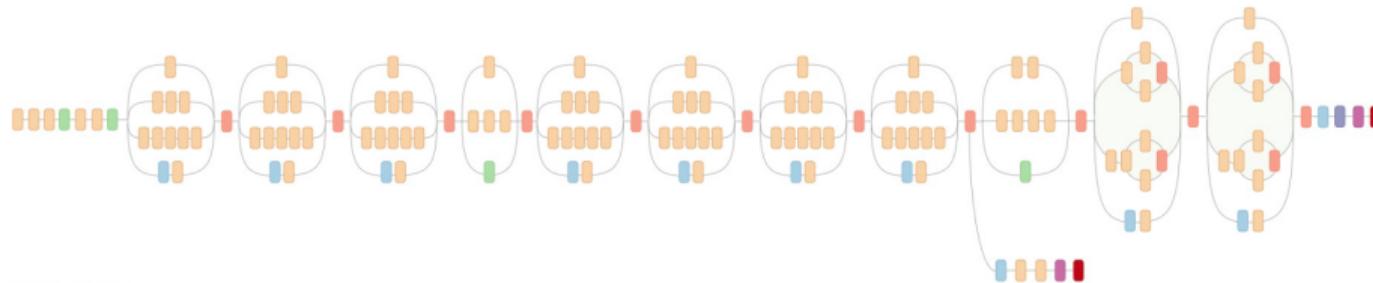
- Performance improvements on various tasks due to novel architectures
- Can we automate this design process, potentially discovering new components/topologies?



[Canziani et al. 2017]

Motivation for NAS

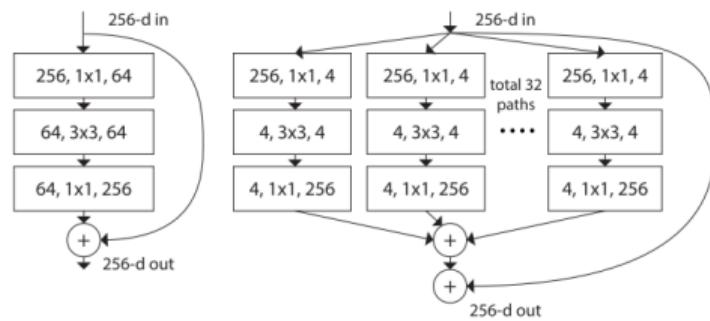
- Manual design of architectures is **time consuming**
- Complex state-of-the-art architectures are a result of **years of trial and errors by experts**



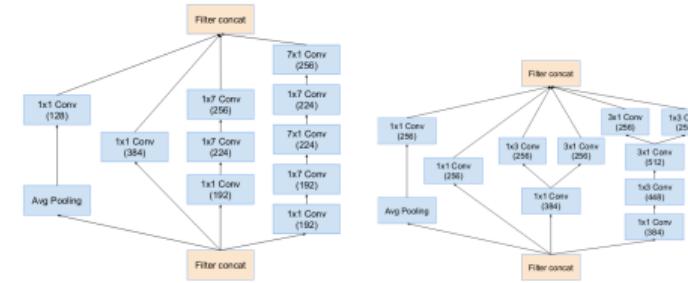
Inception-v3 [Szegedy et al. 2015]

Motivation for NAS

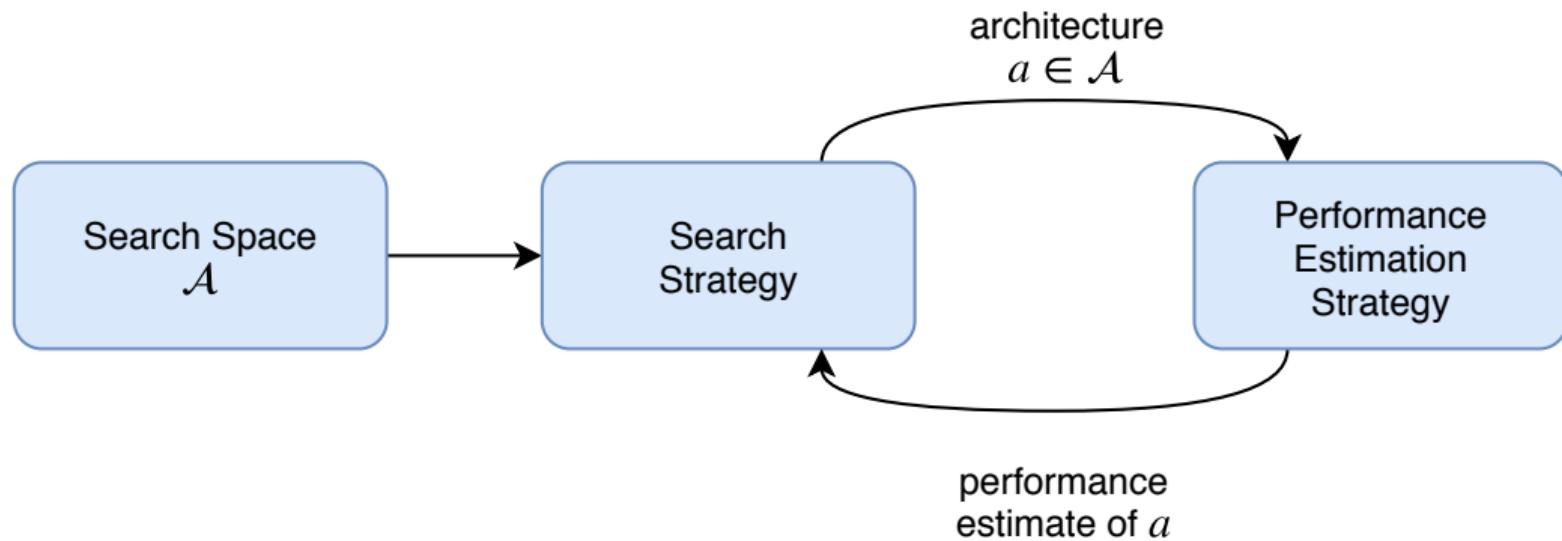
- Manual design of architectures is **time consuming**
- Complex state-of-the-art architectures are a result of **years of trial and errors by experts**
 - Main pattern: Repeated blocks with same structure (topology)



ResNet/ResNeXt blocks
[He et al. 2016; Xie et al. 2016]



Inception-v4 blocks [Szegedy et al. 2016]



- **Search Space:** the types of architectures we consider; micro, macro, hierarchical, etc.
- **Search Strategy:** Reinforcement learning, evolutionary strategies, Bayesian optimization, gradient-based, etc.
- **Performance Estimation Strategy:** validation performance, lower fidelity estimates, one-shot model performance, etc.

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
List three major components of NAS methods.
- Discussion:
Is there a problem for which you would like to apply NAS yourself?

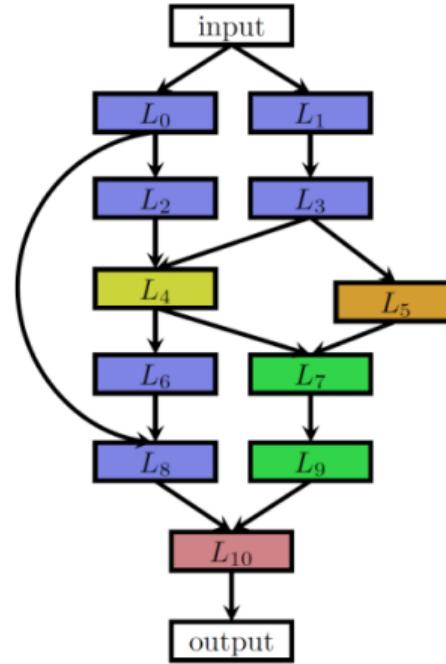
AutoML: Neural Architecture Search (NAS) Search Spaces

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Basic Neural Architecture Search Spaces

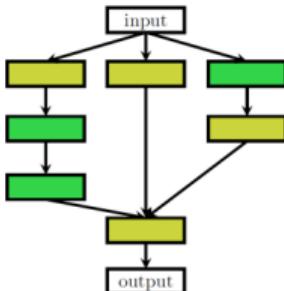
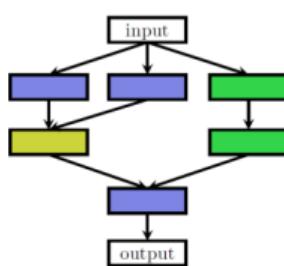


Chain-structured space
(different colours:
different layer types)

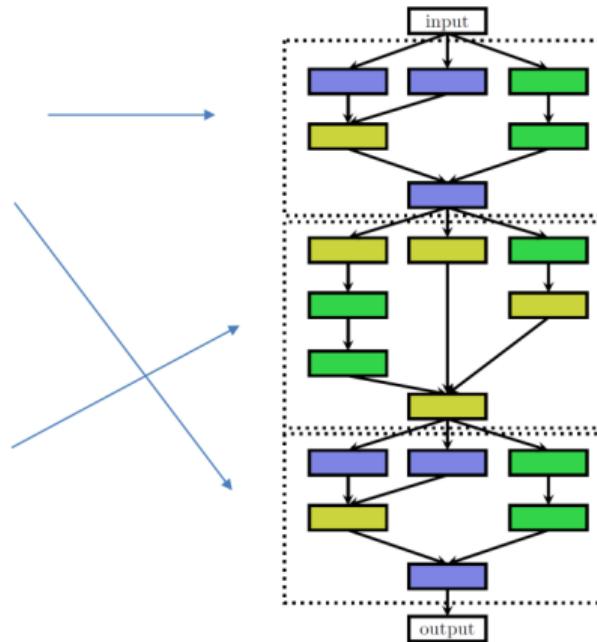


More complex space
with multiple branches
and skip connections

Cell Search Spaces [Zoph et al. 2018]



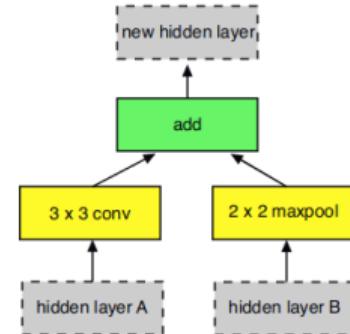
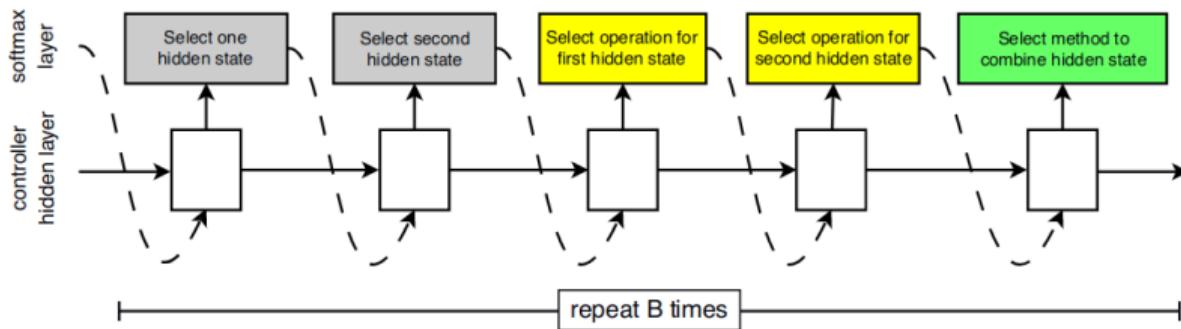
Two possible cells



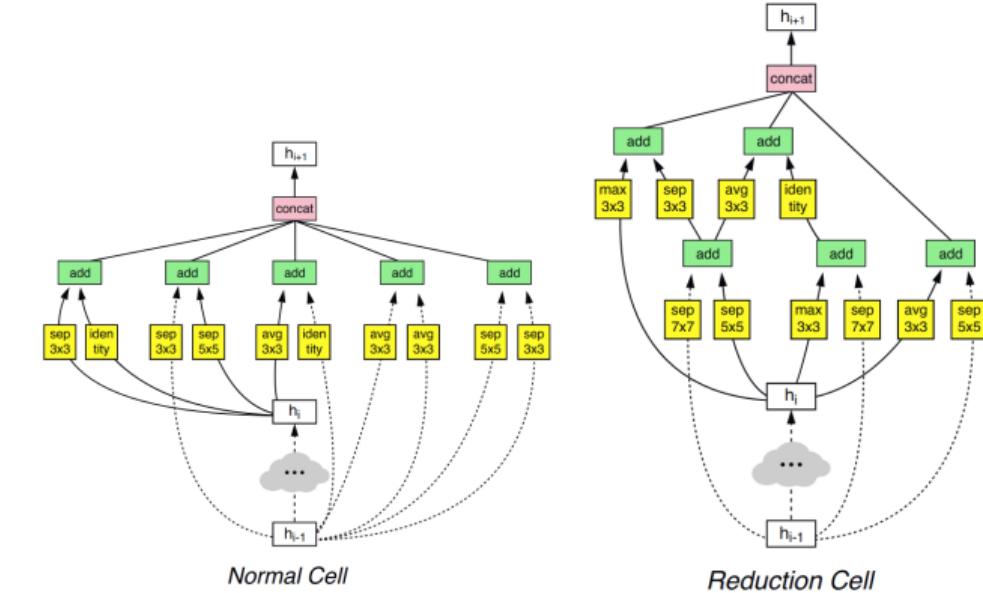
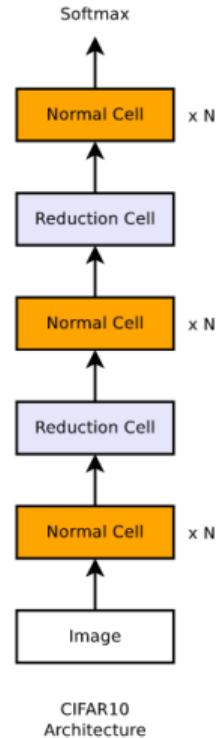
Architecture composed
of stacking together
individual cells

Details on Cell Search Spaces [Zoph et al. 2018]

- 2 types of cells: normal and reduction cells
- For each type of cell: B blocks, each with 5 choices
 - Choose two previous feature maps (from this cell)
 - For each of these, choose an operation (3×3 conv, max-pool, etc.)
 - Choose a merge operation to combine the two results (concat or add)



Example of an architecture sample with $B=5$



Source: [Zoph et al. 2018]

Pros and Cons of Cell Search Space

What are some pros and cons of the cell search space compared to the basic one?

Please think about this for a few minutes before continuing.

Pros and Cons of Cell Search Space

Pros:

- Reduced search space size; speed-ups in terms of search time.
- Transferability to other datasets (e.g., cells found on CIFAR-10 transfer to ImageNet)
- Stacking repeating patterns is proven to be a useful design principle (ResNet, Inception, etc.)

Pros and Cons of Cell Search Space

Pros:

- Reduced search space size; speed-ups in terms of search time.
- Transferability to other datasets (e.g., cells found on CIFAR-10 transfer to ImageNet)
- Stacking repeating patterns is proven to be a useful design principle (ResNet, Inception, etc.)

Cons:

- Still need to (manually) determine the *macro* architecture, i.e., the way in which cells are connected.
- Limiting if different cells work better in different parts of the network
 - E.g., different spatial resolutions may favour different convolutions

Hierarchical representation of search space [Liu et al. 2017]

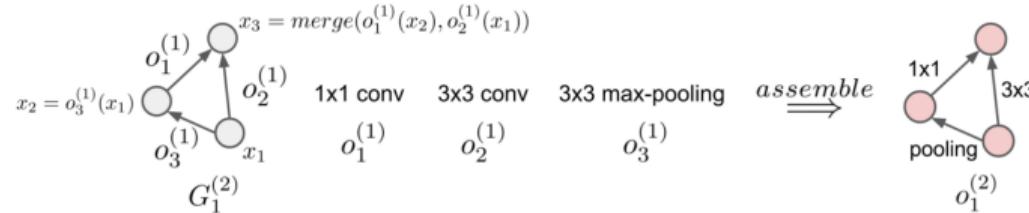
- Directed Acyclic Graph (DAG) representation of architectures
 - Each node is a latent representation; each edge is an operation/motif

Hierarchical representation of search space [Liu et al. 2017]

- Directed Acyclic Graph (DAG) representation of architectures
 - Each node is a latent representation; each edge is an operation/motif
- There are different **levels** of motifs
 - ▶ **Level-1 primitives:** standard operators; e.g., 3x3 conv, max pooling, ...

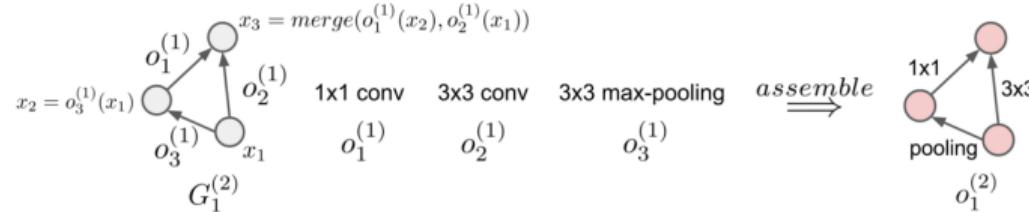
Hierarchical representation of search space [Liu et al. 2017]

- Directed Acyclic Graph (DAG) representation of architectures
 - Each node is a latent representation; each edge is an operation/motif
- There are different **levels** of motifs
 - ▶ **Level-1 primitives:** standard operators; e.g., 3x3 conv, max pooling, ...
 - ▶ **Level-2 motifs:** combinations of level-1 primitives

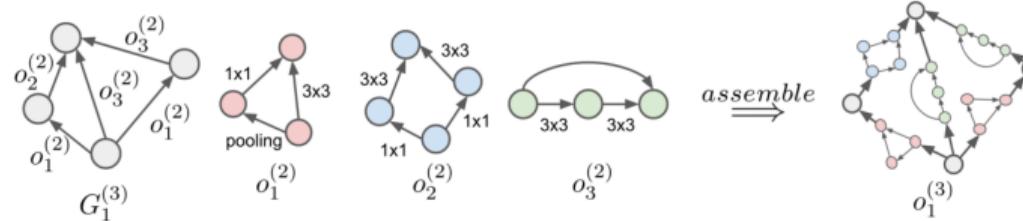


Hierarchical representation of search space [Liu et al. 2017]

- Directed Acyclic Graph (DAG) representation of architectures
 - Each node is a latent representation; each edge is an operation/motif
- There are different **levels** of motifs
 - ▶ **Level-1 primitives:** standard operators; e.g., 3x3 conv, max pooling, ...
 - ▶ **Level-2 motifs:** combinations of level-1 primitives



- ▶ **Level-3 motifs:** combinations of level-2 motifs



Pros and Cons of Hierarchical Search Space

What are some pros and cons of a hierarchical search space compared to the cell search space?

Please think about this for a few minutes before continuing.

Pros and Cons of Hierarchical Search Space

Pros:

- Flexibility of constructing building blocks and reusing them many times
 - ▶ like a cell search space
- Flexibility of using different building blocks in different parts of the network
 - ▶ like a basic search space
- Ability to reuse building blocks at various levels of abstraction
 - ▶ again, this pattern has been used in manual design, e.g., in Inception nets

Pros and Cons of Hierarchical Search Space

Pros:

- Flexibility of constructing building blocks and reusing them many times
 - ▶ like a cell search space
- Flexibility of using different building blocks in different parts of the network
 - ▶ like a basic search space
- Ability to reuse building blocks at various levels of abstraction
 - ▶ again, this pattern has been used in manual design, e.g., in Inception nets

Cons:

- Larger than cell search space
- Vastly more expressive than cell search space → potentially much harder to search

Questions to Answer for Yourself / Discuss with Friends

- Repetition:

What are some pros and cons of the cell search space compared to the basic one?

- Repetition:

Explain the way in which level-3 motifs in the hierarchical search space use level-2 motifs.

- Repetition:

What are some pros and cons of the hierarchical search space compared to the other ones?

AutoML: Neural Architecture Search (NAS)

Blackbox Optimization Methods

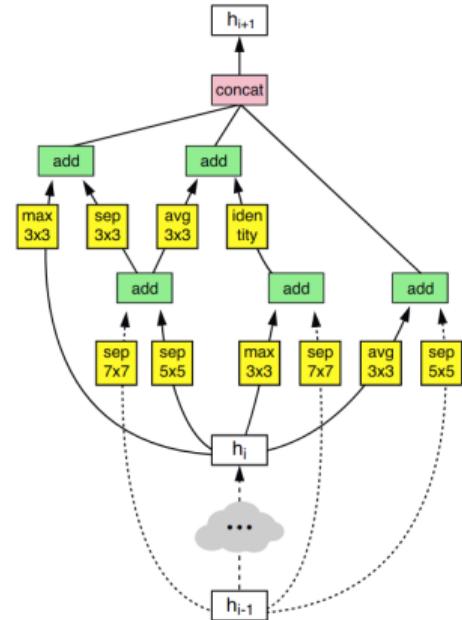
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

NAS as Hyperparameter Optimization

- NAS can be formulated as a HPO problem

NAS as Hyperparameter Optimization

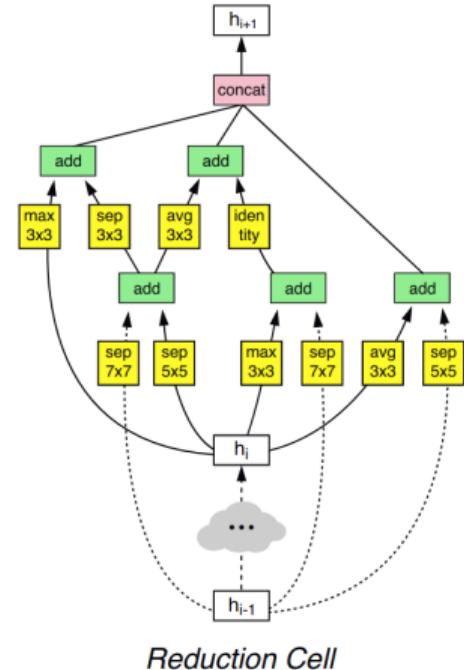
- NAS can be formulated as a HPO problem
- E.g., cell search space by [Zoph et al. 2018] has 5 categorical choices per block
 - ▶ 2 categorical choices of hidden states
 - ★ For block N , the domain of these categorical variables is $\{h_i, h_{i-1}, \text{output of block } 1, \dots, \text{output of block } N-1\}$
 - ▶ 2 categorical variables choosing between operations
 - ▶ 1 categorical variable choosing the combination method
 - ▶ Total number of hyperparameters for the cell:
5B (with $B=5$ by default)



Reduction Cell

NAS as Hyperparameter Optimization

- NAS can be formulated as a HPO problem
- E.g., cell search space by [Zoph et al. 2018]
has 5 categorical choices per block
 - ▶ 2 categorical choices of hidden states
 - ★ For block N , the domain of these categorical variables is $\{h_i, h_{i-1}, \text{output of block 1}, \dots, \text{output of block } N-1\}$
 - ▶ 2 categorical variables choosing between operations
 - ▶ 1 categorical variable choosing the combination method
 - ▶ Total number of hyperparameters for the cell:
5B (with $B=5$ by default)
- In general: one may require conditional hyperparameters
 - ▶ E.g., chain-structured search space
 - ★ Top-level hyperparameter: number of layers L
 - ★ Hyperparameters of layer k conditional on $L \geq k$



Early Work on Neuroevolution (already since the 1990s)

[Kitano. 1990; Angeline et al. 1994; Stanley and Miikkulainen. 2002; Bayer et al. 2009; Floreano et al. 2008]

- Evolves architectures & often also their weights

Early Work on Neuroevolution (already since the 1990s)

[Kitano. 1990; Angeline et al. 1994; Stanley and Miikkulainen. 2002; Bayer et al. 2009; Floreano et al. 2008]

- Evolves architectures & often also their weights
- Typical approach:
 - ▶ Initialize a population of N random architectures
 - ▶ Sample N individuals from that population (with replacement) according to their fitness
 - ▶ Apply mutations to those N individuals to produce the next generation's population
 - ▶ Optionally: **elitism** to keep best individuals in the population

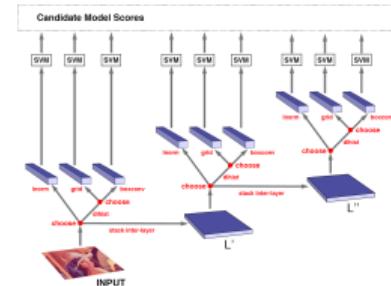
Early Work on Neuroevolution (already since the 1990s)

[Kitano. 1990; Angeline et al. 1994; Stanley and Miikkulainen. 2002; Bayer et al. 2009; Floreano et al. 2008]

- Evolves architectures & often also their weights
- Typical approach:
 - ▶ Initialize a population of N random architectures
 - ▶ Sample N individuals from that population (with replacement) according to their fitness
 - ▶ Apply mutations to those N individuals to produce the next generation's population
 - ▶ Optionally: **elitism** to keep best individuals in the population
- Mutations include adding, changing or removing a layer

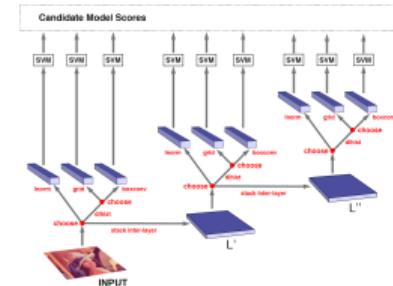
Early Work on Bayesian Optimization (since 2013)

- With TPE [Bergstra et al. 2011]:
 - Joint optimization of a vision architecture with 238 hyperparameters [Bergstra et al. 2013]
 - State-of-the-art performance on 3 disparate problems:
 - Face matching, face identification, and object recognition



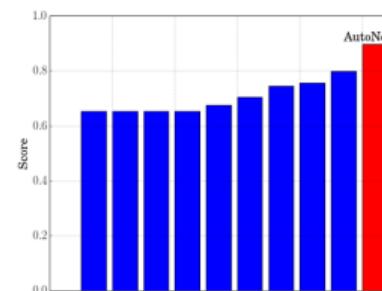
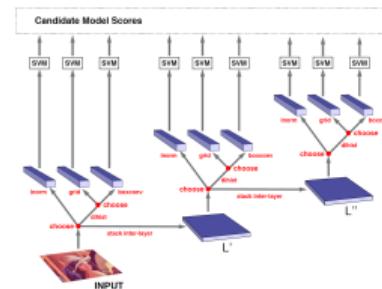
Early Work on Bayesian Optimization (since 2013)

- With TPE [Bergstra et al. 2011]:
 - Joint optimization of a vision architecture with 238 hyperparameters [Bergstra et al. 2013]
 - State-of-the-art performance on 3 disparate problems:
 - Face matching, face identification, and object recognition
- With SMAC [Hutter et al. 2011]:
 - New state-of-the-art performance on CIFAR-10 w/o data augmentation [Domhan et al. 2015]
 - Joint architecture and hyperparameter search, yielding Auto-Net [Mendoza et al. 2016]



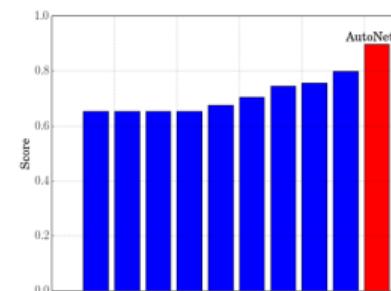
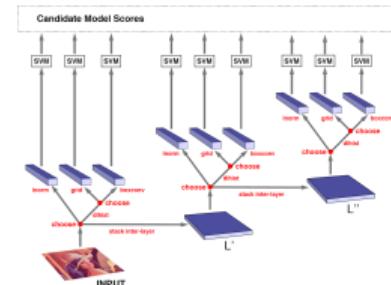
Early Work on Bayesian Optimization (since 2013)

- With TPE [Bergstra et al. 2011]:
 - Joint optimization of a vision architecture with 238 hyperparameters [Bergstra et al. 2013]
 - State-of-the-art performance on 3 disparate problems:
 - Face matching, face identification, and object recognition
 - With SMAC [Hutter et al. 2011]:
 - New state-of-the-art performance on CIFAR-10 w/o data augmentation [Domhan et al. 2015]
 - Joint architecture and hyperparameter search, yielding Auto-Net [Mendoza et al. 2016]
 - In 2015, Auto-Net already had several successes in ML competitions
 - E.g., human action recognition:
54491 data points, 5000 features, 18 classes
 - First automated deep learning (Auto-DL) method to win a machine learning competition dataset against human experts

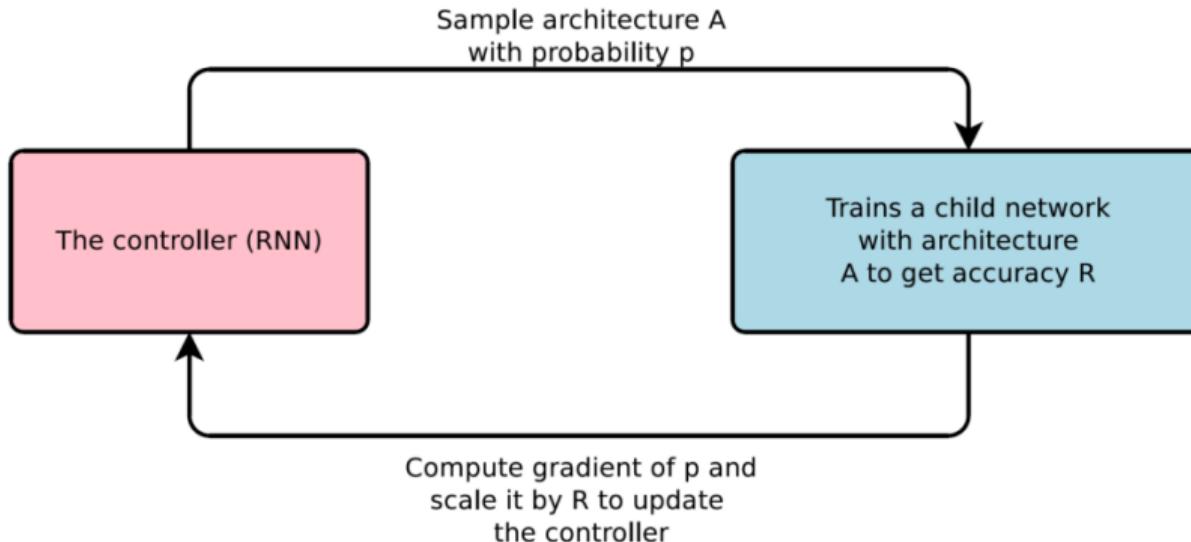


Early Work on Bayesian Optimization (since 2013)

- With TPE [Bergstra et al. 2011]:
 - Joint optimization of a vision architecture with 238 hyperparameters [Bergstra et al. 2013]
 - State-of-the-art performance on 3 disparate problems:
 - Face matching, face identification, and object recognition
- With SMAC [Hutter et al. 2011]:
 - New state-of-the-art performance on CIFAR-10 w/o data augmentation [Domhan et al. 2015]
 - Joint architecture and hyperparameter search, yielding Auto-Net [Mendoza et al. 2016]
 - In 2015, Auto-Net already had several successes in ML competitions
 - E.g., human action recognition:
54491 data points, 5000 features, 18 classes
 - First automated deep learning (Auto-DL) method to win a machine learning competition dataset against human experts
- With Gaussian processes:
 - Arc kernel [Swersky et al. 2013]

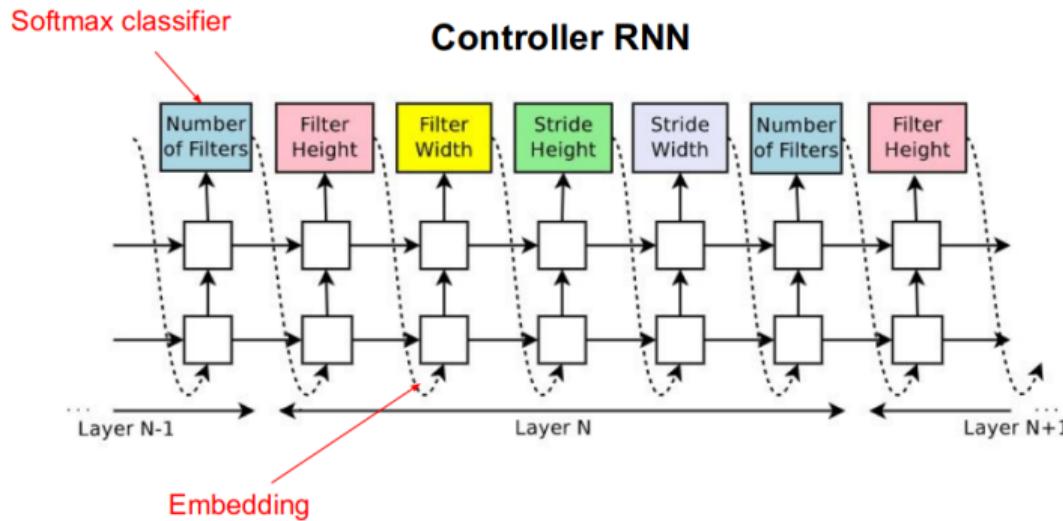


Reinforcement Learning [Zoph and Le. 2016]



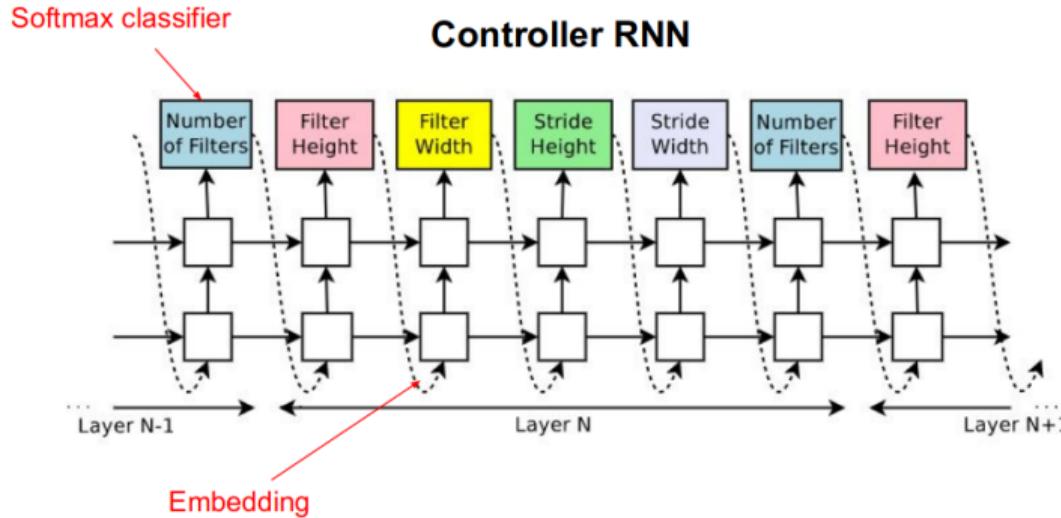
- Use RNN (“Controller”) to generate a NN architecture piece-by-piece
- Train this NN (“Child Network”) and evaluate it on a validation set
- Use Reinforcement Learning (RL) to update the parameters of the Controller RNN to optimize the performance of the child models

Learning CNNs with RL [Zoph and Le. 2016]



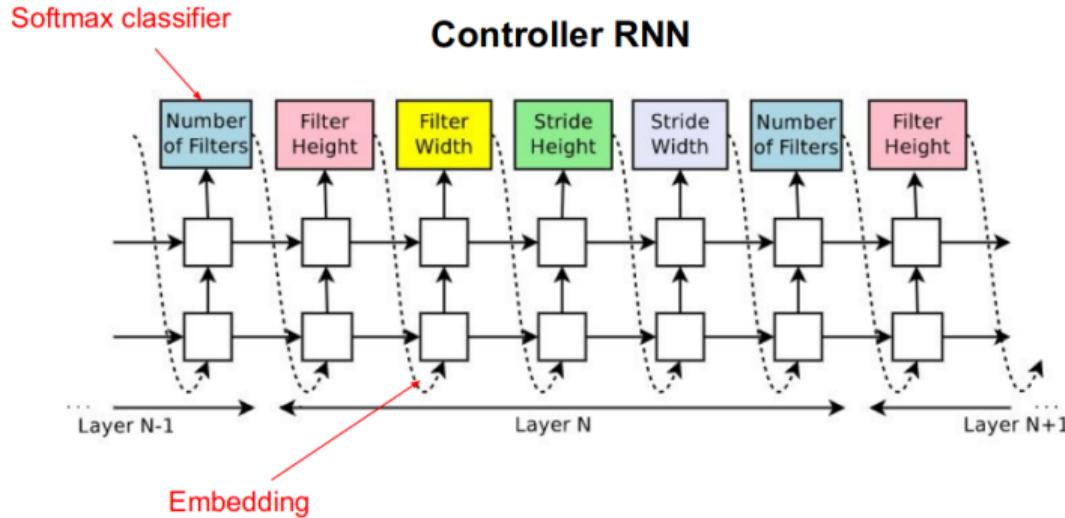
- For a fixed number of layers, select:
 - Filter width/height, stride width/height, number of filters

Learning CNNs with RL [Zoph and Le. 2016]



- For a fixed number of layers, select:
 - Filter width/height, stride width/height, number of filters
- Large computational demands (800 GPUs for 2 weeks, 12.800 architectures evaluated)

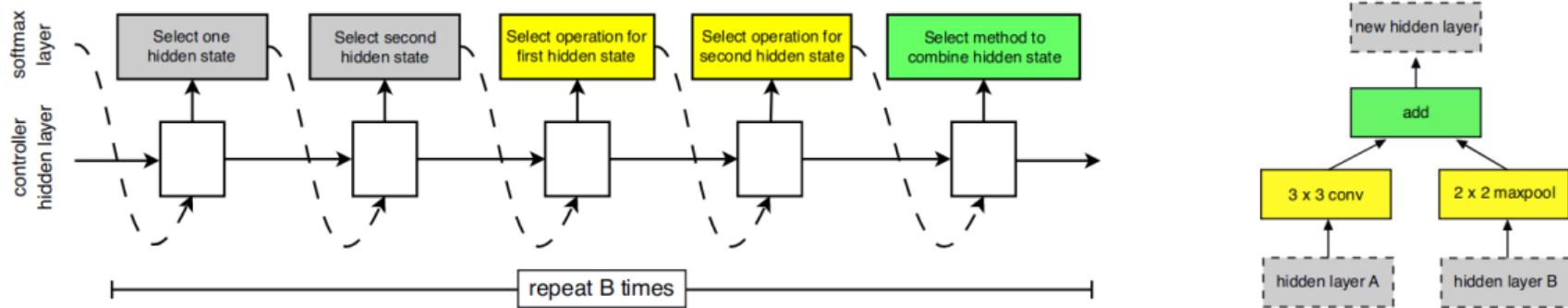
Learning CNNs with RL [Zoph and Le. 2016]



- For a fixed number of layers, select:
 - Filter width/height, stride width/height, number of filters
- Large computational demands (800 GPUs for 2 weeks, 12.800 architectures evaluated)
- State-of-the-art results for CIFAR-10 & Penn Treebank architecture
 - Brought NAS into the limelight

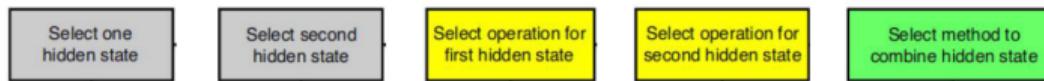
Learning CNN cells with RL [Zoph et al. 2018]

- 2 types of cells: normal and reduction cells
- For each type of cell: B blocks, each with 5 choices
 - Choose two previous feature maps (from this cell)
 - For each of these, choose an operation (3×3 conv, max-pool, etc.)
 - Choose a merge operation to combine the two results (concat or add)



Learning CNN cells with evolution [Real et al. 2018]

- 2 types of cells: normal and reduction cells
- For each type of cell: B blocks, each with 5 choices
 - Choose two previous feature maps (from this cell)
 - For each of these, choose an operation (3×3 conv, max-pool, etc.)
 - Choose a merge operation to combine the two results (concat or add)
- Evolution simply tackles this as a HPO problem with $2 \times 5 \times B$ variables:

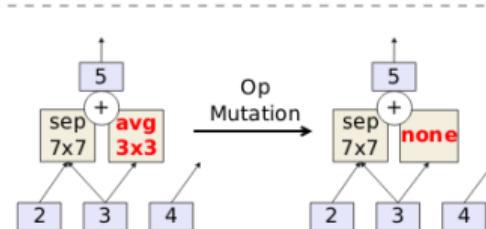
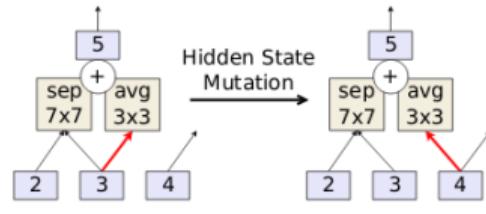


Regularized/Aging Evolution [Real et al. 2018]

- Quite standard evolutionary algorithm
 - ▶ But oldest solutions are dropped from population, instead of the worst

Regularized/Aging Evolution [Real et al. 2018]

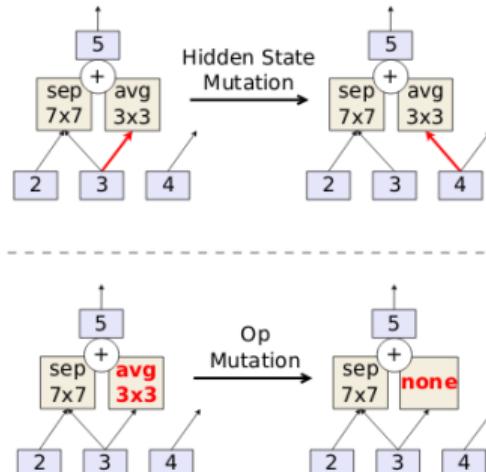
- Quite standard evolutionary algorithm
 - But oldest solutions are dropped from population, instead of the worst
- Standard SGD for training weights (optimizing the same blackbox as RL)
- Same fixed-length (HPO) search space as used for RL [Zoph et al. 2018]



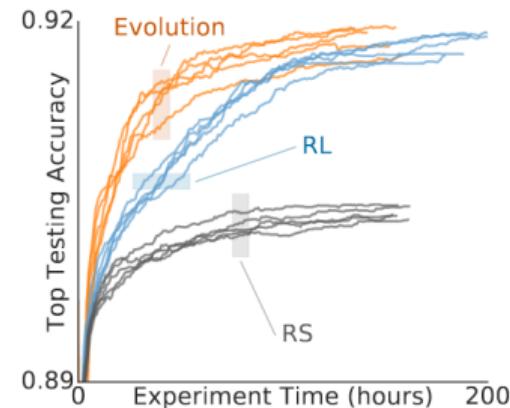
Different types of mutations in cell search space

Regularized/Aging Evolution [Real et al. 2018]

- Quite standard evolutionary algorithm
 - But oldest solutions are dropped from population, instead of the worst
- Standard SGD for training weights (optimizing the same blackbox as RL)
- Same fixed-length (HPO) search space as used for RL [Zoph et al. 2018]



Different types of mutations in cell search space



State-of-the-art performance in apples-to-apples comparison
→ AmoebaNet

Bayesian Optimization (BO)

- Encode the architecture space by categorical hyperparameters (like regularized evolution)
- Strong performance with tree-based models
 - ▶ TPE [Bergstra et al. 2013]
 - ▶ SMAC [Domhan et al. 2015; Mendoza et al. 2016; Zela et al. 2018]

Bayesian Optimization (BO)

- Encode the architecture space by categorical hyperparameters (like regularized evolution)
- Strong performance with tree-based models
 - ▶ TPE [Bergstra et al. 2013]
 - ▶ SMAC [Domhan et al. 2015; Mendoza et al. 2016; Zela et al. 2018]
- Kernels for GP-based NAS
 - Arc kernel [Swersky et al. 2013]
 - NASBOT [Kandasamy et al. 2018]

Bayesian Optimization (BO)

- Encode the architecture space by categorical hyperparameters (like regularized evolution)
- Strong performance with tree-based models
 - ▶ TPE [Bergstra et al. 2013]
 - ▶ SMAC [Domhan et al. 2015; Mendoza et al. 2016; Zela et al. 2018]
- Kernels for GP-based NAS
 - Arc kernel [Swersky et al. 2013]
 - NASBOT [Kandasamy et al. 2018]
- There are also several recent promising BO approaches based on neural networks
 - BANANAS [White et al. 2019]

Bayesian Optimization (BO)

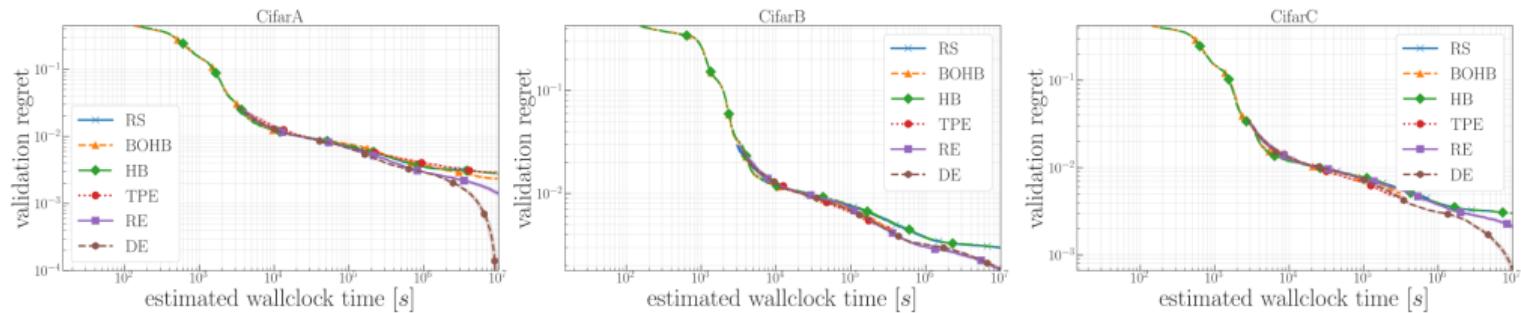
- Encode the architecture space by categorical hyperparameters (like regularized evolution)
- Strong performance with tree-based models
 - ▶ TPE [Bergstra et al. 2013]
 - ▶ SMAC [Domhan et al. 2015; Mendoza et al. 2016; Zela et al. 2018]
- Kernels for GP-based NAS
 - Arc kernel [Swersky et al. 2013]
 - NASBOT [Kandasamy et al. 2018]
- There are also several recent promising BO approaches based on neural networks
 - BANANAS [White et al. 2019]
- BO is very competitive, has been shown to outperform RL [Ying et al. 2019]

Current State of the Art: Differential Evolution

- Comprehensive experiments on a wide range of 12 different NAS benchmarks
[Awad et al. 2020]

Current State of the Art: Differential Evolution

- Comprehensive experiments on a wide range of 12 different NAS benchmarks
[Awad et al. 2020]
- Results:
 - ▶ Regularized evolution is very robust, typically amongst best of the methods discussed so far
 - ▶ Evolution variant of differential evolution is yet better; most efficient and robust method



Questions to Answer for Yourself / Discuss with Friends

- Repetition:
What are some pros and cons of using black-box optimizers for NAS?
- Repetition:
How can NAS be modelled as a HPO problem?
- Discussion:
Given enough resources, will blackbox NAS approaches always improve performance?
- Discussion:
Why does discarding the oldest individual (rather than the worst) help in regularized/aging evolution?
- Transfer:
How would you write NAS with the hierarchical search space as a HPO problem?

AutoML: Neural Architecture Search (NAS) Speedup Techniques

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Overview of NAS Speedup Methods

- Multi-fidelity optimization
- Learning curve prediction
- Meta-learning across datasets
- Network morphisms & weight inheritance
- Weight sharing & the one-shot model

NAS Speedup Technique 1: Multi-fidelity optimization

- Analogous to multi-fidelity optimization in HPO

- Many evaluations for cheaper fidelities (less epochs, smaller datasets, down-sampled images, shallower networks, etc)
- Fewer evaluations necessary for more expensive fidelities

NAS Speedup Technique 1: Multi-fidelity optimization

- Analogous to multi-fidelity optimization in HPO

- Many evaluations for cheaper fidelities (less epochs, smaller datasets, down-sampled images, shallower networks, etc)
- Fewer evaluations necessary for more expensive fidelities

- Compatible with any blackbox optimization method

- Using random search: ASHA [Li and Talwalkar. 2019]
- Using Bayesian optimization: BOHB [Zela et al. 2018]
- Using differential evolution: DEHB [Awad et al. under review]
- Using regularized evolution: progressive dynamic hurdles [So et al. 2019]

NAS Speedup Technique 1: Multi-fidelity optimization

- Analogous to multi-fidelity optimization in HPO
 - Many evaluations for cheaper fidelities (less epochs, smaller datasets, down-sampled images, shallower networks, etc)
 - Fewer evaluations necessary for more expensive fidelities
- Compatible with any blackbox optimization method
 - Using random search: ASHA [Li and Talwalkar. 2019]
 - Using Bayesian optimization: BOHB [Zela et al. 2018]
 - Using differential evolution: DEHB [Awad et al. under review]
 - Using regularized evolution: progressive dynamic hurdles [So et al. 2019]
- Often used for joint optimization of architecture & hyperparameters
 - Auto-Pytorch [Mendoza et al. 2019; Zimmer et al. 2020]
 - “Auto-RL” [Runge et al. 2019]

NAS Speedup Technique 2: Learning Curve Prediction

- Analogous to learning curve prediction in HPO
 - Observe initial learning curve and predict performance at the end
 - Can use features of the architecture as input (just like hyperparameters as inputs)

NAS Speedup Technique 2: Learning Curve Prediction

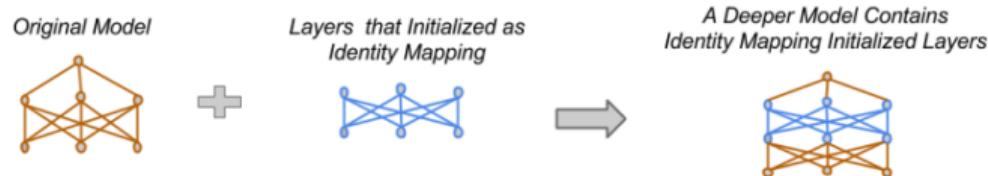
- Analogous to learning curve prediction in HPO
 - Observe initial learning curve and predict performance at the end
 - Can use features of the architecture as input (just like hyperparameters as inputs)
- Often used for joint optimization of architecture & hyperparameters
- Compatible with any blackbox optimization method
 - Using random search and Bayesian optimization: [Domhan et al. 2015]
 - Using reinforcement learning: [Baker et al. 2018]

NAS Speedup Technique 3: Meta-Learning

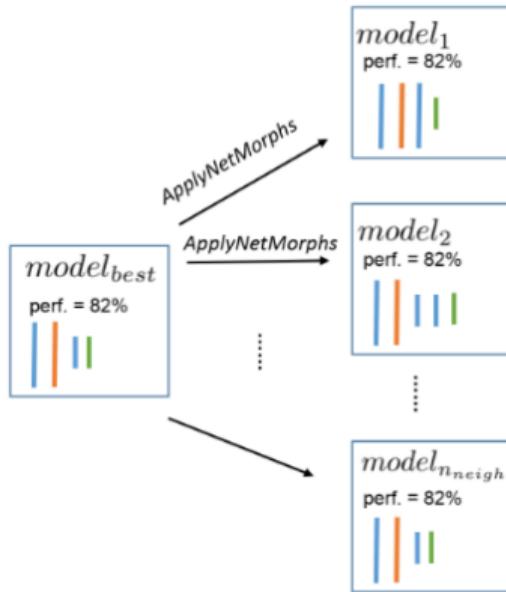
- Lots of work on meta-learning for HPO
- Only little work on meta-learning for NAS
 - Find a set of good architectures to initialize BOHB in Auto-Pytorch [Zimmer et al. 2020]
 - Learn RL agent's policy network on previous datasets [Wong et al. 2018]
 - Learn neural architecture that can be quickly adapted [Lian et al. 2019; Elsken et al. 2019]

NAS Speedup Technique 4: Network Morphisms

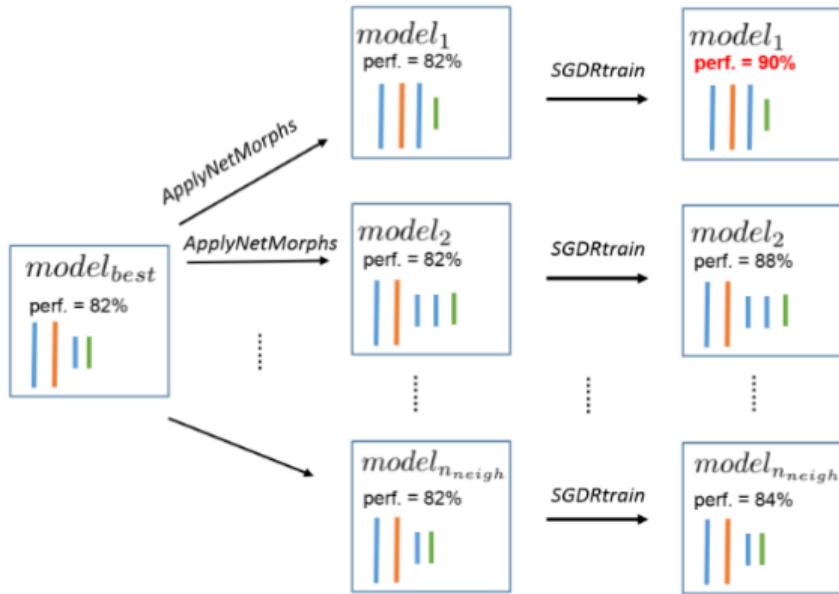
- **Network Morphisms** [Chen et al. 2016; Wei et al. 2016; Cai et al. 2017]
 - Change the network structure, but not the modelled function
 - I.e., for every input the network yields the same output as before applying the network morphisms operations
 - Examples: “Net2DeeperNet”, “Net2WiderNet”, etc.



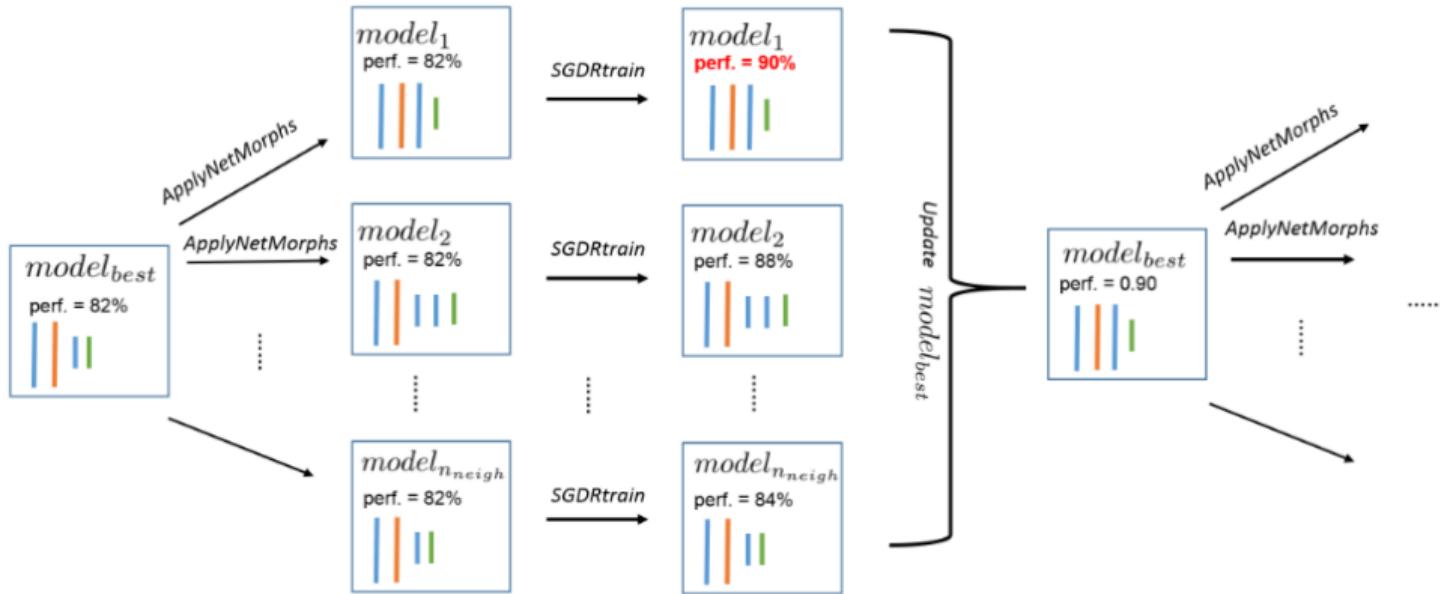
Network Morphisms Allow Efficient Moves in Architecture Space



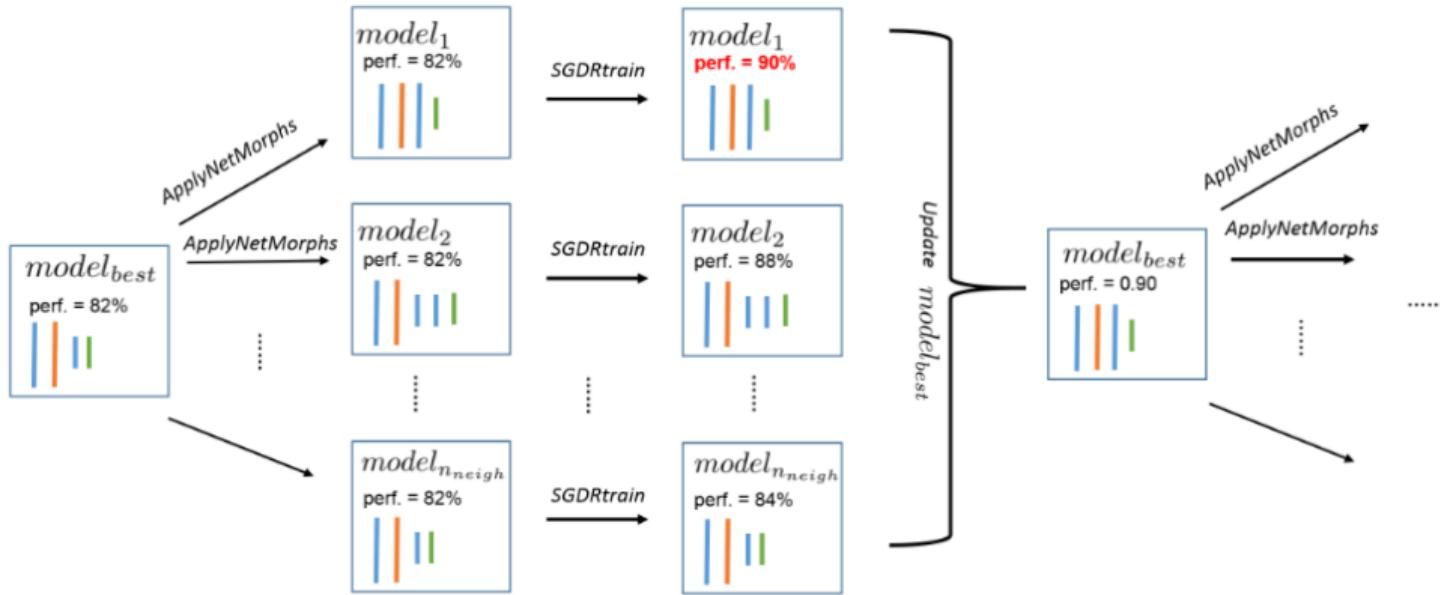
Network Morphisms Allow Efficient Moves in Architecture Space



Network Morphisms Allow Efficient Moves in Architecture Space



Network Morphisms Allow Efficient Moves in Architecture Space

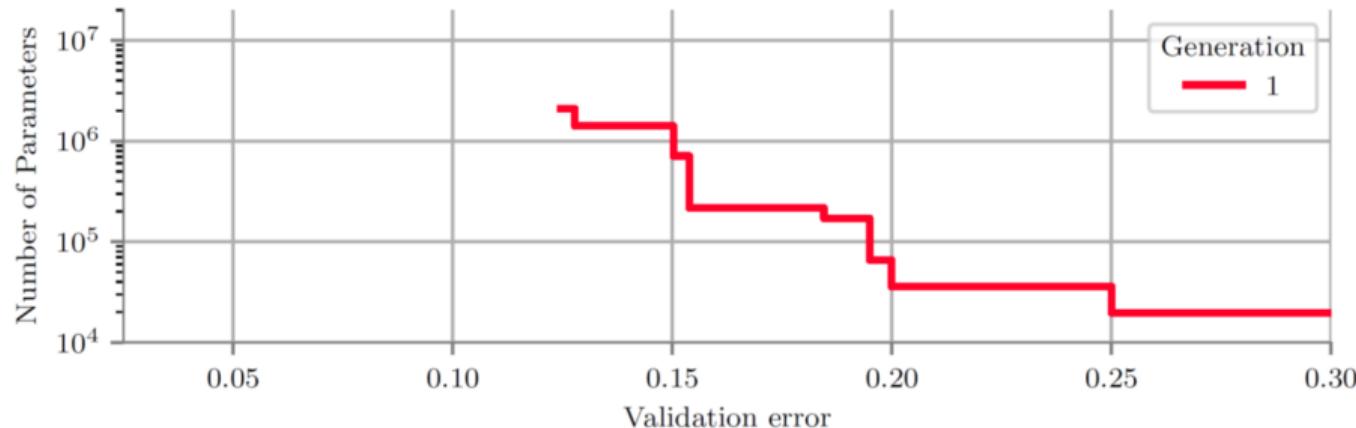


Weight inheritance avoids expensive retraining from scratch

[Real et al. 2017; Cai et al. 2018; Elsken et al. 2017; Cortes et al. 2017; Cai et al. 2018; Elsken et al. 2019]

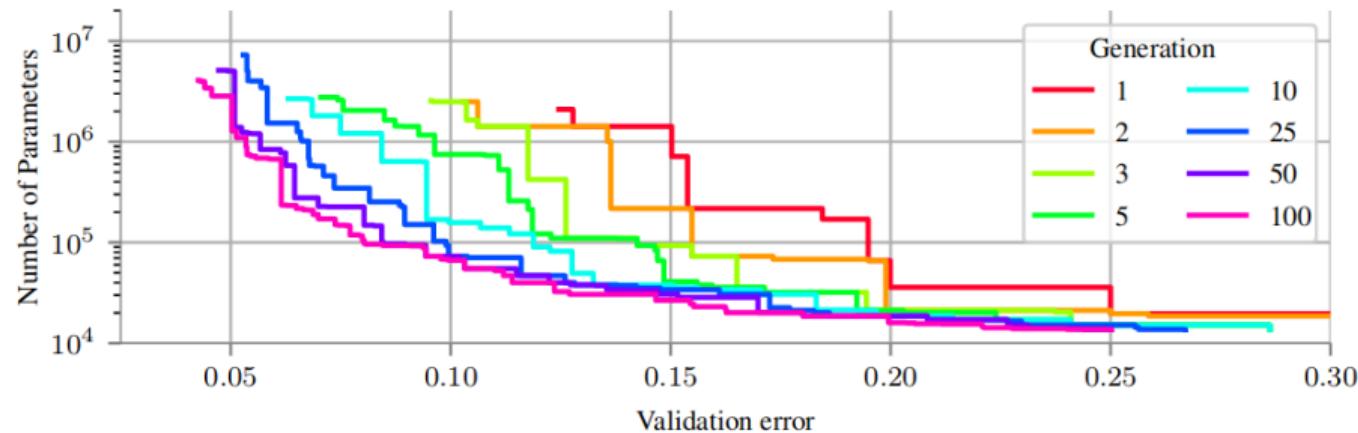
Network Morphisms for Multi-objective NAS [Elsken et al. 2019]

- To trade off error vs. resource consumption (e.g, #parameters):
 - ▶ Maintain a **Pareto front** of the two objectives
 - ▶ Evolve a population of Pareto-optimal architectures over time



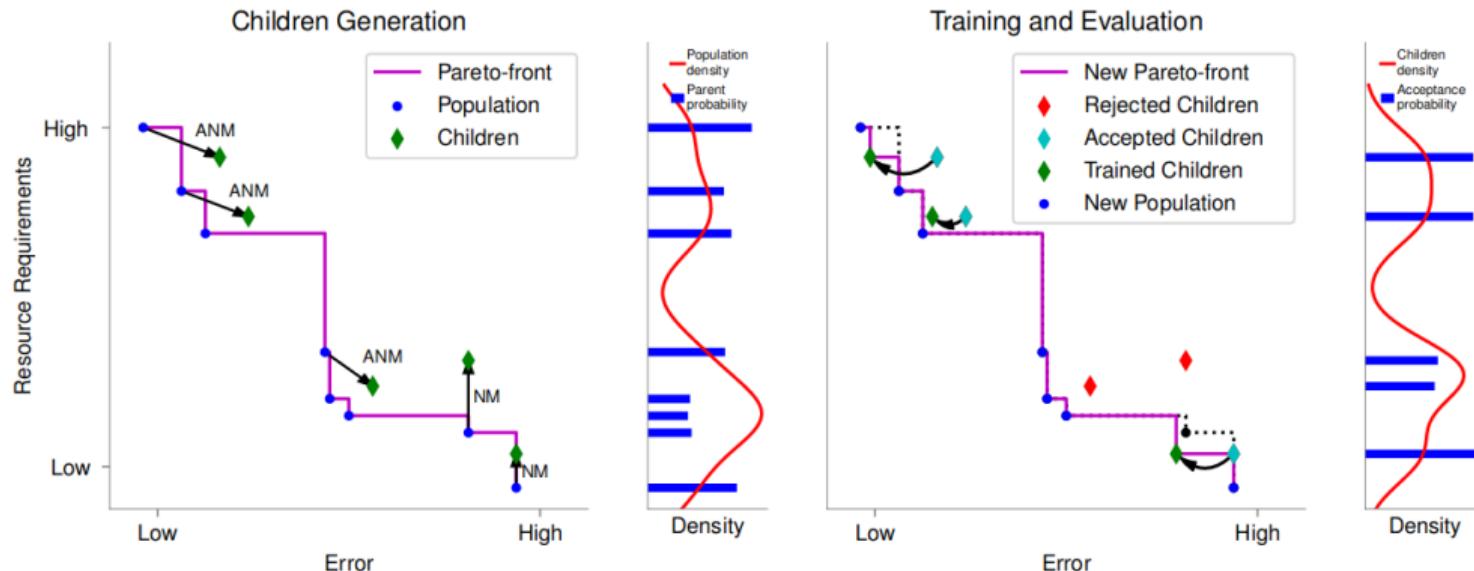
Network Morphisms for Multi-objective NAS [Elsken et al. 2019]

- To trade off error vs. resource consumption (e.g, #parameters):
 - ▶ Maintain a **Pareto front** of the two objectives
 - ▶ Evolve a population of Pareto-optimal architectures over time



Network Morphisms for Multi-objective NAS [Elsken et al. 2019]

- LEMONADE: Lamarckian Evolution for Multi-Objective Neural Architecture Design
- Weight inheritance through approximate morphisms (ANMs)
 - ▶ Dropping layers, dropping units within a layer, etc (function not preserved perfectly)



NAS Speedup Technique 5: Weight Sharing and One-shot Models

[Pham et al. 2018; Bender et al. 2018]

- All possible architectures are subgraphs of a large supergraph: the one-shot model

NAS Speedup Technique 5: Weight Sharing and One-shot Models

[Pham et al. 2018; Bender et al. 2018]

- All possible architectures are subgraphs of a large supergraph: the one-shot model
- Weights are shared between different architectures with common edges in the supergraph

NAS Speedup Technique 5: Weight Sharing and One-shot Models

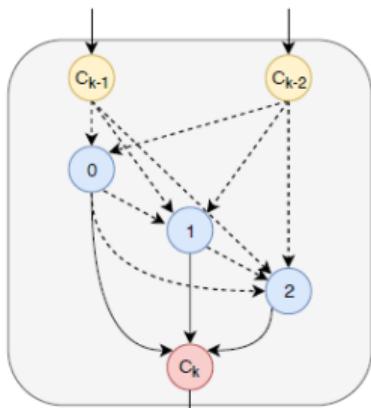
[Pham et al. 2018; Bender et al. 2018]

- All possible architectures are subgraphs of a large supergraph: the one-shot model
- Weights are shared between different architectures with common edges in the supergraph
- Search costs are reduced drastically since one only has to train a single model (the one-shot model).

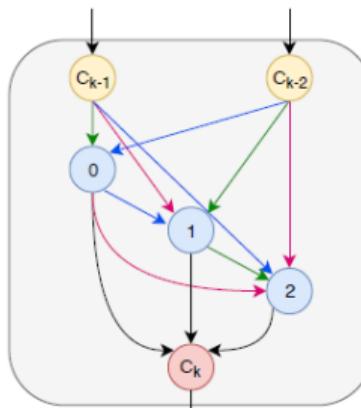
NAS Speedup Technique 5: Weight Sharing and One-shot Models

[Pham et al. 2018; Bender et al. 2018]

- The one-shot model can be seen as a directed acyclic multigraph
 - ⇒ Nodes - latent representations.
 - ⇒ Edges (dashed) - operations.



(a) One-shot search



(b) Final evaluation

- Architecture optimization problem: Find optimal path from the input to the output

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
List five methods to speed up NAS over blackbox approaches
- Repetition:
Which speedup techniques directly carry over from HPO to NAS?
- Discussion:
Why do network morphisms and the one-shot model only apply to NAS, and not to HPO?

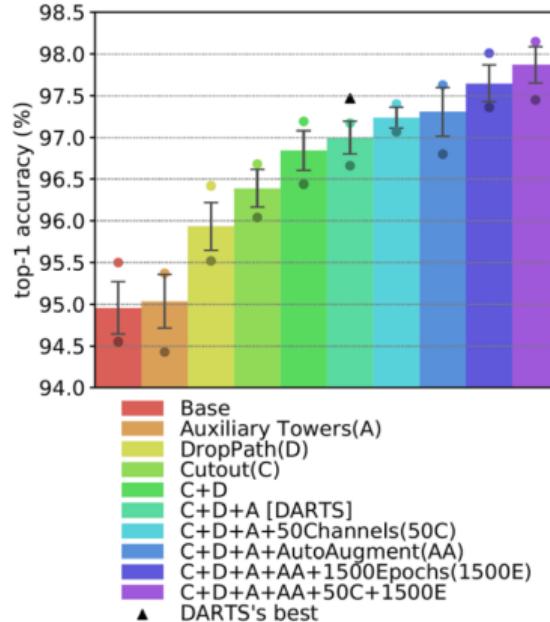
AutoML: Neural Architecture Search (NAS)

Issues and Best Practices in NAS Research

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Issues in NAS Research & Evaluations

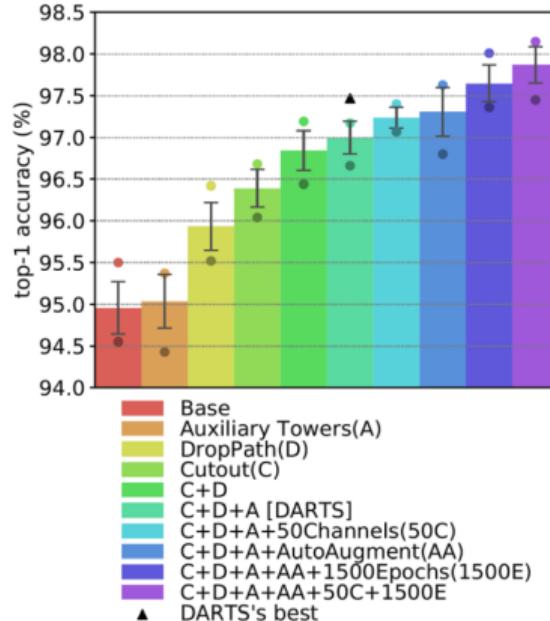
- Most NAS methods are **extremely difficult to reproduce and compare** [Li and Talwalkar. 2019]
- For benchmarks used in almost all NAS papers:
 - Training pipeline matters much more than neural architecture



[Yang et al. 2020]

Issues in NAS Research & Evaluations

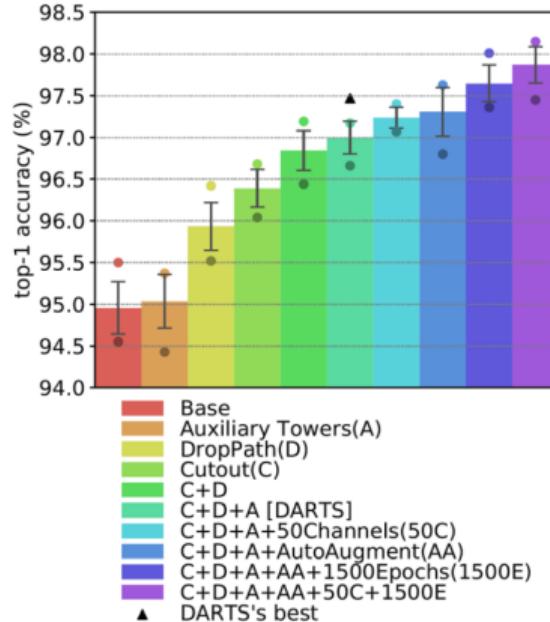
- Most NAS methods are extremely difficult to reproduce and compare [Li and Talwalkar. 2019]
- For benchmarks used in almost all NAS papers:
 - Training pipeline matters much more than neural architecture
- The final benchmark results reported in different papers are typically incomparable
 - Different training code (often unavailable)
 - Different search spaces
 - Different evaluation schemes



[Yang et al. 2020]

Issues in NAS Research & Evaluations

- Most NAS methods are extremely difficult to reproduce and compare [Li and Talwalkar. 2019]
 - For benchmarks used in almost all NAS papers:
 - Training pipeline matters much more than neural architecture
 - The final benchmark results reported in different papers are typically incomparable
 - Different training code (often unavailable)
 - Different search spaces
 - Different evaluation schemes
- We emphasize concepts, not published performance numbers



[Yang et al. 2020]

Building a Scientific Community for NAS

- **Benchmarks**

- NAS-Bench-101 [Ying et al. 2019]
- NAS-Bench-201 [Dong and Yang. 2020]
- NAS-Bench-1Shot1 [Zela et al. 2020]

Building a Scientific Community for NAS

- **Benchmarks**

- NAS-Bench-101 [Ying et al. 2019]
- NAS-Bench-201 [Dong and Yang. 2020]
- NAS-Bench-1Shot1 [Zela et al. 2020]

- **Best Practice Checklist for Scientific Research in NAS**

[Lindauer and Hutter. 2020]

Building a Scientific Community for NAS

- **Benchmarks**
 - NAS-Bench-101 [Ying et al. 2019]
 - NAS-Bench-201 [Dong and Yang. 2020]
 - NAS-Bench-1Shot1 [Zela et al. 2020]
- **Best Practice Checklist for Scientific Research in NAS**
[Lindauer and Hutter. 2020]
- **Unifying open-source implementation** of modern NAS algorithms
[Zela et al. 2020]
 - Finally enables empirical comparisons without confounding factors

Building a Scientific Community for NAS

- **Benchmarks**
 - NAS-Bench-101 [Ying et al. 2019]
 - NAS-Bench-201 [Dong and Yang. 2020]
 - NAS-Bench-1Shot1 [Zela et al. 2020]
- **Best Practice Checklist for Scientific Research in NAS**
[Lindauer and Hutter. 2020]
- **Unifying open-source implementation** of modern NAS algorithms
[Zela et al. 2020]
 - Finally enables empirical comparisons without confounding factors
- **First NAS workshop** at ICLR 2020

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for releasing code
 - ▶ Code for the training pipeline used to evaluate the final architectures
 - ▶ Hyperparameters used for the final evaluation pipeline, as well as random seeds
 - ▶ Code for the search space

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for releasing code
 - ▶ Code for the training pipeline used to evaluate the final architectures
 - ▶ Hyperparameters used for the final evaluation pipeline, as well as random seeds
 - ▶ Code for the search space
 - ▶ Code for your NAS method
 - ▶ Hyperparameters for your NAS method, as well as random seeds

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for releasing code
 - ▶ Code for the training pipeline used to evaluate the final architectures
 - ▶ Hyperparameters used for the final evaluation pipeline, as well as random seeds
 - ▶ Code for the search space
 - ▶ Code for your NAS method
 - ▶ Hyperparameters for your NAS method, as well as random seeds
- Note that the easiest way to satisfy the first three is to use existing NAS benchmarks

Definition: NAS Benchmark [Lindauer and Hutter, 2020]

A NAS benchmark consists of a dataset (with a predefined training-test split), a search space, and available runnable code with pre-defined hyperparameters for training the architectures.

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for comparing NAS methods
 - ▶ For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?

Best Practice Checklist for NAS Research [Lindauer and Hutter. 2020]

- Best practices for comparing NAS methods
 - ▶ For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
 - ▶ Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for comparing NAS methods
 - ▶ For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
 - ▶ Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
 - ▶ Did you run ablation studies?

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for comparing NAS methods
 - ▶ For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
 - ▶ Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
 - ▶ Did you run ablation studies?
 - ▶ Did you use the same evaluation protocol for the methods being compared?

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for comparing NAS methods
 - ▶ For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
 - ▶ Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
 - ▶ Did you run ablation studies?
 - ▶ Did you use the same evaluation protocol for the methods being compared?
 - ▶ Did you compare performance over time?

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for comparing NAS methods
 - ▶ For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
 - ▶ Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
 - ▶ Did you run ablation studies?
 - ▶ Did you use the same evaluation protocol for the methods being compared?
 - ▶ Did you compare performance over time?
 - ▶ Did you compare to random search?

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for comparing NAS methods
 - ▶ For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
 - ▶ Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
 - ▶ Did you run ablation studies?
 - ▶ Did you use the same evaluation protocol for the methods being compared?
 - ▶ Did you compare performance over time?
 - ▶ Did you compare to random search?
 - ▶ Did you perform multiple runs of your experiments and report seeds?

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for comparing NAS methods
 - ▶ For all NAS methods you compare, did you use exactly the same NAS benchmark, including the same dataset (with the same training-test split), search space and code for training the architectures and hyperparameters for that code?
 - ▶ Did you control for confounding factors (different hardware, versions of DL libraries, different runtimes for the different methods)?
 - ▶ Did you run ablation studies?
 - ▶ Did you use the same evaluation protocol for the methods being compared?
 - ▶ Did you compare performance over time?
 - ▶ Did you compare to random search?
 - ▶ Did you perform multiple runs of your experiments and report seeds?
 - ▶ Did you use tabular or surrogate benchmarks for in-depth evaluations?

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for reporting important details
 - ▶ Did you report how you tuned hyperparameters, and what time and resources this required?

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for reporting important details
 - ▶ Did you report how you tuned hyperparameters, and what time and resources this required?
 - ▶ Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)?

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for reporting important details
 - ▶ Did you report how you tuned hyperparameters, and what time and resources this required?
 - ▶ Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)?
 - ▶ Did you report all the details of your experimental setup?

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

- Best practices for reporting important details
 - ▶ Did you report how you tuned hyperparameters, and what time and resources this required?
 - ▶ Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)?
 - ▶ Did you report all the details of your experimental setup?
- It might not always be possible to satisfy all these best practices, but being aware of them is the first step ...

Best Practice Checklist for NAS Research [Lindauer and Hutter, 2020]

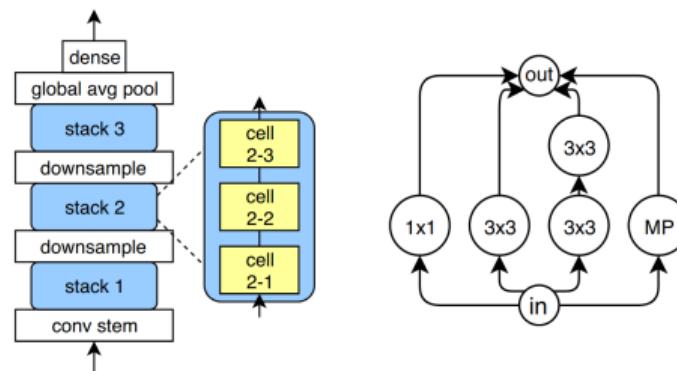
- Best practices for reporting important details
 - ▶ Did you report how you tuned hyperparameters, and what time and resources this required?
 - ▶ Did you report the time for the entire end-to-end NAS method (rather than, e.g., only for the search phase)?
 - ▶ Did you report all the details of your experimental setup?
- It might not always be possible to satisfy all these best practices, but being aware of them is the first step ...
- We believe the community would benefit a lot from:
 - ▶ Clean NAS benchmarks for new applications
 - ★ Including all details for the application. No need to also develop a new method.
 - ▶ Open-source library of NAS methods to compare methods without confounding factors
 - ★ First version already developed: NASlib [Zela et al, under review]

NAS-Bench-101: The First NAS Benchmark [Ying et al. 2019]

- Dataset: CIFAR-10, with the standard training/test split
- Runnable open-source code provided in Tensorflow
- Cell-structured search space consisting of all directed acyclic graphs (DAGs) on V nodes, where each possible node has L operation choices.

NAS-Bench-101: The First NAS Benchmark [Ying et al. 2019]

- Dataset: CIFAR-10, with the standard training/test split
- Runnable open-source code provided in Tensorflow
- Cell-structured search space consisting of all directed acyclic graphs (DAGs) on V nodes, where each possible node has L operation choices.
- To limit the number of architectures, NAS-Bench-101 has the following constraints:
 - ▶ $L = 3$ operators:
 - 3×3 convolution
 - 1×1 convolution
 - 3×3 max-pooling
 - ▶ $V \leq 7$ nodes
 - ▶ A maximum of 9 edges

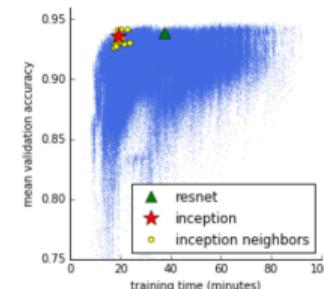
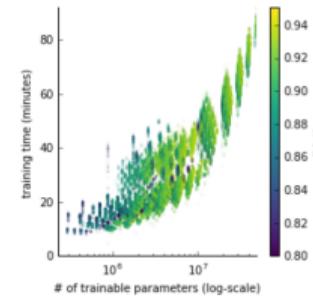


NAS-Bench-101: The First Tabular NAS Benchmark [Ying et al. 2019]

- **Tabular benchmark**: we exhaustively trained and evaluated all possible models on CIFAR-10 to create a tabular (look-up table) benchmark
- Based on this table, anyone can now run NAS experiments in seconds without a GPU.

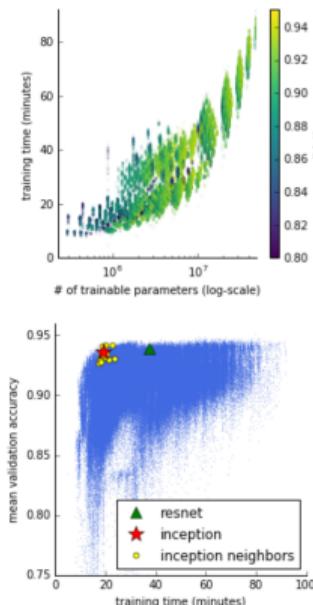
NAS-Bench-101: The First Tabular NAS Benchmark [Ying et al. 2019]

- **Tabular benchmark:** we exhaustively trained and evaluated all possible models on CIFAR-10 to create a tabular (look-up table) benchmark
- Based on this table, anyone can now run NAS experiments in seconds without a GPU.
- Around 423k **unique** cells
 - 4 epoch budgets: 4, 12, 36, 108
 - 3 repeats
 - around 5M trained and evaluated models
 - 120 TPU years of computation
 - the best architecture mean test accuracy: 94.32%



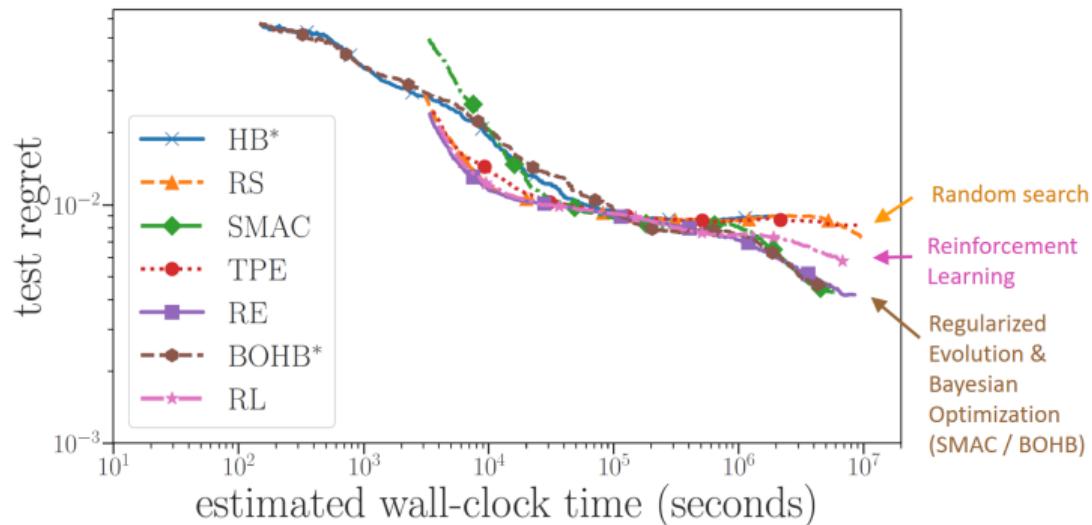
NAS-Bench-101: The First Tabular NAS Benchmark [Ying et al. 2019]

- **Tabular benchmark:** we exhaustively trained and evaluated all possible models on CIFAR-10 to create a tabular (look-up table) benchmark
- Based on this table, anyone can now run NAS experiments in seconds without a GPU.
- Around 423k **unique** cells
 - 4 epoch budgets: 4, 12, 36, 108
 - 3 repeats
 - around 5M trained and evaluated models
 - 120 TPU years of computation
 - the best architecture mean test accuracy: 94.32%
- Given an architecture encoding A , budget E_{stop} and trial number, one can query from NAS-Bench-101 the following quantities:
 - training/validation/test accuracy
 - training time in seconds
 - number of trainable model parameters



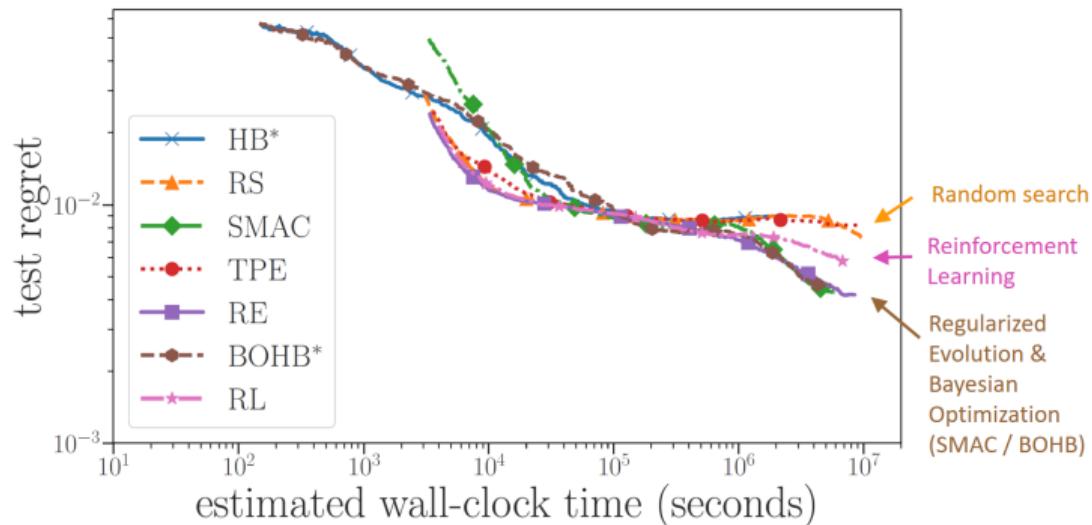
Evaluation of Blackbox NAS Methods on NAS-Bench-101 [Ying et al. 2019]

- RL outperforms random search
- BO and regularized evolution perform best, better than RL



Evaluation of Blackbox NAS Methods on NAS-Bench-101 [Ying et al. 2019]

- RL outperforms random search
- BO and regularized evolution perform best, better than RL



- Note that the BO method SMAC [Hutter et al. 2011] predicated RL for NAS [Zoph and Le. 2017] by 6 years
 - Only now, benchmarks like NAS-Bench-101 allow for efficient comparisons

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
For the most common NAS search space, how important is the NAS component compared to the importance of the training pipeline used?
- Repetition:
Why do we need proper benchmarking of NAS algorithms?
- Repetition:
What does a NAS benchmark consist of?
- Repetition:
List all best practices for NAS you remember.

AutoML: Neural Architecture Search (NAS)

The One-Shot Model

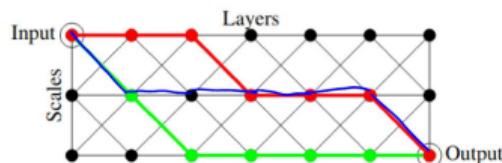
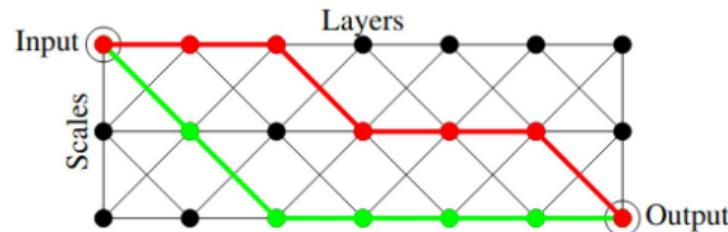
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

One-shot models: convolutional neural fabrics [Saxena and Verbeek. 2017]

- A **one-shot model** is a big model that has all architectures in a search space as submodels
 - ▶ This allows weights sharing across architectures
 - ▶ One **only needs to train the single one-shot model**, and implicitly trains an exponential number of individual architectures

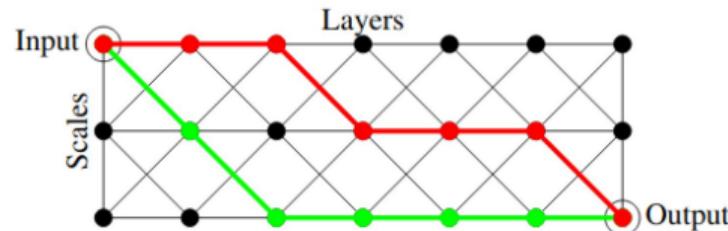
One-shot models: convolutional neural fabrics [Saxena and Verbeek. 2017]

- A **one-shot model** is a big model that has all architectures in a search space as submodels
 - ▶ This allows weights sharing across architectures
 - ▶ One **only needs to train the single one-shot model**, and implicitly trains an exponential number of individual architectures
 - The first type of one-shot models: **convolutional neural fabrics**



One-shot models: convolutional neural fabrics [Saxena and Verbeek. 2017]

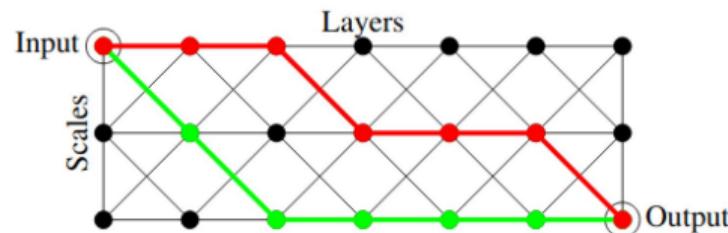
- A **one-shot model** is a big model that has all architectures in a search space as submodels
 - ▶ This allows weights sharing across architectures
 - ▶ One **only needs to train the single one-shot model**, and implicitly trains an exponential number of individual architectures
- The first type of one-shot models: **convolutional neural fabrics**



- ▶ Each path from the input to the output represents an architecture

One-shot models: convolutional neural fabrics [Saxena and Verbeek. 2017]

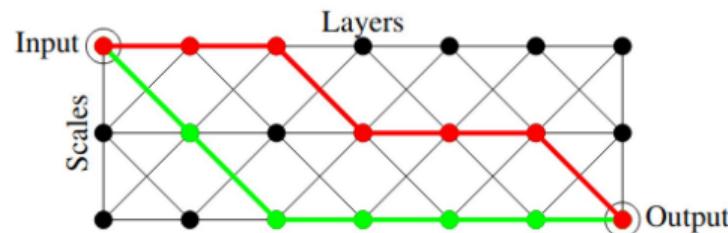
- A **one-shot model** is a big model that has all architectures in a search space as submodels
 - ▶ This allows weights sharing across architectures
 - ▶ One **only needs to train the single one-shot model**, and implicitly trains an exponential number of individual architectures
- The first type of one-shot models: **convolutional neural fabrics**



- ▶ Each path from the input to the output represents an architecture
- ▶ The **nodes** represent tensors
- ▶ The **edges** represent computations (e.g., convolution / strided convolution)

One-shot models: convolutional neural fabrics [Saxena and Verbeek. 2017]

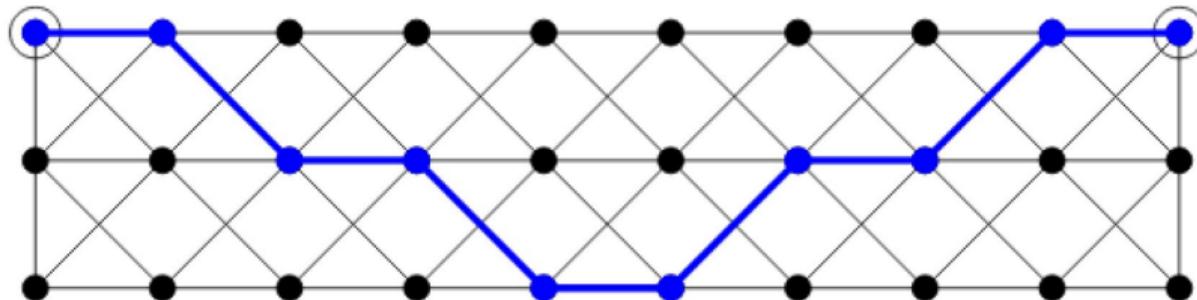
- A **one-shot model** is a big model that has all architectures in a search space as submodels
 - ▶ This allows weights sharing across architectures
 - ▶ One **only needs to train the single one-shot model**, and implicitly trains an exponential number of individual architectures
- The first type of one-shot models: **convolutional neural fabrics**



- ▶ Each path from the input to the output represents an architecture
- ▶ The nodes represent tensors
- ▶ The edges represent computations (e.g., convolution / strided convolution)
- ▶ Weights for the operation on an edge are shared across all (exponentially many) architectures that have that edge

One-shot models: convolutional neural fabrics [Saxena and Verbeek. 2017]

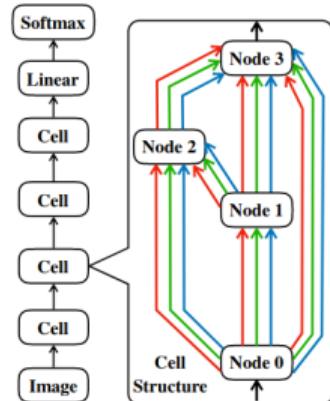
- A one-shot model is a big model that has all architectures in a search space as submodels
 - ▶ This allows weights sharing across architectures
 - ▶ One only needs to train the single one-shot model, and implicitly trains an exponential number of individual architectures
- The first type of one-shot models: convolutional neural fabrics



- ▶ Each path from the input to the output represents an architecture
- ▶ The nodes represent tensors
- ▶ The edges represent computations (e.g., convolution / strided convolution)
- ▶ Weights for the operation on an edge are shared
across all (exponentially many) architectures that have that edge

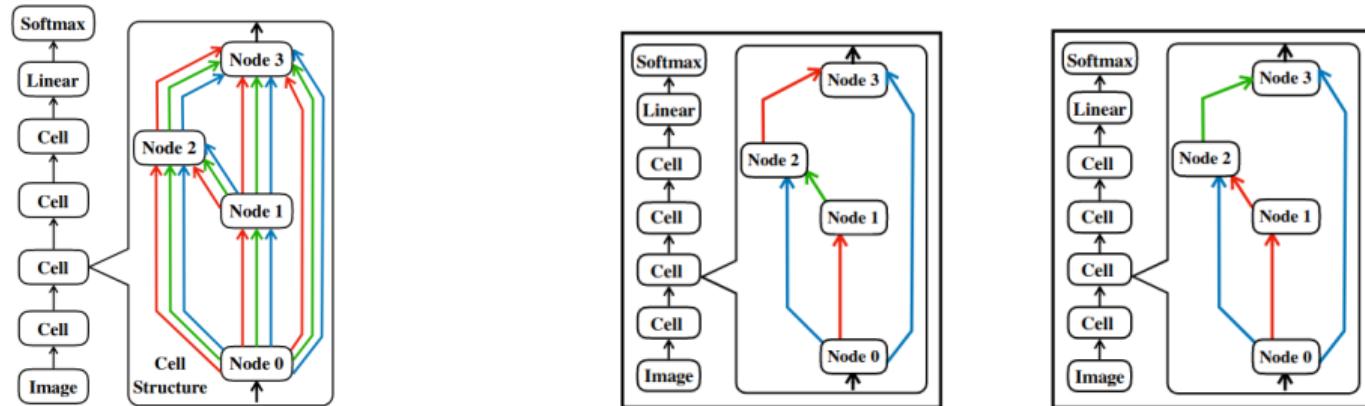
One-shot models for cell search spaces

- Directed acyclic multigraph to capture all (exponentially many) cell architectures
 - ▶ The nodes represent tensors
 - ▶ The edges represent computations (e.g., 3x3 conv, 5x5 conv, max pool, ...)
 - ▶ The results of operations on multiple edges between two nodes are combined (addition(concatenation))



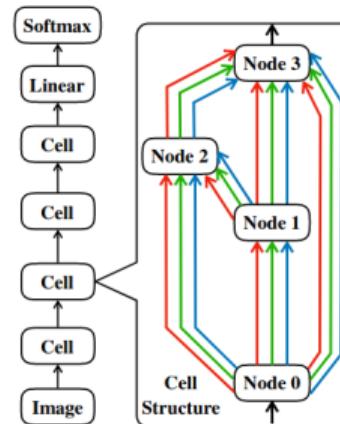
One-shot models for cell search spaces

- Directed acyclic multigraph to capture all (exponentially many) cell architectures
 - ▶ The nodes represent tensors
 - ▶ The edges represent computations (e.g., 3x3 conv, 5x5 conv, max pool, ...)
 - ▶ The results of operations on multiple edges between two nodes are combined (addition(concatenation))
- Individual architectures are subgraphs of this multigraph
 - ▶ Weights for the operation on an edge are shared across all (exponentially many) architectures that have that edge



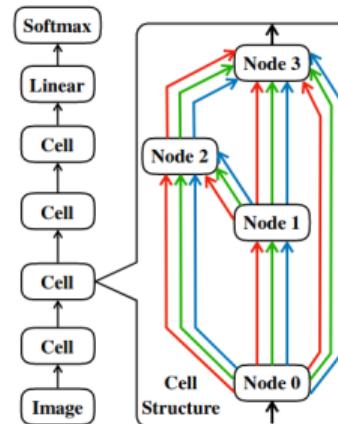
Training the one-shot model – standard SGD [Saxena and Verbeek. 2017]

- One-shot model is an acyclic graph; thus, backpropagation applies
 - ▶ Simplest method: standard training with SGD
 - ▶ This implicitly trains **an exponential number of architectures**



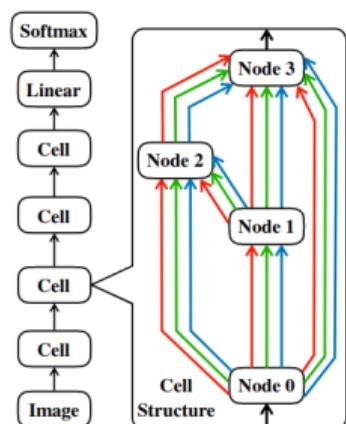
Training the one-shot model – standard SGD [Saxena and Verbeek. 2017]

- One-shot model is an acyclic graph; thus, backpropagation applies
 - ▶ Simplest method: standard training with SGD
 - ▶ This implicitly trains **an exponential number of architectures**
- Potential issue: co-adaptation of weights
 - ▶ Weights are implicitly optimized to work well on average across all architectures
 - ▶ They are **not** optimized specifically for the top-performing architecture



Training the one-shot model – DropPath [Bender et al. 2018]

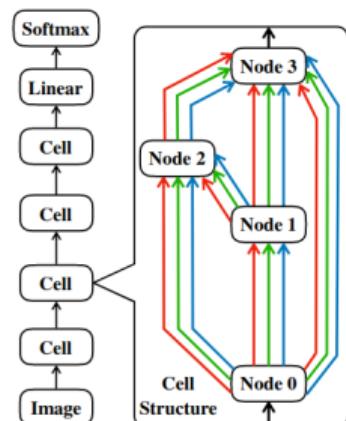
- To avoid coadaptation of weights, we can use **DropPath**, a technique analogous to **Dropout** [Srivastava et al., 2014]:



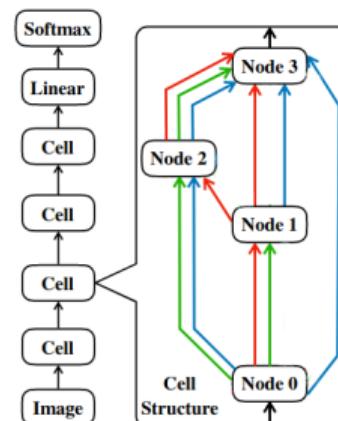
One-shot model

Training the one-shot model – DropPath [Bender et al. 2018]

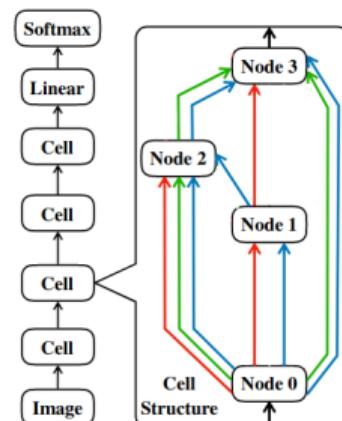
- To avoid coadaptation of weights, we can use **DropPath**, a technique analogous to **Dropout** [Srivastava et al., 2014]:
 - At each mini-batch iteration:
for each operation connecting 2 nodes, zero it out with probability p



One-shot model



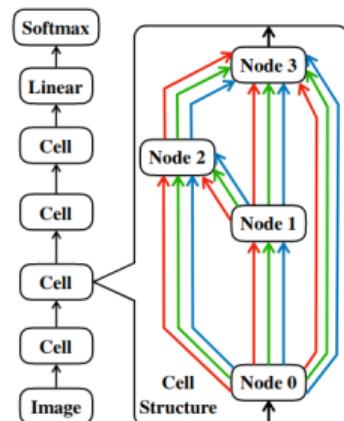
Architecture for batch 1



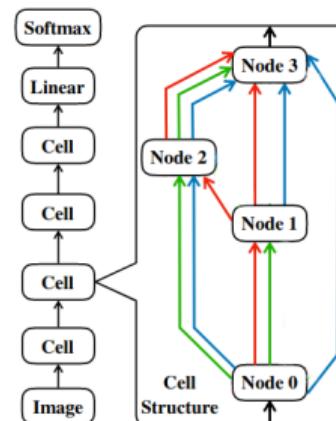
Architecture for batch 2

Training the one-shot model – DropPath [Bender et al. 2018]

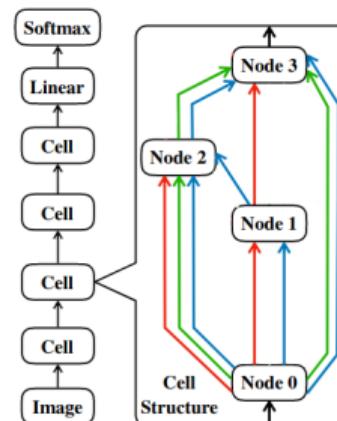
- To avoid coadaptation of weights, we can use **DropPath**, a technique analogous to **Dropout** [Srivastava et al., 2014]:
 - At each mini-batch iteration:
for each operation connecting 2 nodes, zero it out with probability p
 - **ScheduledDropPath**: starts with $p = 0$ and increases p linearly to p_{max} at the end of training



One-shot model



Architecture for batch 1



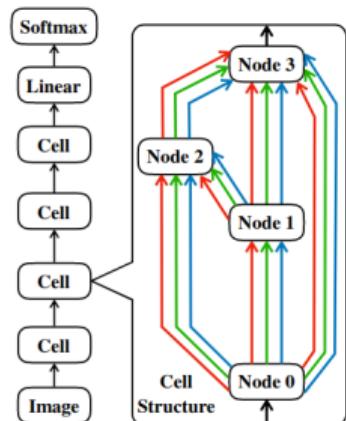
Architecture for batch 2

Training the one-shot model – Sampling

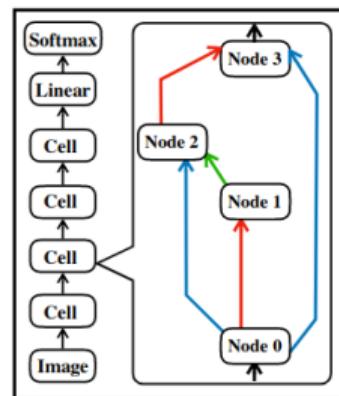
- At each mini-batch iteration during the training of the one-shot model sample a single architecture from the search space

Training the one-shot model – Sampling

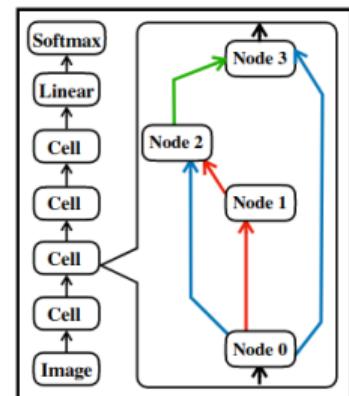
- At each mini-batch iteration during the training of the one-shot model sample a single architecture from the search space
- Update the parameters of the one-shot model corresponding to only that architecture



One-shot model



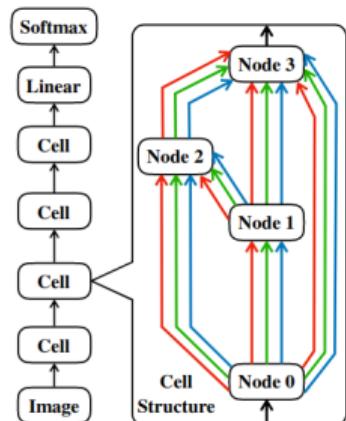
Architecture for batch 1



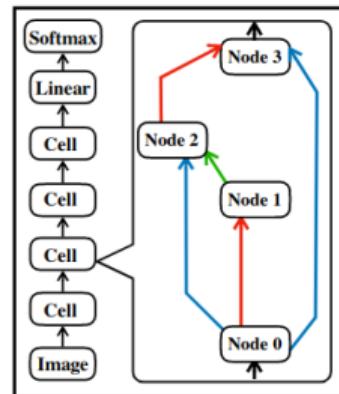
Architecture for batch 2

Training the one-shot model – Sampling

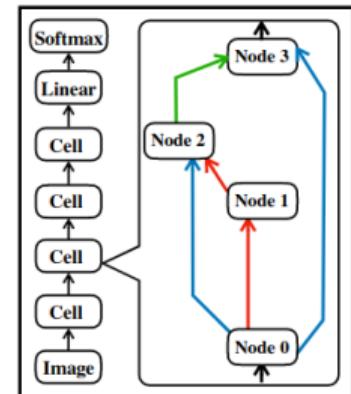
- At each mini-batch iteration during the training of the one-shot model **sample a single architecture** from the search space
 - Random Search with Weight Sharing** [Li and Talwalkar. 2020] → sample from uniform distribution
 - ENAS** [Pham et al. 2018] → sample from the learned policy of a RNN controller
- Update the parameters of the **one-shot model** corresponding to only that architecture



One-shot model



Architecture for batch 1



Architecture for batch 2

How to utilize the trained one-shot model?

- After training the one-shot model we have to **select the best individual architecture** from it
- There are multiple ways one can approach this. Some of these are:

How to utilize the trained one-shot model?

- After training the one-shot model we have to **select the best individual architecture** from it
- There are multiple ways one can approach this. Some of these are:
 1. Sample uniformly at random M architectures and rank them based on their validation error using the **one-shot model parameters**

How to utilize the trained one-shot model?

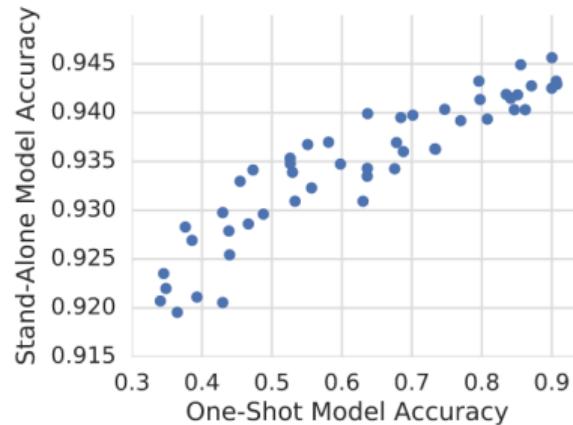
- After training the one-shot model we have to **select the best individual architecture** from it
- There are multiple ways one can approach this. Some of these are:
 1. Sample uniformly at random M architectures and rank them based on their validation error **using the one-shot model parameters**
 - 1b. (Optional) Select top K ($K < M$) and retrain them from scratch for a couple of epochs

How to utilize the trained one-shot model?

- After training the one-shot model we have to **select the best individual architecture** from it
- There are multiple ways one can approach this. Some of these are:
 1. Sample uniformly at random M architectures and rank them based on their validation error **using the one-shot model parameters**
 - 1b. (Optional) Select top K ($K < M$) and retrain them from scratch for a couple of epochs
 2. Return the top performing architecture to **retrain from scratch** for longer

How to utilize the trained one-shot model?

- After training the one-shot model we have to **select the best individual architecture** from it
- There are multiple ways one can approach this. Some of these are:
 - Sample uniformly at random M architectures and rank them based on their validation error **using the one-shot model parameters**
 - (Optional) Select top K ($K < M$) and retrain them from scratch for a couple of epochs
 - Return the top performing architecture to **retrain from scratch** for longer
- Pitfall:** the correlation between architectures evaluated with the one-shot weights and retrained from scratch (stand-alone models) should be high
- If not, **selecting the best architecture based on the one-shot weights** is sub-optimal.



From [Bender et al. 2018]

Questions to Answer for Yourself / Discuss with Friends

- Repetition:
[How are the weights shared in the one-shot model?](#)
- Repetition:
[What is the difference between Random Search with Weight Sharing and ENAS?](#)
- Discussion:
[What might be some downsides of using the one-shot model for NAS?](#)

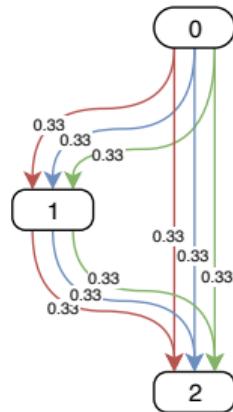
AutoML: Neural Architecture Search (NAS)

DARTS: Differentiable Architecture Search

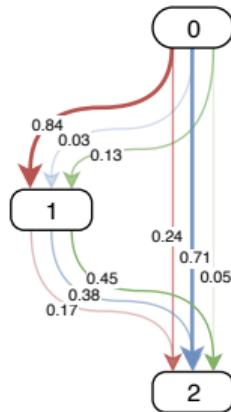
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

DARTS: Differentiable Architecture Search [Liu et al, 2018]

- Use one-shot model with continuous architecture weight α for each operator



(a) Initialization

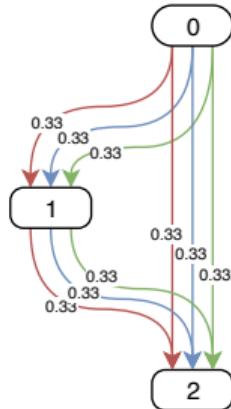


(b) Search end

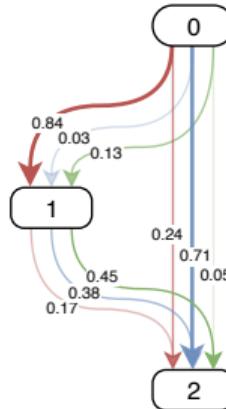
DARTS: Differentiable Architecture Search [Liu et al, 2018]

- Use one-shot model with continuous architecture weight α for each operator

$$x^{(j)} = \sum_{i < j} \tilde{o}^{(i,j)}(x^{(i)}) = \sum_{i < j} \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x^{(i)})$$



(a) Initialization

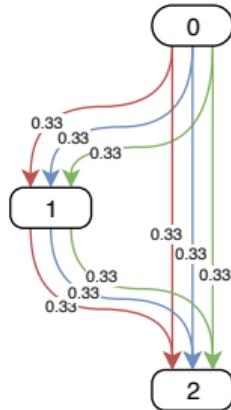


(b) Search end

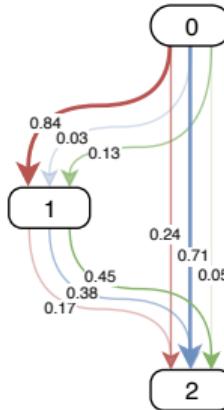
DARTS: Differentiable Architecture Search [Liu et al, 2018]

- Use one-shot model with continuous architecture weight α for each operator

$$x^{(j)} = \sum_{i < j} \tilde{o}^{(i,j)}(x^{(i)}) = \sum_{i < j} \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x^{(i)})$$



(a) Initialization



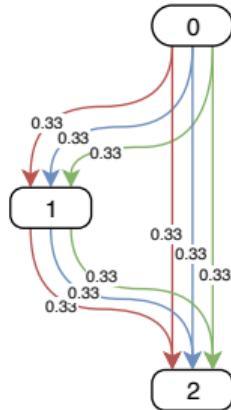
(b) Search end

- By optimizing the architecture weights α , DARTS assigns importance to each operation
 - ▶ Since the α are continuous, we can optimize them with gradient descent

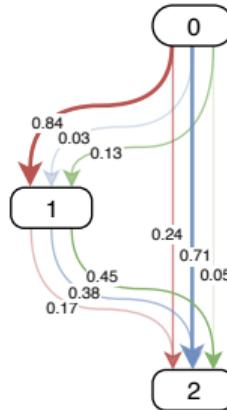
DARTS: Differentiable Architecture Search [Liu et al, 2018]

- Use one-shot model with continuous architecture weight α for each operator

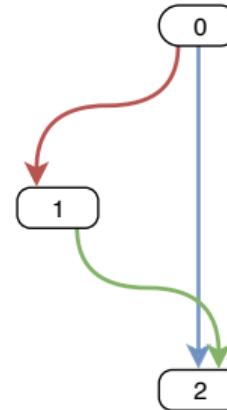
$$x^{(j)} = \sum_{i < j} \tilde{o}^{(i,j)}(x^{(i)}) = \sum_{i < j} \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x^{(i)})$$



(a) Initialization



(b) Search end



(c) Final cell

- By optimizing the architecture weights α , DARTS assigns importance to each operation
 - ▶ Since the α are continuous, we can optimize them with gradient descent
- In the end, DARTS discretizes to obtain a single architecture (c)

DARTS: Architecture Optimization

- The optimization problem ($a \rightarrow b$) is a bi-level optimization problem:

$$\begin{aligned} & \min_{\alpha} \mathcal{L}_{\text{val}}(w^*(\alpha), \alpha) \\ \text{s.t. } & w^*(\alpha) \in \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w, \alpha) \end{aligned}$$

DARTS: Architecture Optimization

- The optimization problem ($a \rightarrow b$) is a bi-level optimization problem:

$$\begin{aligned} & \min_{\alpha} \mathcal{L}_{\text{val}}(w^*(\alpha), \alpha) \\ \text{s.t. } & w^*(\alpha) \in \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w, \alpha) \end{aligned}$$

- This is solved using alternating SGD steps on architectural parameters α and weights w

Algorithm: DARTS 1st order

while *not converged* **do**

Update one-shot weights w by $\nabla_w \mathcal{L}_{\text{train}}(w, \alpha)$

Update architectural parameters α by $\nabla_\alpha \mathcal{L}_{\text{valid}}(w, \alpha)$

return $\arg \max_{o \in \mathcal{O}} \alpha_o^{(i,j)}$ for each edge (i, j)

DARTS: Architecture Optimization

- The optimization problem ($a \rightarrow b$) is a bi-level optimization problem:

$$\begin{aligned} & \min_{\alpha} \mathcal{L}_{\text{val}}(w^*(\alpha), \alpha) \\ \text{s.t. } & w^*(\alpha) \in \operatorname{argmin}_w \mathcal{L}_{\text{train}}(w, \alpha) \end{aligned}$$

- This is solved using alternating SGD steps on architectural parameters α and weights w

Algorithm: DARTS 1st order

while *not converged* **do**

Update one-shot weights w by $\nabla_w \mathcal{L}_{\text{train}}(w, \alpha)$

Update architectural parameters α by $\nabla_\alpha \mathcal{L}_{\text{valid}}(w, \alpha)$

return $\arg \max_{o \in \mathcal{O}} \alpha_o^{(i,j)}$ for each edge (i, j)

- Note: there is no theory showing that this process converges

Strong performance on some benchmarks

- E.g., original CNN search space
 - ▶ 8 operations on each MixedOp
 - ▶ 28 MixedOps in total
 - ▶ $> 10^{23}$ possible architectures
- Performance
 - ▶ < 3% error on CIFAR-10 in less than 1 GPU day of search

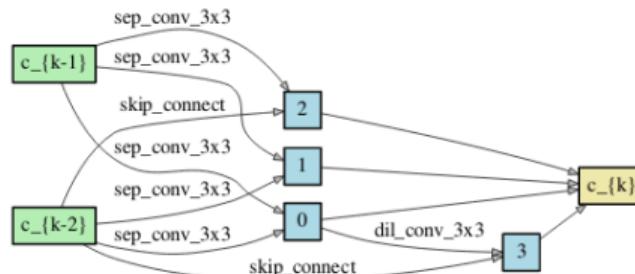


Figure 4: Normal cell learned on CIFAR-10.

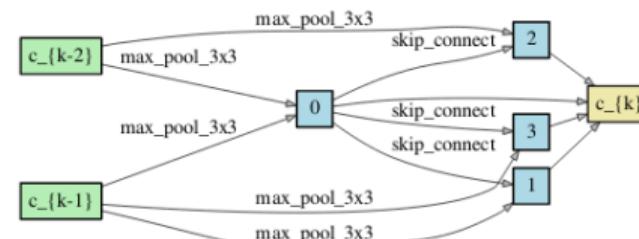


Figure 5: Reduction cell learned on CIFAR-10.

Issues – Non-robust behaviour

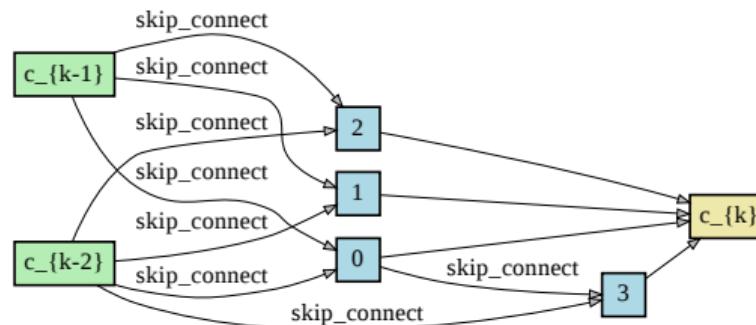
- DARTS is very sensitive w.r.t. its own hyperparameters (e.g. one-shot learning rate, L_2 regularization, etc.)

Issues – Non-robust behaviour

- DARTS is very sensitive w.r.t. its own hyperparameters (e.g. one-shot learning rate, L_2 regularization, etc.)
 - Tuning these hyperparameters for every new task/search space is computationally expensive

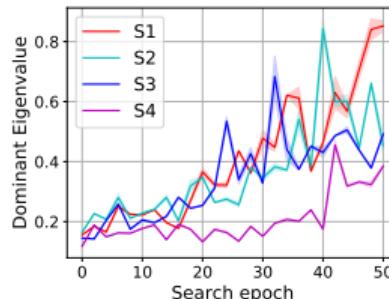
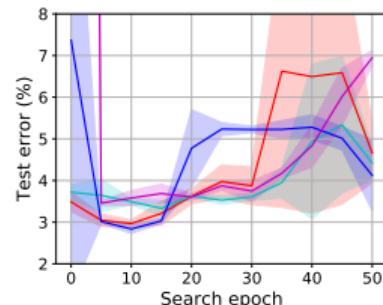
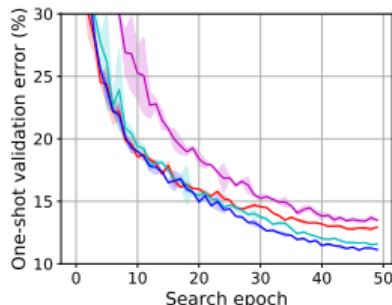
Issues – Non-robust behaviour

- DARTS is very sensitive w.r.t. its own hyperparameters (e.g. one-shot learning rate, L_2 regularization, etc.)
 - Tuning these hyperparameters for every new task/search space is computationally expensive
 - DARTS may return degenerate architectures, e.g., cells composed with only skip connections



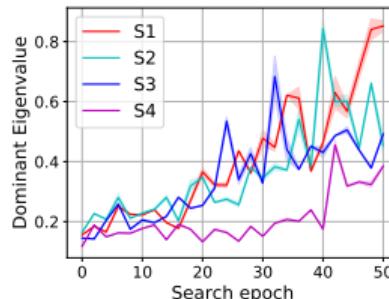
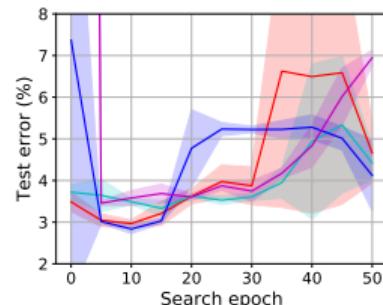
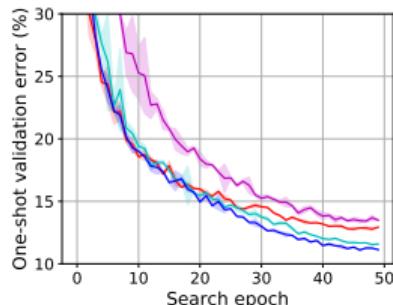
Issues – Non-robust behaviour

- DARTS is very sensitive w.r.t. its own hyperparameters (e.g. one-shot learning rate, L_2 regularization, etc.)
 - Tuning these hyperparameters for every new task/search space is computationally expensive
 - DARTS may return degenerate architectures, e.g., cells composed with only skip connections
- RobustDARTS [Zela et al, 2020] – tracks the curvature of the validation loss and stops the search early based on that



Issues – Non-robust behaviour

- DARTS is very sensitive w.r.t. its own hyperparameters (e.g. one-shot learning rate, L_2 regularization, etc.)
 - Tuning these hyperparameters for every new task/search space is computationally expensive
 - DARTS may return degenerate architectures, e.g., cells composed with only skip connections
- RobustDARTS [Zela et al, 2020] – tracks the curvature of the validation loss and stops the search early based on that
- SmoothDARTS [Chen and Hsieh, 2020] – applies random perturbation and adversarial training to avoid bad regions



Issues – Memory constraints

- DARTS keeps the [entire one-shot model in memory](#), together with its computed tensors

Issues – Memory constraints

- DARTS keeps the **entire one-shot model in memory**, together with its computed tensors
 - This constrains the search space size and the fidelity used to train the one-shot model
 - **Impossible** to run on large datasets as ImageNet

Issues – Memory constraints

- DARTS keeps the **entire one-shot model in memory**, together with its computed tensors
 - This constrains the search space size and the fidelity used to train the one-shot model
 - **Impossible** to run on large datasets as ImageNet

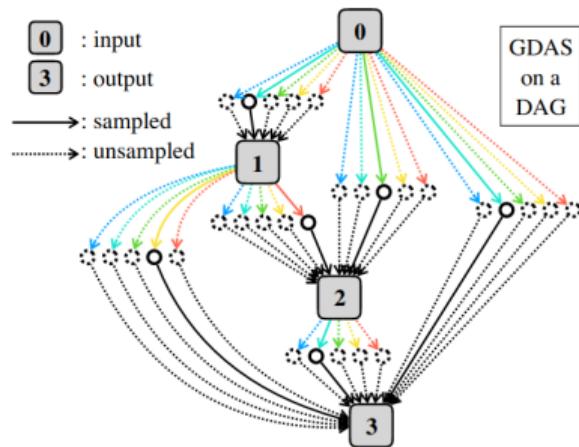
A lot of research aims to fix this issue:

Issues – Memory constraints

- DARTS keeps the **entire one-shot model in memory**, together with its computed tensors
 - This constrains the search space size and the fidelity used to train the one-shot model
 - **Impossible** to run on large datasets as ImageNet

A lot of research aims to fix this issue:

- **GDAS** [Dong et al, 2019] – samples from a Gumbel Softmax distribution to keep only a single architecture in memory

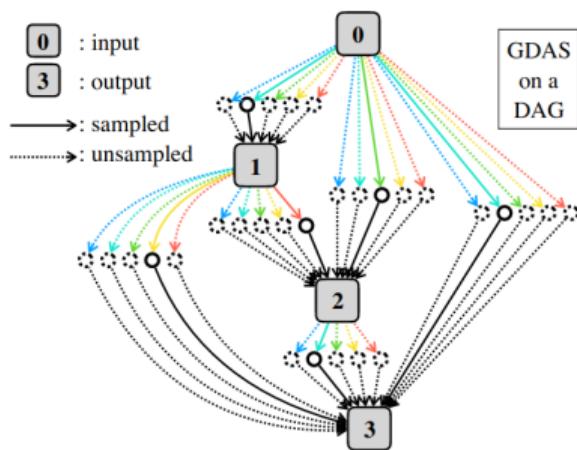


Issues – Memory constraints

- DARTS keeps the **entire one-shot model in memory**, together with its computed tensors
 - This constrains the search space size and the fidelity used to train the one-shot model
 - **Impossible** to run on large datasets as ImageNet

A lot of research aims to fix this issue:

- **GDAS** [Dong et al, 2019] – samples from a Gumbel Softmax distribution to keep only a single architecture in memory
- **ProxylessNAS** [Cai et al, 2019] – computes approximate gradients on α keeping only 2 edges between two nodes in memory at a time

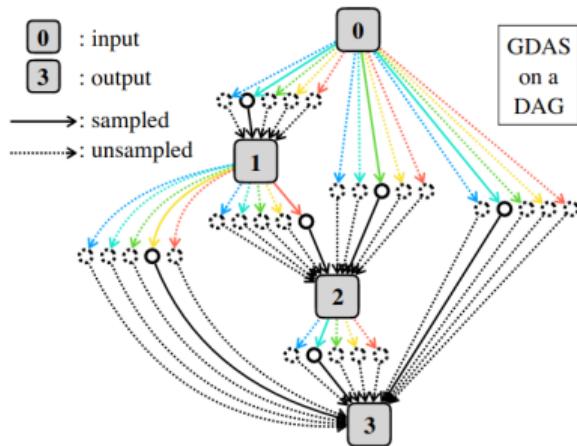


Issues – Memory constraints

- DARTS keeps the **entire one-shot model in memory**, together with its computed tensors
 - This constrains the search space size and the fidelity used to train the one-shot model
 - **Impossible** to run on large datasets as ImageNet

A lot of research aims to fix this issue:

- **GDAS** [Dong et al, 2019] – samples from a Gumbel Softmax distribution to keep only a single architecture in memory
- **ProxylessNAS** [Cai et al, 2019] – computes approximate gradients on α keeping only 2 edges between two nodes in memory at a time
- **PC-DARTS** [Xu et al, 2020] – performs the search on a subset of the channels in the one-shot model



Questions to Answer for Yourself / Discuss with Friends

- Repetition:
What is the main difference between DARTS and the other one-shot NAS methods we saw before?
- Repetition:
How does DARTS optimize the architectural weights and one-shot weights?
- Repetition:
What are DARTS' main issues and how can they be fixed?
- Discussion:
RobustDARTS stops the architecture optimization early, before the curvature of the validation loss becomes high. Why do you think this works?
[Hint: think about the discretization step after the DARTS search.]

AutoML: Neural Architecture Search (NAS)

NASLib: A Modular and Extensible NAS Library

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Motivation for NASLib [Zela et al. 2020]

NASLib is a **framework** for easily implementing different NAS methods, aiming to:

- Allow **fair comparisons without confounding factors**, which could be due to
 - Different codebases
 - Different search and evaluation pipelines
 - Different hyperparameter settings
 - Other confounding factors, e.g., library versions, GPU types, etc.

Motivation for NASLib [Zela et al. 2020]

NASLib is a **framework** for easily implementing different NAS methods, aiming to:

- Allow **fair comparisons without confounding factors**, which could be due to
 - Different codebases
 - Different search and evaluation pipelines
 - Different hyperparameter settings
 - Other confounding factors, e.g., library versions, GPU types, etc.
- Modularize different components of NAS optimizers to allow combining them

Motivation for NASLib [Zela et al. 2020]

NASLib is a **framework** for easily implementing different NAS methods, aiming to:

- Allow **fair comparisons without confounding factors**, which could be due to
 - Different codebases
 - Different search and evaluation pipelines
 - Different hyperparameter settings
 - Other confounding factors, e.g., library versions, GPU types, etc.
- Modularize different components of NAS optimizers to allow combining them
- Offer **researchers** a convenient way of prototyping new NAS methods

Motivation for NASLib [Zela et al. 2020]

NASLib is a **framework** for easily implementing different NAS methods, aiming to:

- Allow **fair comparisons without confounding factors**, which could be due to
 - Different codebases
 - Different search and evaluation pipelines
 - Different hyperparameter settings
 - Other confounding factors, e.g., library versions, GPU types, etc.
- Modularize different components of NAS optimizers to allow combining them
- Offer **researchers** a convenient way of prototyping new NAS methods
- Offer **users** reliable implementations of NAS methods
 - ▶ Facilitate the use of NAS for new search spaces
 - ▶ Develop a robust true AutoML framework

NASLib building blocks: Search Spaces, Optimizers, Evaluators

- NASLib implements a broad range of NAS optimizers
 - ▶ Blackbox NAS methods, e.g., Regularized Evolution
 - ▶ One-shot NAS methods, e.g., DARTS

NASLib building blocks: Search Spaces, Optimizers, Evaluators

- NASLib implements a broad range of NAS optimizers
 - ▶ Blackbox NAS methods, e.g., Regularized Evolution
 - ▶ One-shot NAS methods, e.g., DARTS
- The optimizers are modularized
 - ▶ This allows to switch from, e.g., DARTS to GDAS or PC-DARTS by just one method call

NASLib building blocks: Search Spaces, Optimizers, Evaluators

- NASLib implements a broad range of NAS optimizers
 - ▶ Blackbox NAS methods, e.g., Regularized Evolution
 - ▶ One-shot NAS methods, e.g., DARTS
- The optimizers are modularized
 - ▶ This allows to switch from, e.g., DARTS to GDAS or PC-DARTS by just one method call
- The evaluators are agnostic to the origin of an architecture
 - ▶ The final architecture is run using exactly the same object to evaluate on the test set

NASLib building blocks: Search Spaces, Optimizers, Evaluators

- NASLib implements a broad range of NAS optimizers
 - ▶ Blackbox NAS methods, e.g., Regularized Evolution
 - ▶ One-shot NAS methods, e.g., DARTS
- The optimizers are modularized
 - ▶ This allows to switch from, e.g., DARTS to GDAS or PC-DARTS by just one method call
- The evaluators are agnostic to the origin of an architecture
 - ▶ The final architecture is run using exactly the same object to evaluate on the test set
- NASLib's main building block is the graph object represented as a NetworkX¹ graph
 - Easily manipulate the graph by adding/removing nodes/edges
 - Hide complexity of dealing with the PyTorch computational graph
 - Easy high-level way of creating complex structures, e.g., hierarchical search spaces

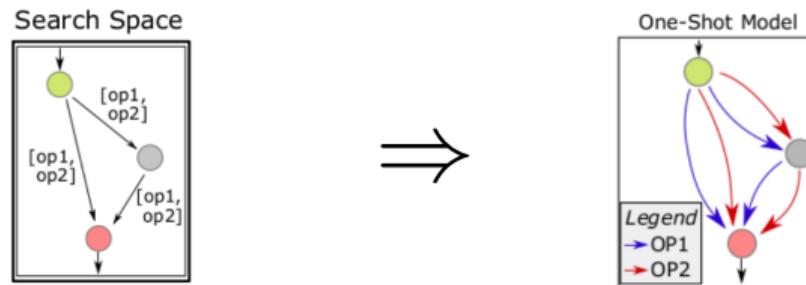
¹<https://networkx.github.io/>

NASLib building blocks: Search Spaces, Optimizers, Evaluators

- The optimizers are agnostic to the search space they are running on
 - ▶ This facilitates their use for new types of search spaces

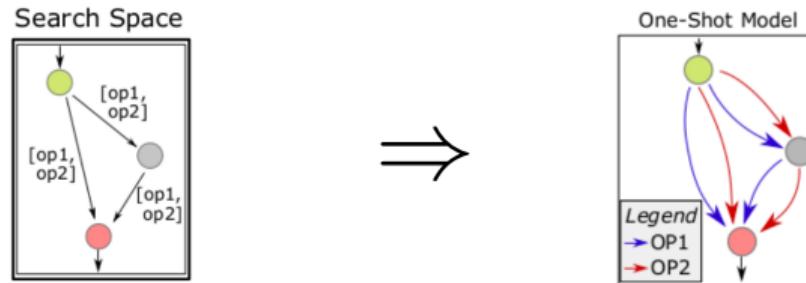
NASLib building blocks: Search Spaces, Optimizers, Evaluators

- The optimizers are agnostic to the search space they are running on
 - ▶ This facilitates their use for new types of search spaces
- An optimizer takes the search space as a NetworkX object and builds the PyTorch computational graph



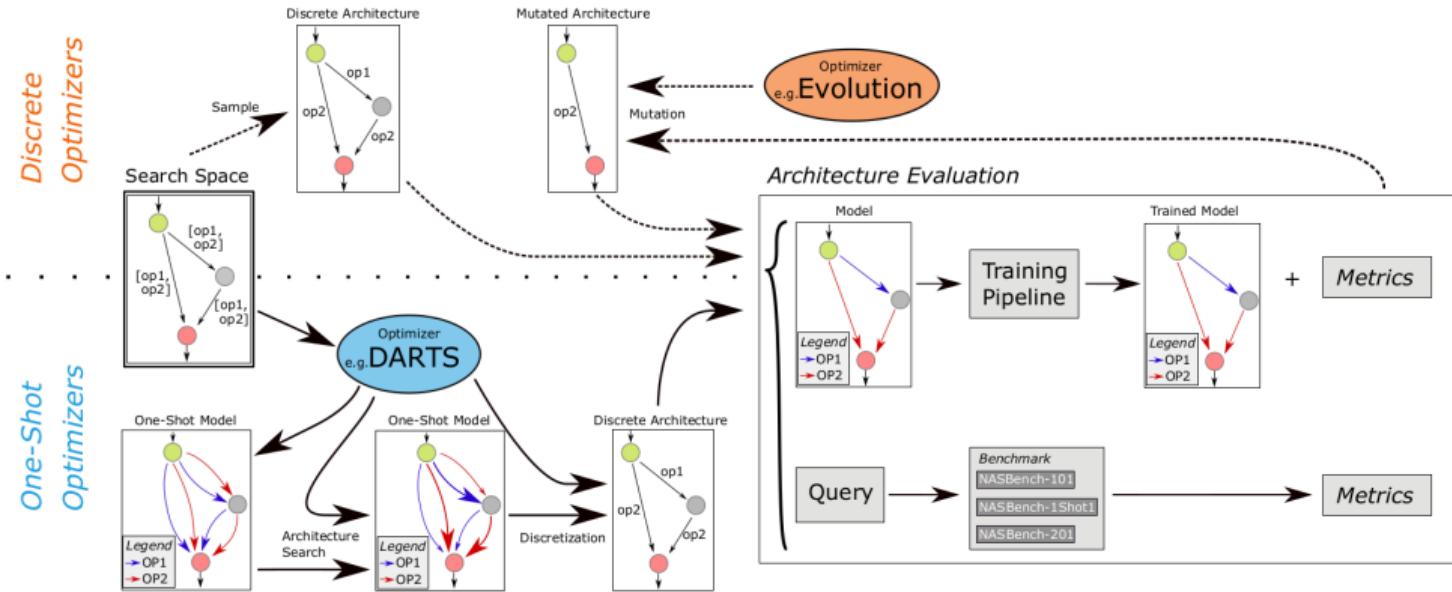
NASLib building blocks: Search Spaces, Optimizers, Evaluators

- The optimizers are agnostic to the search space they are running on
 - ▶ This facilitates their use for new types of search spaces
- An optimizer takes the search space as a NetworkX object and builds the PyTorch computational graph



- Depending on the optimizer, each operation choice in the NetworkX object becomes:
 - a MixedOp – for one-shot NAS optimizers, e.g. DARTS
 - a CategoricalOp – for black-box optimizers, e.g. Regularized Evolution

NASLib: Overview



Tabular benchmarks for one-shot NAS

- NAS-Bench-101 [Ying et al. 2019] is not directly compatible with one-shot NAS methods
 - ▶ Mainly due to the constraint of at most 9 edges in the cell

Tabular benchmarks for one-shot NAS

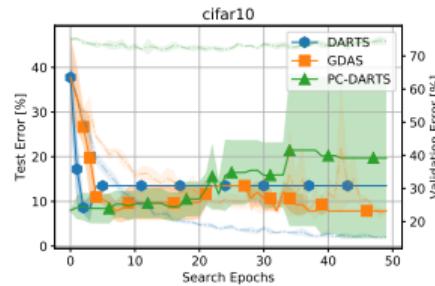
- NAS-Bench-101 [Ying et al. 2019] is not directly compatible with one-shot NAS methods
 - ▶ Mainly due to the constraint of at most 9 edges in the cell
- NAS-Bench-1Shot1 [Zela et al. 2020]
 - 3 sub-spaces of NAS-Bench-101 that are compatible with one-shot methods
 - ★ 6240, 29160, and 363648 architectures, respectively
 - Currently the largest one-shot NAS tabular benchmark

Tabular benchmarks for one-shot NAS

- NAS-Bench-101 [Ying et al. 2019] is not directly compatible with one-shot NAS methods
 - ▶ Mainly due to the constraint of at most 9 edges in the cell
- NAS-Bench-1Shot1 [Zela et al. 2020]
 - 3 sub-spaces of NAS-Bench-101 that are compatible with one-shot methods
 - ★ 6 240, 29 160, and 363 648 architectures, respectively
 - Currently the largest one-shot NAS tabular benchmark
- NAS-Bench-201 [Dong and Yang. 2020]
 - Much smaller than NAS-Bench-101 and largest NAS-Bench-1Shot1 subspace
 - ★ 15 625 architectures
 - Every architecture in the search space evaluated on 3 image classification datasets

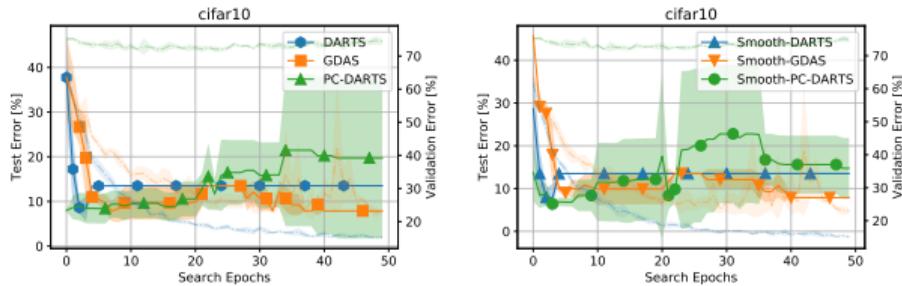
NASLib case study: Results on NAS-Bench-201

- NAS-Bench-201 is already integrated in NASLib and we can run any one-shot optimizer on it



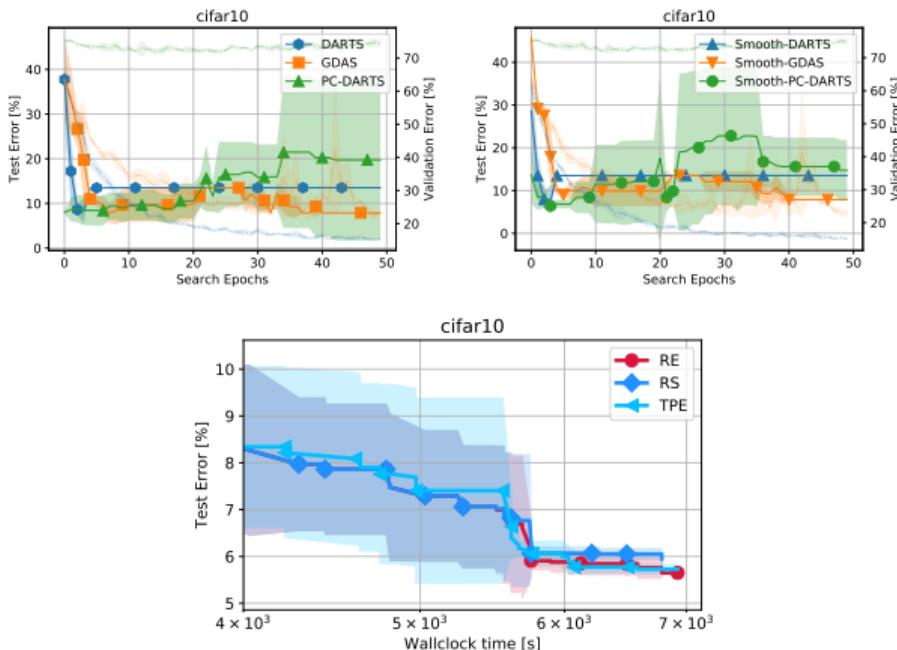
NASLib case study: Results on NAS-Bench-201

- NAS-Bench-201 is already integrated in NASLib and we can run any one-shot optimizer on it
- We can also combine random perturbations [Chen and Hsieh. 2020] with any one-shot optimizer



NASLib case study: Results on NAS-Bench-201

- NAS-Bench-201 is already integrated in NASLib and we can run any one-shot optimizer on it
- We can also combine random perturbations [Chen and Hsieh. 2020] with any one-shot optimizer
- We can also evaluate black-box optimizers cheaply with a tabular benchmark



Opportunities with NASLib

Room for many interesting projects and theses

- Applications of NAS to **your problem of interest**, with interesting search spaces
 - ▶ NASLib is the first library that separates the NAS method from the search spaces
 - ★ Therefore, **no changes are required in the NAS methods**
 - ★ This should make new applications much easier

Opportunities with NASLib

Room for many interesting projects and theses

- Applications of NAS to **your problem of interest**, with interesting search spaces
 - ▶ NASLib is the first library that separates the NAS method from the search spaces
 - ★ Therefore, **no changes are required in the NAS methods**
 - ★ This should make new applications much easier
- Studying **hierarchical search spaces** in detail

Opportunities with NASLib

Room for many interesting projects and theses

- Applications of NAS to **your problem of interest**, with interesting search spaces
 - ▶ NASLib is the first library that separates the NAS method from the search spaces
 - ★ Therefore, **no changes are required in the NAS methods**
 - ★ This should make new applications much easier
- Studying **hierarchical search spaces** in detail
- **Combining different components** of existing NAS methods
 - ▶ So far, it has been very hard on a code level to mix and match components
 - ▶ It ought to be possible to design the world's best NAS method by combining the right components

Opportunities with NASLib

Room for many interesting projects and theses

- Applications of NAS to your problem of interest, with interesting search spaces
 - ▶ NASLib is the first library that separates the NAS method from the search spaces
 - ★ Therefore, no changes are required in the NAS methods
 - ★ This should make new applications much easier
- Studying hierarchical search spaces in detail
- Combining different components of existing NAS methods
 - ▶ So far, it has been very hard on a code level to mix and match components
 - ▶ It ought to be possible to design the world's best NAS method by combining the right components

Room for interesting Hiwi projects

- Not everything is perfect yet, we can use lots of support by great programmers

Questions to Answer for Yourself / Discuss with Friends

- Repetition:

What would one have to do in order to apply the methods in NASLib to a new search space?

- Discussion:

Is there a problem of your interest that you would like to apply the methods in NASLib to?

- Discussion:

Given that NASLib's modular design allows mixing and matching components of one-shot NAS methods, which of the methods we discussed might make sense to combine?

NASLib: A Modular and Extensible NAS Library



AutoML: Neural Architecture Search (NAS)

Practical Recommendations for NAS and HPO

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Maturity of the Fields of NAS and HPO

- Hyperparameter optimization is a mature field
 - Blackbox HPO has been researched for decades; there are many software packages
 - Multi-fidelity HPO has also become quite mature

Maturity of the Fields of NAS and HPO

- Hyperparameter optimization is a mature field
 - Blackbox HPO has been researched for decades; there are many software packages
 - Multi-fidelity HPO has also become quite mature
- Neural architecture search is still a very young field

Maturity of the Fields of NAS and HPO

- Hyperparameter optimization is a mature field
 - Blackbox HPO has been researched for decades; there are many software packages
 - Multi-fidelity HPO has also become quite mature
- Neural architecture search is still a very young field
 - Blackbox is quite mature, but slow
 - Multi-fidelity NAS is also quite mature and faster

Maturity of the Fields of NAS and HPO

- Hyperparameter optimization is a mature field
 - Blackbox HPO has been researched for decades; there are many software packages
 - Multi-fidelity HPO has also become quite mature
- Neural architecture search is still a very young field
 - Blackbox is quite mature, but slow
 - Multi-fidelity NAS is also quite mature and faster
 - Meta-learning + multi-fidelity NAS is fast, but is still a very young field

Maturity of the Fields of NAS and HPO

- Hyperparameter optimization is a mature field
 - Blackbox HPO has been researched for decades; there are many software packages
 - Multi-fidelity HPO has also become quite mature
- Neural architecture search is still a very young field
 - Blackbox is quite mature, but slow
 - Multi-fidelity NAS is also quite mature and faster
 - Meta-learning + multi-fidelity NAS is fast, but is still a very young field
 - Gradient-based NAS is fast, but can have failure modes with terrible performance
 - Gradient-based NAS hasn't reached the hands-off AutoML stage yet

Maturity of the Fields of NAS and HPO

- Hyperparameter optimization is a mature field
 - Blackbox HPO has been researched for decades; there are many software packages
 - Multi-fidelity HPO has also become quite mature
- Neural architecture search is still a very young field
 - Blackbox is quite mature, but slow
 - Multi-fidelity NAS is also quite mature and faster
 - Meta-learning + multi-fidelity NAS is fast, but is still a very young field
 - Gradient-based NAS is fast, but can have failure modes with terrible performance
 - Gradient-based NAS hasn't reached the hands-off AutoML stage yet
- NAS is mainly used to [create new architectures that many others can reuse](#)

Maturity of the Fields of NAS and HPO

- Hyperparameter optimization is a mature field
 - Blackbox HPO has been researched for decades; there are many software packages
 - Multi-fidelity HPO has also become quite mature
- Neural architecture search is still a very young field
 - Blackbox is quite mature, but slow
 - Multi-fidelity NAS is also quite mature and faster
 - Meta-learning + multi-fidelity NAS is fast, but is still a very young field
 - Gradient-based NAS is fast, but can have failure modes with terrible performance
 - Gradient-based NAS hasn't reached the hands-off AutoML stage yet
- NAS is mainly used to create new architectures that many others can reuse
- Given a new dataset, HPO is crucial for good performance; NAS may not be necessary
 - The biggest gains typically come from tuning key hyperparameters (learning rate, etc)
 - Reusing a previous architecture often yields competitive results

Practical Recommendations for NAS and HPO

- Recommendations for a new dataset
 - Always run HPO
 - Try NAS if you can

Practical Recommendations for NAS and HPO

- Recommendations for a new dataset
 - Always run HPO
 - Try NAS if you can
- How to combine NAS & HPO
 - If the compute budget suffices, **optimize them jointly**, e.g., using BOHB
 - + Auto-PyTorch Tabular [Zimmer, Lindauer & Hutter, 2020]
 - + Auto-RL [Runge et al. 2019]

Practical Recommendations for NAS and HPO

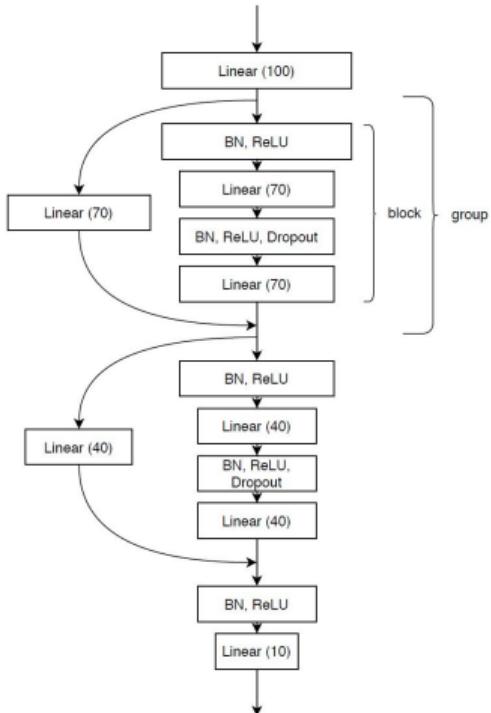
- Recommendations for a new dataset
 - Always run HPO
 - Try NAS if you can
- How to combine NAS & HPO
 - If the compute budget suffices, **optimize them jointly**, e.g., using BOHB
 - + Auto-PyTorch Tabular [Zimmer, Lindauer & Hutter, 2020]
 - + Auto-RL [Runge et al. 2019]
 - Else
 - + If you have decent hyperparameters:
run NAS, followed by HPO for fine-tuning [Saikat et al. 2019]

Practical Recommendations for NAS and HPO

- Recommendations for a new dataset
 - Always run HPO
 - Try NAS if you can
- How to combine NAS & HPO
 - If the compute budget suffices, **optimize them jointly**, e.g., using BOHB
 - + Auto-PyTorch Tabular [Zimmer, Lindauer & Hutter, 2020]
 - + Auto-RL [Runge et al. 2019]
 - Else
 - + If you have decent hyperparameters:
run NAS, followed by HPO for fine-tuning [Saikat et al. 2019]
 - + If you don't have decent hyperparameters: **first run HPO** to get competitive

Case Study: NAS & HPO in Auto-PyTorch Tabular [Zimmer, Lindauer & Hutter, 2020]

- Joint Architecture Search and Hyperparameter Optimization

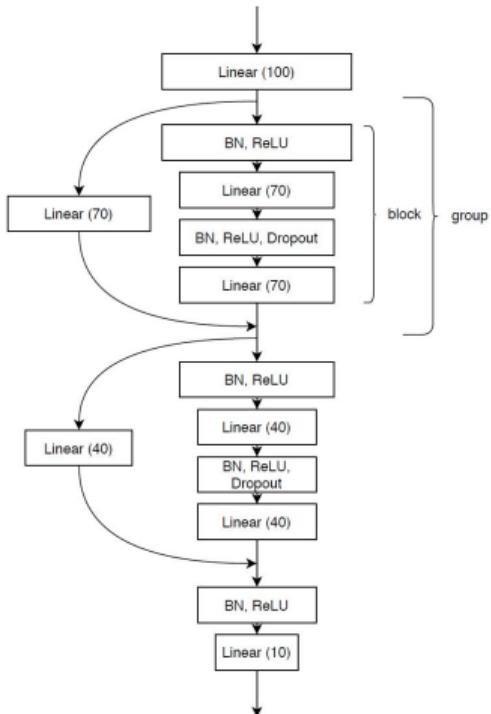


	Name	Range	log	type	cond.
Architecture	network type	[ResNet, MLPNet]	-	cat	-
	num layers (MLP)	[1, 6]	-	int	✓
	max units (MLP)	[64, 1024]	✓	int	✓
	max dropout (MLP)	[0, 1]	-	float	✓
	num groups (Res)	[1, 5]	-	int	✓
	blocks per group (Res)	[1, 3]	-	int	✓
	max units (Res)	[32, 512]	✓	int	✓
	use dropout (Res)	[F, T]	-	bool	✓
	use shake drop	[F, T]	-	bool	✓
	use shake shake	[F, T]	-	bool	✓
Hyper-parameters	max dropout (Res)	[0, 1]	-	float	✓
	max shake drop (Res)	[0, 1]	-	float	✓
	batch size	[16, 512]	✓	int	-
	optimizer	[SGD, Adam]	-	cat	-
	learning rate (SGD)	[1e-4, 1e-1]	✓	float	✓
	L2 reg. (SGD)	[1e-5, 1e-1]	-	float	✓
	momentum	[0.1, 0.999]	-	float	✓
	learning rate (Adam)	[1e-4, 1e-1]	✓	float	✓
	L2 reg. (Adam)	[1e-5, 1e-1]	-	float	✓
	training technique	[standard, mixup]	-	cat	-
Data	mixup alpha	[0, 1]	-	float	✓
	preprocessor	[none, trunc. SVD]	-	cat	-
	SVD target dim	[10, 256]	-	int	✓
	label noise	[0, 1]	-	float	✓

Case Study: NAS & HPO in Auto-PyTorch Tabular [Zimmer, Lindauer & Hutter, 2020]

Joint Architecture Search and Hyperparameter Optimization

- Purely using HPO techniques: very similar methods as in Auto-sklearn 2.0

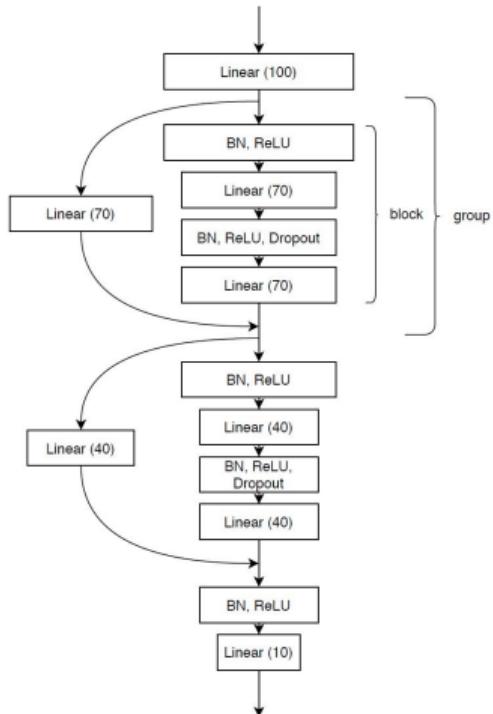


	Name	Range	log	type	cond.
Architecture	network type	[ResNet, MLPNet]	-	cat	-
	num layers (MLP)	[1, 6]	-	int	✓
	max units (MLP)	[64, 1024]	✓	int	✓
	max dropout (MLP)	[0, 1]	-	float	✓
	num groups (Res)	[1, 5]	-	int	✓
	blocks per group (Res)	[1, 3]	-	int	✓
	max units (Res)	[32, 512]	✓	int	✓
	use dropout (Res)	[F, T]	-	bool	✓
	use shake drop	[F, T]	-	bool	✓
	use shake shake	[F, T]	-	bool	✓
Hyperparameters	max dropout (Res)	[0, 1]	-	float	✓
	max shake drop (Res)	[0, 1]	-	float	✓
	batch size	[16, 512]	✓	int	-
	optimizer	[SGD, Adam]	-	cat	-
	learning rate (SGD)	[1e-4, 1e-1]	✓	float	✓
	L2 reg. (SGD)	[1e-5, 1e-1]	-	float	✓
	momentum	[0.1, 0.999]	-	float	✓
	learning rate (Adam)	[1e-4, 1e-1]	✓	float	✓
	L2 reg. (Adam)	[1e-5, 1e-1]	-	float	✓
	training technique	[standard, mixup]	-	cat	-
	mixup alpha	[0, 1]	-	float	✓
	preprocessor	[none, trunc. SVD]	-	cat	-
	SVD target dim	[10, 256]	-	int	✓

Case Study: NAS & HPO in Auto-PyTorch Tabular [Zimmer, Lindauer & Hutter, 2020]

Joint Architecture Search and Hyperparameter Optimization

- Purely using HPO techniques: very similar methods as in [Auto-sklearn 2.0](#)
- Multi-fidelity optimization with BOHB
- Meta-learning with task-independent recommendations

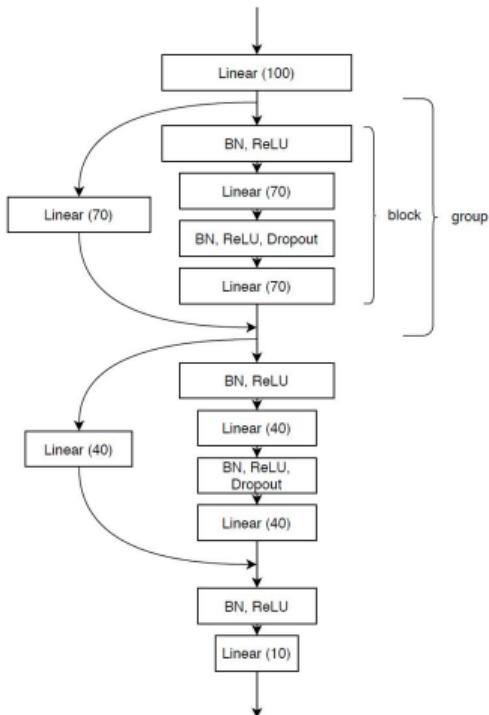


	Name	Range	log	type	cond.
Architecture	network type	[ResNet, MLPNet]	-	cat	-
	num layers (MLP)	[1, 6]	-	int	✓
	max units (MLP)	[64, 1024]	✓	int	✓
	max dropout (MLP)	[0, 1]	-	float	✓
	num groups (Res)	[1, 5]	-	int	✓
	blocks per group (Res)	[1, 3]	-	int	✓
	max units (Res)	[32, 512]	✓	int	✓
	use dropout (Res)	[F, T]	-	bool	✓
	use shake drop	[F, T]	-	bool	✓
	use shake shake	[F, T]	-	bool	✓
Hyperparameters	max dropout (Res)	[0, 1]	-	float	✓
	max shake drop (Res)	[0, 1]	-	float	✓
	batch size	[16, 512]	✓	int	-
	optimizer	[SGD, Adam]	-	cat	-
	learning rate (SGD)	[1e-4, 1e-1]	✓	float	✓
	L2 reg. (SGD)	[1e-5, 1e-1]	-	float	✓
	momentum	[0.1, 0.999]	-	float	✓
	learning rate (Adam)	[1e-4, 1e-1]	✓	float	✓
	L2 reg. (Adam)	[1e-5, 1e-1]	-	float	✓
	training technique	[standard, mixup]	-	cat	-
	mixup alpha	[0, 1]	-	float	✓
	preprocessor	[none, trunc. SVD]	-	cat	-
	SVD target dim	[10, 256]	-	int	✓

Case Study: NAS & HPO in Auto-PyTorch Tabular [Zimmer, Lindauer & Hutter, 2020]

- Joint Architecture Search and Hyperparameter Optimization

- Purely using HPO techniques: very similar methods as in [Auto-sklearn 2.0](#)
- Multi-fidelity optimization with BOHB
- Meta-learning with task-independent recommendations
- Ensembling of neural nets and traditional ML

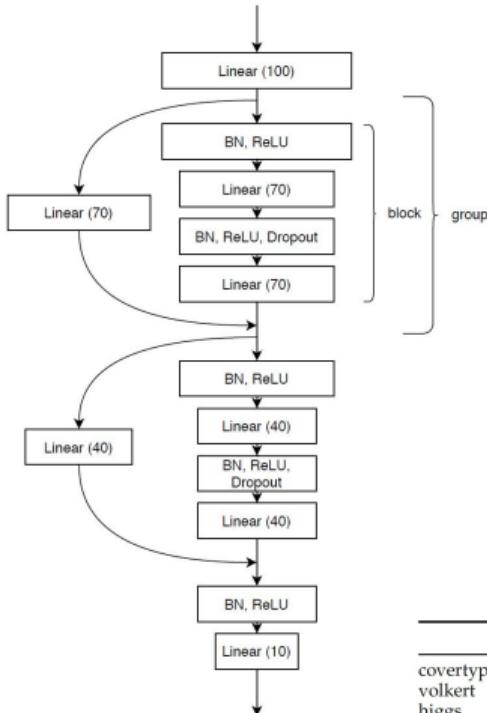


	Name	Range	log	type	cond.
Architecture	network type	[ResNet, MLPNet]	-	cat	-
	num layers (MLP)	[1, 6]	-	int	✓
	max units (MLP)	[64, 1024]	✓	int	✓
	max dropout (MLP)	[0, 1]	-	float	✓
	num groups (Res)	[1, 5]	-	int	✓
	blocks per group (Res)	[1, 3]	-	int	✓
	max units (Res)	[32, 512]	✓	int	✓
	use dropout (Res)	[F, T]	-	bool	✓
	use shake drop	[F, T]	-	bool	✓
	use shake shake	[F, T]	-	bool	✓
Hyperparameters	max dropout (Res)	[0, 1]	-	float	✓
	max shake drop (Res)	[0, 1]	-	float	✓
	batch size	[16, 512]	✓	int	-
	optimizer	[SGD, Adam]	-	cat	-
	learning rate (SGD)	[1e-4, 1e-1]	✓	float	✓
	L2 reg. (SGD)	[1e-5, 1e-1]	-	float	✓
	momentum	[0.1, 0.999]	-	float	✓
	learning rate (Adam)	[1e-4, 1e-1]	✓	float	✓
	L2 reg. (Adam)	[1e-5, 1e-1]	-	float	✓
	training technique	[standard, mixup]	-	cat	-
	mixup alpha	[0, 1]	-	float	✓
	preprocessor	[none, trunc. SVD]	-	cat	-
	SVD target dim	[10, 256]	-	int	✓

Case Study: NAS & HPO in Auto-PyTorch Tabular [Zimmer, Lindauer & Hutter, 2020]

- Joint Architecture Search and Hyperparameter Optimization

- Purely using HPO techniques: very similar methods as in Auto-sklearn 2.0
- Multi-fidelity optimization with BOHB
- Meta-learning with task-independent recommendations
- Ensembling of neural nets and traditional ML



	Name	Range	log	type	cond.
Architecture	network type	[ResNet, MLPNet]	-	cat	-
	num layers (MLP)	[1, 6]	-	int	✓
	max units (MLP)	[64, 1024]	✓	int	✓
	max dropout (MLP)	[0, 1]	-	float	✓
	num groups (Res)	[1, 5]	-	int	✓
	blocks per group (Res)	[1, 3]	-	int	✓
	max units (Res)	[32, 512]	✓	int	✓
	use dropout (Res)	[F, T]	-	bool	✓
	use shake drop	[F, T]	-	bool	✓
	use shake shake	[F, T]	-	bool	✓
Hyperparameters	max dropout (Res)	[0, 1]	-	float	✓
	max shake drop (Res)	[0, 1]	-	float	✓
	batch size	[16, 512]	✓	int	-
	optimizer	[SGD, Adam]	-	cat	-
	learning rate (SGD)	[1e-4, 1e-1]	✓	float	✓
Performance	L2 reg. (SGD)	[1e-5, 1e-1]	-	float	✓
	momentum	[0.1, 0.999]	-	float	✓
	learning rate (Adam)	[1e-4, 1e-1]	✓	float	✓
	L2 reg. (Adam)	[1e-5, 1e-1]	-	float	✓
	training technique	[standard, mixup]	-	cat	-
	mixup alpha	[0, 1]	-	float	✓
	preprocessor	[none, trunc. SVD]	-	cat	-
	SVD target dim	[10, 256]	-	int	✓

	Auto-PyTorch	AutoGluon	AutoKeras	Auto-Sklearn	hyperopt-sklearn
covertype	96.86 ± 0.41	-	61.61 ± 3.52	-	-
volkert	79.46 ± 0.43	68.34 ± 0.10	44.25 ± 2.38	67.32 ± 0.46	-
higgs	73.01 ± 0.09	72.6 ± 0.00	71.25 ± 0.29	72.03 ± 0.33	-
car	99.22 ± 0.02	97.19 ± 0.35	93.39 ± 2.82	98.42 ± 0.62	98.95 ± 0.96
mfeat-factors	99.10 ± 0.18	98.03 ± 0.23	97.73 ± 0.23	98.64 ± 0.39	97.88 ± 38.48
apsfailure	99.32 ± 0.01	99.5 ± 0.03	-	99.43 ± 0.04	-
phoneme	90.59 ± 0.13	89.62 ± 0.06	86.76 ± 0.12	89.26 ± 0.14	89.79 ± 4.54
dibert	99.04 ± 0.15	98.17 ± 0.05	96.51 ± 0.62	98.14 ± 0.47	-

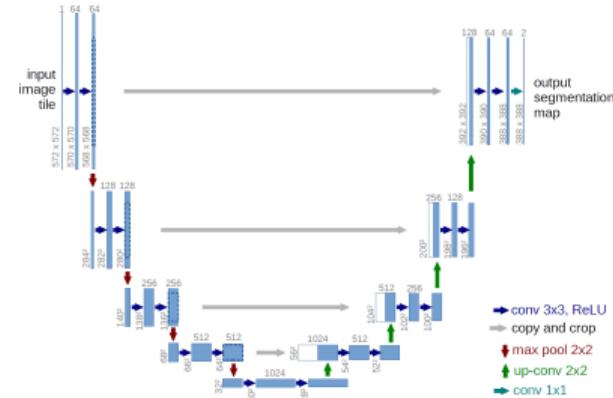
Case Study: NAS & HPO in Auto-DispNet

- Problem: disparity estimation
 - Estimate depth from stereo images



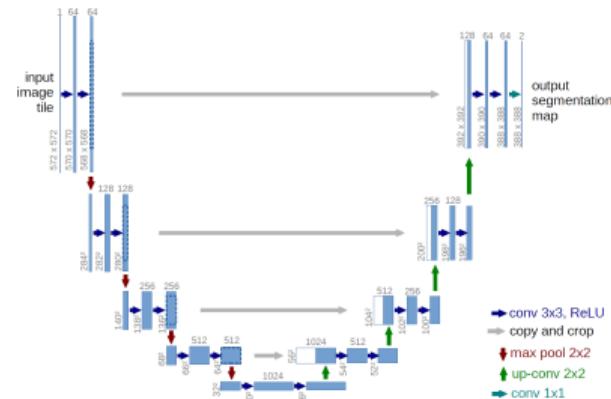
Case Study: NAS & HPO in Auto-DispNet

- Problem: disparity estimation
 - Estimate depth from stereo images
- Background: U-Net
 - ▶ Skip connections from similar spatial resolution to avoid loosing information



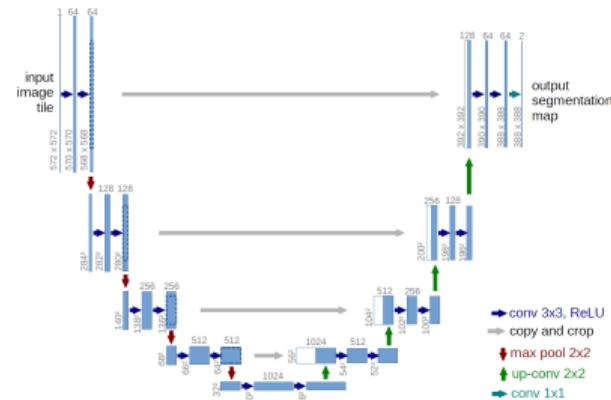
Case Study: NAS & HPO in Auto-DispNet

- Problem: disparity estimation
 - Estimate depth from stereo images
- Background: U-Net
 - ▶ Skip connections from similar spatial resolution to avoid loosing information
- Search space for DARTS
 - ▶ 3 cells: keeping spatial resolution, downsampling, and new upsampling cell that supports U-Net like skip connections



Case Study: NAS & HPO in Auto-DispNet

- Problem: disparity estimation
 - Estimate depth from stereo images
- Background: U-Net
 - ▶ Skip connections from similar spatial resolution to avoid loosing information
- Search space for DARTS
 - ▶ 3 cells: keeping spatial resolution, downsampling, and new upsampling cell that supports U-Net like skip connections
- Both NAS and HPO improved the state of the art
[Saikat et al. 2019]:
 - ▶ End-point-error (EPE) on Sintel dataset: $2.36 \rightarrow 2.14$ (by DARTS)
 - ▶ Subsequent HPO: $2.14 \rightarrow 1.94$ (by BOHB)



Questions to Answer for Yourself / Discuss with Friends

- Repetition:
If you want to use both HPO and NAS for your problem, how could you proceed?
- Discussion:
Think of a problem of your particular interest. For that problem, which approach would you use to combine HPO and NAS, and why?

AutoML: Neural Architecture Search (NAS)

Further Reading

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Further Reading

Survey on NAS: [Elsken et al. 2019]

AutoML: Dynamic Configuration & Learning Overview

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Black vs. Grey vs. White Box

- Often we treat AutoML as a **black-box problem**
 - ▶ Black box: We choose input to the black box and observe outcome

Black vs. Grey vs. White Box

- Often we treat AutoML as a **black-box problem**
 - ▶ Black box: We choose input to the black box and observe outcome
 - ▶ E.g., classical hyperparameter optimizer:
Input: Hyperparameter configuration → Output: Accuracy

Black vs. Grey vs. White Box

- Often we treat AutoML as a **black-box problem**
 - ▶ Black box: We choose input to the black box and observe outcome
 - ▶ E.g., classical hyperparameter optimizer:
Input: Hyperparameter configuration → Output: Accuracy
- We discussed how to extend AutoML to a more **grey-box approach**:
 - ▶ Grey Box: We still choose the input, but we can observe more than the outcome, e.g, intermediate results

Black vs. Grey vs. White Box

- Often we treat AutoML as a **black-box problem**
 - ▶ Black box: We choose input to the black box and observe outcome
 - ▶ E.g., classical hyperparameter optimizer:
Input: Hyperparameter configuration → Output: Accuracy
- We discussed how to extend AutoML to a more **grey-box approach**:
 - ▶ Grey Box: We still choose the input, but we can observe more than the outcome, e.g., intermediate results
 - ▶ We might control the “box” a bit, e.g., early termination

Black vs. Grey vs. White Box

- Often we treat AutoML as a **black-box problem**
 - ▶ Black box: We choose input to the black box and observe outcome
 - ▶ E.g., classical hyperparameter optimizer:
Input: Hyperparameter configuration → Output: Accuracy
- We discussed how to extend AutoML to a more **grey-box approach**:
 - ▶ Grey Box: We still choose the input, but we can observe more than the outcome, e.g., intermediate results
 - ▶ We might control the “box” a bit, e.g., early termination
 - ▶ E.g., learning curve predictions, multi-fidelity optimization, ...

Black vs. Grey vs. White Box

- Often we treat AutoML as a **black-box problem**
 - ▶ Black box: We choose input to the black box and observe outcome
 - ▶ E.g., classical hyperparameter optimizer:
Input: Hyperparameter configuration → Output: Accuracy
- We discussed how to extend AutoML to a more **grey-box approach**:
 - ▶ Grey Box: We still choose the input, but we can observe more than the outcome, e.g., intermediate results
 - ▶ We might control the “box” a bit, e.g., early termination
 - ▶ E.g., learning curve predictions, multi-fidelity optimization, ...
 - ~~ often more efficient than black-box approaches (if done right)

Black vs. Grey vs. White Box

- Often we treat AutoML as a **black-box problem**
 - ▶ Black box: We choose input to the black box and observe outcome
 - ▶ E.g., classical hyperparameter optimizer:
Input: Hyperparameter configuration → Output: Accuracy
- We discussed how to extend AutoML to a more **grey-box approach**:
 - ▶ Grey Box: We still choose the input, but we can observe more than the outcome, e.g., intermediate results
 - ▶ We might control the “box” a bit, e.g., early termination
 - ▶ E.g., learning curve predictions, multi-fidelity optimization, ...
 - ~~> often more efficient than black-box approaches (if done right)
- Ultimately, we would like to treat AutoML as a **white-box problem**
 - ▶ White-box: We can observe and control all details of an algorithm run

Black vs. Grey vs. White Box

- Often we treat AutoML as a **black-box problem**
 - ▶ Black box: We choose input to the black box and observe outcome
 - ▶ E.g., classical hyperparameter optimizer:
Input: Hyperparameter configuration → Output: Accuracy
- We discussed how to extend AutoML to a more **grey-box approach**:
 - ▶ Grey Box: We still choose the input, but we can observe more than the outcome, e.g., intermediate results
 - ▶ We might control the “box” a bit, e.g., early termination
 - ▶ E.g., learning curve predictions, multi-fidelity optimization, ...
 - ~~> often more efficient than black-box approaches (if done right)
- Ultimately, we would like to treat AutoML as a **white-box problem**
 - ▶ White-box: We can observe and control all details of an algorithm run
 - ~~> Goal: Replace algorithm components by learned policies

Iterative Optimization Heuristics

IOHs

Iterative Optimization Heuristics (IOHs) propose a set of solution candidates in each iteration based on previous evaluations.

Iterative Optimization Heuristics

IOHs

Iterative Optimization Heuristics (IOHs) propose a set of solution candidates in each iteration based on previous evaluations.

Important Observations:

- Many ML algorithms are **iterative** in nature, in particular for big data, e.g.:
 - ▶ SGD (for linear models or for deep neural networks)
 - ▶ Tree-based algorithms

Iterative Optimization Heuristics

IOHs

Iterative Optimization Heuristics (IOHs) propose a set of solution candidates in each iteration based on previous evaluations.

Important Observations:

- Many ML algorithms are **iterative** in nature, in particular for big data, e.g.:
 - ▶ SGD (for linear models or for deep neural networks)
 - ▶ Tree-based algorithms
- Often we have only a **single solution candidate** (e.g., weights of neural network)
 - ▶ If we use an evolutionary strategy as in neural evolution, we have a population of solution candidates

Iterative Optimization Heuristics

IOHs

Iterative Optimization Heuristics (IOHs) propose a set of solution candidates in each iteration based on previous evaluations.

Important Observations:

- Many ML algorithms are **iterative** in nature, in particular for big data, e.g.:
 - ▶ SGD (for linear models or for deep neural networks)
 - ▶ Tree-based algorithms
- Often we have only a **single solution candidate** (e.g., weights of neural network)
 - ▶ If we use an evolutionary strategy as in neural evolution, we have a population of solution candidates
- Hopefully, the **quality** of solution candidates **improves** in each iteration
 - ▶ Update of the weights of a neural network

Iterative Optimization Heuristics

IOHs

Iterative Optimization Heuristics (IOHs) propose a set of solution candidates in each iteration based on previous evaluations.

Important Observations:

- Many ML algorithms are **iterative** in nature, in particular for big data, e.g.:
 - ▶ SGD (for linear models or for deep neural networks)
 - ▶ Tree-based algorithms
- Often we have only a **single solution candidate** (e.g., weights of neural network)
 - ▶ If we use an evolutionary strategy as in neural evolution, we have a population of solution candidates
- Hopefully, the **quality** of solution candidates **improves** in each iteration
 - ▶ Update of the weights of a neural network
- Main component is the **heuristic for proposal mechanism** of new solution candidates

Dynamic Adaptation of Hyperparameters

The goal is to dynamically adapt hyperparameters based on some feedback from the algorithm.

Learning for IOHs

Dynamic Adaptation of Hyperparameters

The goal is to dynamically adapt hyperparameters based on some feedback from the algorithm.

Dynamic Algorithm Configuration: DAC

The goal of DAC is to learn a policy from data
that adapts the **hyperparameter settings** of an IOH.

Learning for IOHs

Dynamic Adaptation of Hyperparameters

The goal is to dynamically adapt hyperparameters based on some feedback from the algorithm.

Dynamic Algorithm Configuration: DAC

The goal of DAC is to learn a policy from data that adapts the [hyperparameter settings](#) of an IOH.

Learning to Learn: L2L

The goal of L2L is to learn a [proposal mechanism](#) from data.

AutoML: Dynamic Configuration & Learning

Dynamic Configuration

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Iterative Optimization Heuristics

- Many iterative heuristics in algorithms are dynamic and adaptive
 - ➊ the algorithm's behavior changes over time
 - ➋ the algorithm's behavior changes based on internal statistics

Iterative Optimization Heuristics

- Many iterative heuristics in algorithms are dynamic and adaptive
 - ➊ the algorithm's behavior changes over time
 - ➋ the algorithm's behavior changes based on internal statistics
- These heuristics might control other hyperparameters of the algorithms

Iterative Optimization Heuristics

- Many iterative heuristics in algorithms are dynamic and adaptive
 - ① the algorithm's behavior changes over time
 - ② the algorithm's behavior changes based on internal statistics
- These heuristics might control other hyperparameters of the algorithms
- Example: learning rate schedules for training DNNs
 - ① exponential decaying learning rate: based on number of iterations, learning rate decreases

Iterative Optimization Heuristics

- Many iterative heuristics in algorithms are dynamic and adaptive
 - ➊ the algorithm's behavior changes over time
 - ➋ the algorithm's behavior changes based on internal statistics
- These heuristics might control other hyperparameters of the algorithms
- Example: learning rate schedules for training DNNs
 - ➊ exponential decaying learning rate: based on number of iterations, learning rate decreases
 - ➋ Reduce learning rate on plateaus: if the learning stagnates for some time, the learning rate is decreased by a factor

Iterative Optimization Heuristics

- Many iterative heuristics in algorithms are dynamic and adaptive
 - ➊ the algorithm's behavior changes over time
 - ➋ the algorithm's behavior changes based on internal statistics
- These heuristics might control other hyperparameters of the algorithms
- Example: learning rate schedules for training DNNs
 - ➊ exponential decaying learning rate: based on number of iterations, learning rate decreases
 - ➋ Reduce learning rate on plateaus: if the learning stagnates for some time, the learning rate is decreased by a factor
- other examples: restart probability of search, mutation rate of evolutionary algorithms, ...

Parametrization of Learning Rate Schedules

- How can we parameterize learning rate schedules?

- ① exponential decaying learning rate:

- ★ initial learning rate
 - ★ minimal learning rate
 - ★ multiplicative factor

Parametrization of Learning Rate Schedules

- How can we parameterize learning rate schedules?

- ① exponential decaying learning rate:

- ★ initial learning rate
 - ★ minimal learning rate
 - ★ multiplicative factor

- ② Reduce learning rate on plateaus:

- ★ patience (in number of epochs)
 - ★ patience threshold
 - ★ decreasing factor
 - ★ cool-down break (in number of epochs)

Parametrization of Learning Rate Schedules

- How can we parameterize learning rate schedules?

- ① exponential decaying learning rate:

- ★ initial learning rate
 - ★ minimal learning rate
 - ★ multiplicative factor

- ② Reduce learning rate on plateaus:

- ★ patience (in number of epochs)
 - ★ patience threshold
 - ★ decreasing factor
 - ★ cool-down break (in number of epochs)

~~~ Many hyperparameters only to control a single hyperparameter

# Parametrization of Learning Rate Schedules

- How can we parameterize learning rate schedules?
  - ① exponential decaying learning rate:
    - ★ initial learning rate
    - ★ minimal learning rate
    - ★ multiplicative factor
  - ② Reduce learning rate on plateaus:
    - ★ patience (in number of epochs)
    - ★ patience threshold
    - ★ decreasing factor
    - ★ cool-down break (in number of epochs)
- ~~ Many hyperparameters only to control a single hyperparameter
- Still not guaranteed that optimal setting of e.g. learning rate schedules will lead to optimal learning behavior
  - ▶ Learning rate schedules are only heuristics

# Dynamic Algorithm Configuration

- So far, we assumed that an algorithm runs with static settings
- However, settings, such as learning rate, have to be adapted over time

## Definition

Let

- $\lambda \in \Lambda$  be a hyperparameter configuration of an algorithm  $\mathcal{A}$ ,

# Dynamic Algorithm Configuration

- So far, we assumed that an algorithm runs with static settings
- However, settings, such as learning rate, have to be adapted over time

## Definition

Let

- $\lambda \in \Lambda$  be a hyperparameter configuration of an algorithm  $\mathcal{A}$ ,
- $p(\mathcal{D})$  be a probability distribution over meta datasets  $\mathcal{D} \in \mathbf{D}$ ,

# Dynamic Algorithm Configuration

- So far, we assumed that an algorithm runs with static settings
- However, settings, such as learning rate, have to be adapted over time

## Definition

Let

- $\lambda \in \Lambda$  be a hyperparameter configuration of an algorithm  $\mathcal{A}$ ,
- $p(\mathcal{D})$  be a probability distribution over meta datasets  $\mathcal{D} \in \mathbf{D}$ ,
- $s^{(t)}$  be a state description of  $\mathcal{A}$  solving  $\mathcal{D}$  at time point  $t$ ,

# Dynamic Algorithm Configuration

- So far, we assumed that an algorithm runs with static settings
- However, settings, such as learning rate, have to be adapted over time

## Definition

Let

- $\lambda \in \Lambda$  be a hyperparameter configuration of an algorithm  $\mathcal{A}$ ,
- $p(\mathcal{D})$  be a probability distribution over meta datasets  $\mathcal{D} \in \mathbf{D}$ ,
- $s^{(t)}$  be a state description of  $\mathcal{A}$  solving  $\mathcal{D}$  at time point  $t$ ,
- $c : \Pi \times \mathbf{D} \rightarrow \mathbb{R}$  be a cost metric assessing the cost of a control policy  $\pi \in \Pi$  on  $\mathcal{D} \in \mathbf{D}$

# Dynamic Algorithm Configuration

- So far, we assumed that an algorithm runs with static settings
- However, settings, such as learning rate, have to be adapted over time

## Definition

Let

- $\lambda \in \Lambda$  be a hyperparameter configuration of an algorithm  $\mathcal{A}$ ,
- $p(\mathcal{D})$  be a probability distribution over meta datasets  $\mathcal{D} \in \mathbf{D}$ ,
- $s^{(t)}$  be a state description of  $\mathcal{A}$  solving  $\mathcal{D}$  at time point  $t$ ,
- $c : \Pi \times \mathbf{D} \rightarrow \mathbb{R}$  be a cost metric assessing the cost of a control policy  $\pi \in \Pi$  on  $\mathcal{D} \in \mathbf{D}$

the *dynamic algorithm configuration problem* is to obtain a policy  $\pi^* : s_t \times \mathcal{D} \mapsto \lambda$  by optimizing its cost across a distribution of datasets:

$$\pi^* \in \arg \min_{\pi \in \Pi} \int_{\mathbf{D}} p(\mathcal{D}) c(\pi, \mathcal{D}) d\mathcal{D}$$

# Dynamic Algorithm Configuration as Contextual MDP [Biedenkapp et al. 2020]

State  $s^{(t)}$  are described by statistics gathered in the algorithm run

## Dynamic Algorithm Configuration as Contextual MDP [Biedenkapp et al. 2020]

State  $s^{(t)}$  are described by statistics gathered in the algorithm run

Action  $a^{(t)}$  change hyperparameters according to some control policy  $\pi$

# Dynamic Algorithm Configuration as Contextual MDP [Biedenkapp et al. 2020]

**State**  $s^{(t)}$  are described by statistics gathered in the algorithm run

**Action**  $a^{(t)}$  change hyperparameters according to some control policy  $\pi$

**Transition** run the algorithm from state  $s^{(t)}$  to  $s^{(t+1)}$  for a "short" moment by using the hyperparameters defined by  $a^{(t)}$

# Dynamic Algorithm Configuration as Contextual MDP [Biedenkapp et al. 2020]

**State**  $s^{(t)}$  are described by statistics gathered in the algorithm run

**Action**  $a^{(t)}$  change hyperparameters according to some control policy  $\pi$

**Transition** run the algorithm from state  $s^{(t)}$  to  $s^{(t+1)}$  for a "short" moment by using the hyperparameters defined by  $a^{(t)}$

**Reward**  $r^{(t)}$  Return your current solution quality (or an approximation)

# Dynamic Algorithm Configuration as Contextual MDP [Biedenkapp et al. 2020]

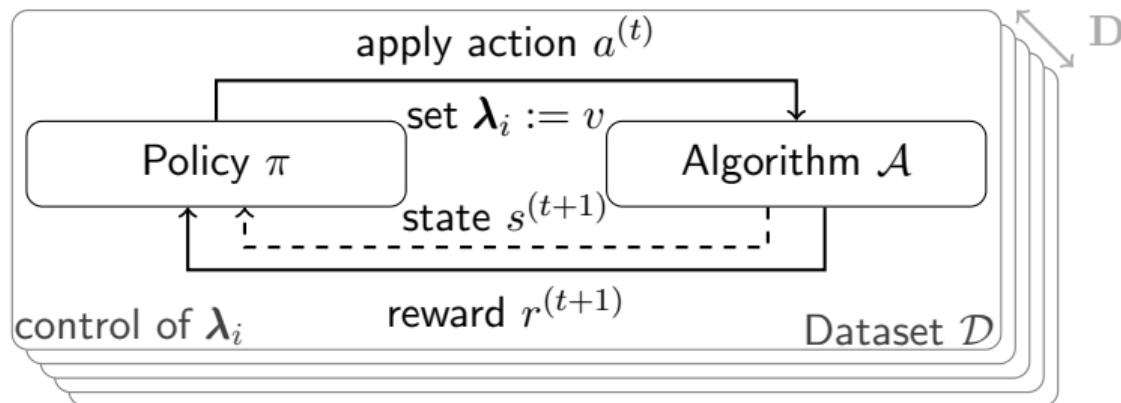
State  $s^{(t)}$  are described by statistics gathered in the algorithm run

Action  $a^{(t)}$  change hyperparameters according to some control policy  $\pi$

Transition run the algorithm from state  $s^{(t)}$  to  $s^{(t+1)}$  for a "short" moment by using the hyperparameters defined by  $a^{(t)}$

Reward  $r^{(t)}$  Return your current solution quality (or an approximation)

Context  $\mathcal{D}$  A given dataset (or task)



# Solving Dynamic Algorithm Configuration

Solve unknown MDP by using reinforcement learning (RL):

$$\mathcal{V}_{\mathcal{D}}^{\pi}(s^{(t)}) = \mathbb{E} \left[ \sum_{k=0}^{\infty} \gamma^k r_{\mathcal{D}}^{(t+k+1)} | s^{(t)} = s \right]$$

# Solving Dynamic Algorithm Configuration

Solve unknown MDP by using reinforcement learning (RL):

$$\mathcal{V}_{\mathcal{D}}^{\pi}(s^{(t)}) = \mathbb{E} \left[ \sum_{k=0}^{\infty} \gamma^k r_{\mathcal{D}}^{(t+k+1)} | s^{(t)} = s \right]$$

$$= \mathbb{E} \left[ r_{\mathcal{D}}^{(t+1)} + \gamma \mathcal{V}_{\mathcal{D}}^{\pi}(s^{(t+1)}) | s^{(t+1)} \sim \mathcal{T}_{\mathcal{D}}(s^{(t)}, \pi(s^{(t)})) \right]$$

# Solving Dynamic Algorithm Configuration

Solve unknown MDP by using reinforcement learning (RL):

$$\mathcal{V}_{\mathcal{D}}^{\pi}(s^{(t)}) = \mathbb{E} \left[ \sum_{k=0}^{\infty} \gamma^k r_{\mathcal{D}}^{(t+k+1)} | s^{(t)} = s \right]$$

$$= \mathbb{E} \left[ r_{\mathcal{D}}^{(t+1)} + \gamma \mathcal{V}_{\mathcal{D}}^{\pi}(s^{(t+1)}) | s^{(t+1)} \sim \mathcal{T}_{\mathcal{D}}(s^{(t)}, \pi(s^{(t)})) \right]$$

$$\pi^* \in \arg \max_{\pi \in \Pi} \int_{\mathbf{D}} p(\mathcal{D}) \int_{\mathcal{S}^{(0)}} \Pr(s^{(0)}) \cdot \mathcal{V}_{\mathcal{D}}^{\pi}(s^{(0)}) \, ds^{(0)} \, d\mathcal{D}$$

# Solving Dynamic Algorithm Configuration

Solve unknown MDP by using reinforcement learning (RL):

$$\mathcal{V}_{\mathcal{D}}^{\pi}(s^{(t)}) = \mathbb{E} \left[ \sum_{k=0}^{\infty} \gamma^k r_{\mathcal{D}}^{(t+k+1)} | s^{(t)} = s \right]$$

$$= \mathbb{E} \left[ r_{\mathcal{D}}^{(t+1)} + \gamma \mathcal{V}_{\mathcal{D}}^{\pi}(s^{(t+1)}) | s^{(t+1)} \sim \mathcal{T}_{\mathcal{D}}(s^{(t)}, \pi(s^{(t)})) \right]$$

$$\pi^* \in \arg \max_{\pi \in \Pi} \int_{\mathbf{D}} p(\mathcal{D}) \int_{\mathcal{S}^{(0)}} \Pr(s^{(0)}) \cdot \mathcal{V}_{\mathcal{D}}^{\pi}(s^{(0)}) \, ds^{(0)} \, d\mathcal{D}$$

↔ equivalent to Dynamic Algorithm Configuration definition

## Dynamic Algorithm Configuration across Datasets [Biedenkapp et al. 2020]

- Challenge: Evaluating all policies on all datasets is often not feasible

## Dynamic Algorithm Configuration across Datasets [Biedenkapp et al. 2020]

- Challenge: Evaluating all policies on all datasets is often not feasible
- Curriculum learning [Bengio et al. 2009] showed that we should have a curriculum of tasks we tackle

- Challenge: Evaluating all policies on all datasets is often not feasible
- Curriculum learning [Bengio et al. 2009] showed that we should have a curriculum of tasks we tackle
- Self-paced learning [Kumar et al. 2010] tries to automatically find such a curriculum
  - ▶ Focus on "easy" tasks where the agent can improve most:

# Dynamic Algorithm Configuration across Datasets [Biedenkapp et al. 2020]

- Challenge: Evaluating all policies on all datasets is often not feasible
- Curriculum learning [Bengio et al. 2009] showed that we should have a curriculum of tasks we tackle
- Self-paced learning [Kumar et al. 2010] tries to automatically find such a curriculum
  - ▶ Focus on "easy" tasks where the agent can improve most:

$$\max_{\pi, \mathbf{v}} \mathcal{C}(\pi, \mathbf{v}, K) = \sum_{i=1}^{|D|} \mathbf{v}_i \mathcal{R}_i(\pi) - \frac{1}{K} \sum_{i=1}^{|D|} \mathbf{v}_i$$

with  $\theta$  being the agent's policy parameters and  
 $\mathbf{v}$  being a masking vector for choosing the tasks at hand.

# AutoML: Dynamic Configuration & Learning

## Learning to Adjust Learning Rates

Bernd Bischl   Frank Hutter   Lars Kotthoff  
Marius Lindauer   Joaquin Vanschoren

## Learning Problem [Daniel et al. 2016]

- Optimization of a function:

$$\theta \in \arg \min F(\mathbf{X}; \theta)$$

where  $\mathbf{X}$  is an input matrix and  $f$  is parameterized by  $\theta$ .

## Learning Problem [Daniel et al. 2016]

- Optimization of a function:

$$\theta \in \arg \min F(\mathbf{X}; \theta)$$

where  $\mathbf{X}$  is an input matrix and  $f$  is parameterized by  $\theta$ .

$$F(\mathbf{X}; \theta) = \frac{1}{N} \sum_{i=1}^N f(\mathbf{x}^{(i)}; \theta)$$

## Learning Step Size Policies [Daniel et al. 2016]

- Idea: Learn the hyperparameters of the weight update (short notation)

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla F(\theta^{(t)})$$

$$\nabla F(\theta^{(t)}) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\theta^{(t)})$$

## Learning Step Size Policies [Daniel et al. 2016]

- Idea: Learn the hyperparameters of the weight update (short notation)

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla F(\theta^{(t)})$$

$$\nabla F(\theta^{(t)}) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\theta^{(t)})$$

- For SGD, this would be for example the learning rate  $\alpha$

## Learning Step Size Policies [Daniel et al. 2016]

- Idea: Learn the hyperparameters of the weight update (short notation)

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla F(\theta^{(t)})$$

$$\nabla F(\theta^{(t)}) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\theta^{(t)})$$

- For SGD, this would be for example the learning rate  $\alpha$
- Note (i):  $\alpha$  have to be adapted in the course of the training
  - similar to learning rate schedules (e.g., cosine annealing)

## Learning Step Size Policies [Daniel et al. 2016]

- Idea: Learn the hyperparameters of the weight update (short notation)

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla F(\theta^{(t)})$$

$$\nabla F(\theta^{(t)}) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\theta^{(t)})$$

- For SGD, this would be for example the learning rate  $\alpha$
- Note (i):  $\alpha$  have to be adapted in the course of the training
  - similar to learning rate schedules (e.g., cosine annealing)
- Note(ii): later we denote the learnt hyperparameters as  $\lambda$

# Learning Step Size Policies [Daniel et al. 2016]

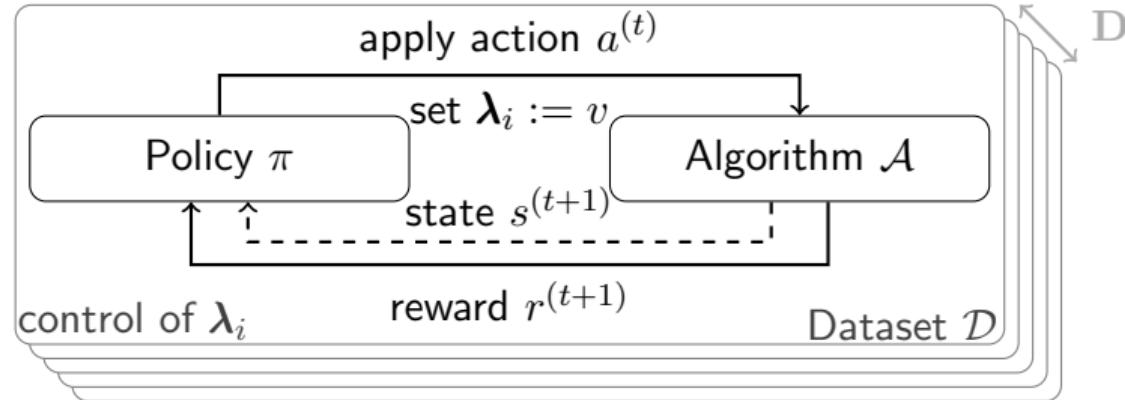
- Idea: Learn the hyperparameters of the weight update (short notation)

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla F(\theta^{(t)})$$

$$\nabla F(\theta^{(t)}) = \frac{1}{N} \sum_{i=1}^N \nabla f_i(\theta^{(t)})$$

- For SGD, this would be for example the learning rate  $\alpha$
- Note (i):  $\alpha$  have to be adapted in the course of the training
  - similar to learning rate schedules (e.g., cosine annealing)
- Note(ii): later we denote the learnt hyperparameters as  $\lambda$
- Idea: Use reinforcement learning to learn a policy  $\pi : s \mapsto a$  to control the learning rate (or other adaptive hyperparameters)

# Recap: Reinforcement Learning for Dynamic Algorithm Configuration



To apply that, we need to define:

- ① State description
- ② Action space
- ③ Reward function

**Predictive change in function value:**

$$s_1 = \log \left( \text{Var}(\Delta \tilde{f}_i) \right)$$

$$\Delta \tilde{f}_i = \tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta) - f(\mathbf{x}^{(i)}; \theta)$$

where  $\tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta)$  is done by a first order Taylor expansion

## RL for Step Size Policies: State [Daniel et al. 2016]

**Predictive change in function value:**

$$s_1 = \log \left( \text{Var}(\Delta \tilde{f}_i) \right)$$

$$\Delta \tilde{f}_i = \tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta) - f(\mathbf{x}^{(i)}; \theta)$$

where  $\tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta)$  is done by a first order Taylor expansion

**Disagreement of function values:**

$$s_2 = \log \left( \text{Var}(f(\mathbf{x}^{(i)}; \theta)) \right)$$

## RL for Step Size Policies: State [Daniel et al. 2016]

**Predictive change in function value:**

$$s_1 = \log \left( \text{Var}(\Delta \tilde{f}_i) \right)$$

$$\Delta \tilde{f}_i = \tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta) - f(\mathbf{x}^{(i)}; \theta)$$

where  $\tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta)$  is done by a first order Taylor expansion

**Disagreement of function values:**

$$s_2 = \log \left( \text{Var}(f(\mathbf{x}^{(i)}; \theta)) \right)$$

**Discounted Average** (smoothing noise from mini-batches):

$$\hat{s}_i \leftarrow \gamma \hat{s}_i + (1 - \gamma) s_i$$

## RL for Step Size Policies: State [Daniel et al. 2016]

**Predictive change in function value:**

$$s_1 = \log \left( \text{Var}(\Delta \tilde{f}_i) \right)$$

$$\Delta \tilde{f}_i = \tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta) - f(\mathbf{x}^{(i)}; \theta)$$

where  $\tilde{f}(\mathbf{x}^{(i)}; \theta + \delta\theta)$  is done by a first order Taylor expansion

**Disagreement of function values:**

$$s_2 = \log \left( \text{Var}(f(\mathbf{x}^{(i)}; \theta)) \right)$$

**Discounted Average** (smoothing noise from mini-batches):

$$\hat{s}_i \leftarrow \gamma \hat{s}_i + (1 - \gamma) s_i$$

**Uncertainty Estimate** (noise level):

$$s_{K+i} \leftarrow \gamma s_{K+i} + (1 - \gamma)(s_i - \hat{s}_i)^2$$

## RL for Step Size Policies: Learning [Daniel et al. 2016]

Reward (average loss improvement over time):

$$r = \frac{1}{T-1} \sum_{t=2}^T \left( \log(L^{(t-1)}) - \log(L^{(t)}) \right)$$

## RL for Step Size Policies: Learning [Daniel et al. 2016]

Reward (average loss improvement over time):

$$r = \frac{1}{T-1} \sum_{t=2}^T \left( \log(L^{(t-1)}) - \log(L^{(t)}) \right)$$

Optimal Policy:

$$\pi^*(\lambda | s) \in \arg \max_{\pi} \int \int p(s) \pi(\lambda | s) r(\lambda, s) ds d\lambda$$

## RL for Step Size Policies: Learning [Daniel et al. 2016]

Reward (average loss improvement over time):

$$r = \frac{1}{T-1} \sum_{t=2}^T \left( \log(L^{(t-1)}) - \log(L^{(t)}) \right)$$

Optimal Policy:

$$\pi^*(\lambda | s) \in \arg \max_{\pi} \int \int p(s) \pi(\lambda | s) r(\lambda, s) ds d\lambda$$

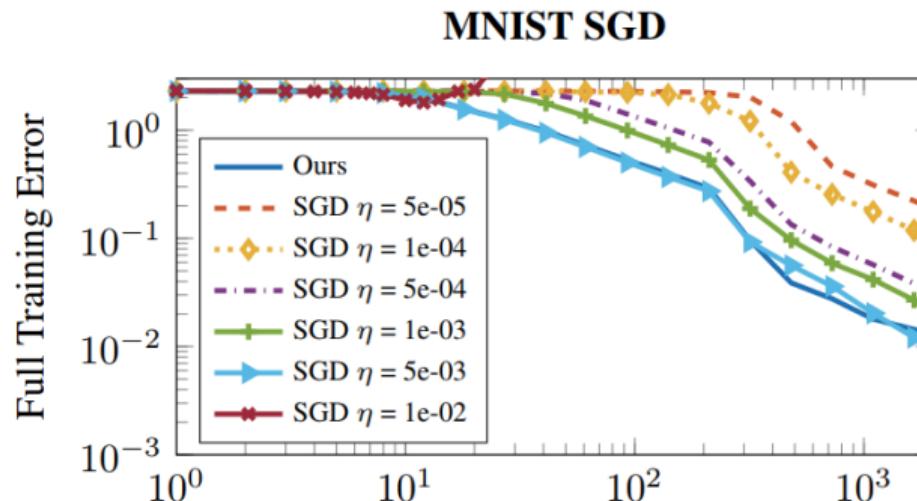
- can be learnt for example via Relative Entropy Policy Search (REPS) [Peter et al. 2010]

## RL for Step Size Policies: Training [Daniel et al. 2016]

- Goal: obtain robust policies,  
i.e., good performance for many different DNN architectures
  - ~~> Sample architectures e.g., with different numbers of filters and layers
  - ~~> (Sub-)Sample dataset
  - ~~> Sample number of optimization steps

# RL for Step Size Policies: Training [Daniel et al. 2016]

- Goal: obtain robust policies,  
i.e., good performance for many different DNN architectures
  - Sample architectures e.g., with different numbers of filters and layers
  - (Sub-)Sample dataset
  - Sample number of optimization steps



"Ours" refers to the approach by [Daniel et al. 2016] and  $\eta$  is the learning rate

# AutoML: Dynamic Configuration & Learning

## Population-based Training

Bernd Bischl   Frank Hutter   Lars Kotthoff  
Marius Lindauer   Joaquin Vanschoren

## On-the-fly Adaption

- Dynamic algorithm configuration assumes that we have access to a **representative learning environment** in an **offline learning phase**

## On-the-fly Adaption

- Dynamic algorithm configuration assumes that we have access to a **representative learning environment** in an **offline learning phase**
- What if we don't access to such an env or don't have time for offline learning?

## On-the-fly Adaption

- Dynamic algorithm configuration assumes that we have access to a **representative learning environment** in an **offline learning phase**
- What if we don't access to such an env or don't have time for offline learning?
  - ~~ Try to figure out best hyperparameter settings on the fly

# Massively parallelized Random Search

$\lambda^{(1)}$

$\lambda^{(2)}$

$\lambda^{(3)}$

$\lambda^{(4)}$

$t$

- Sample many hyperparameter configurations  $\lambda^{(i)}$  and evaluate them all in parallel

# Massively parallelized Random Search

$\lambda^{(1)}$

$\lambda^{(2)}$

$\lambda^{(3)}$

$\lambda^{(4)}$

$t$

- Sample many hyperparameter configurations  $\lambda^{(i)}$  and evaluate them all in parallel
- Pure exploration on a large population of configurations

# Massively parallelized Random Search

$\lambda^{(1)}$

$\lambda^{(2)}$

$\lambda^{(3)}$

$\lambda^{(4)}$

$t$

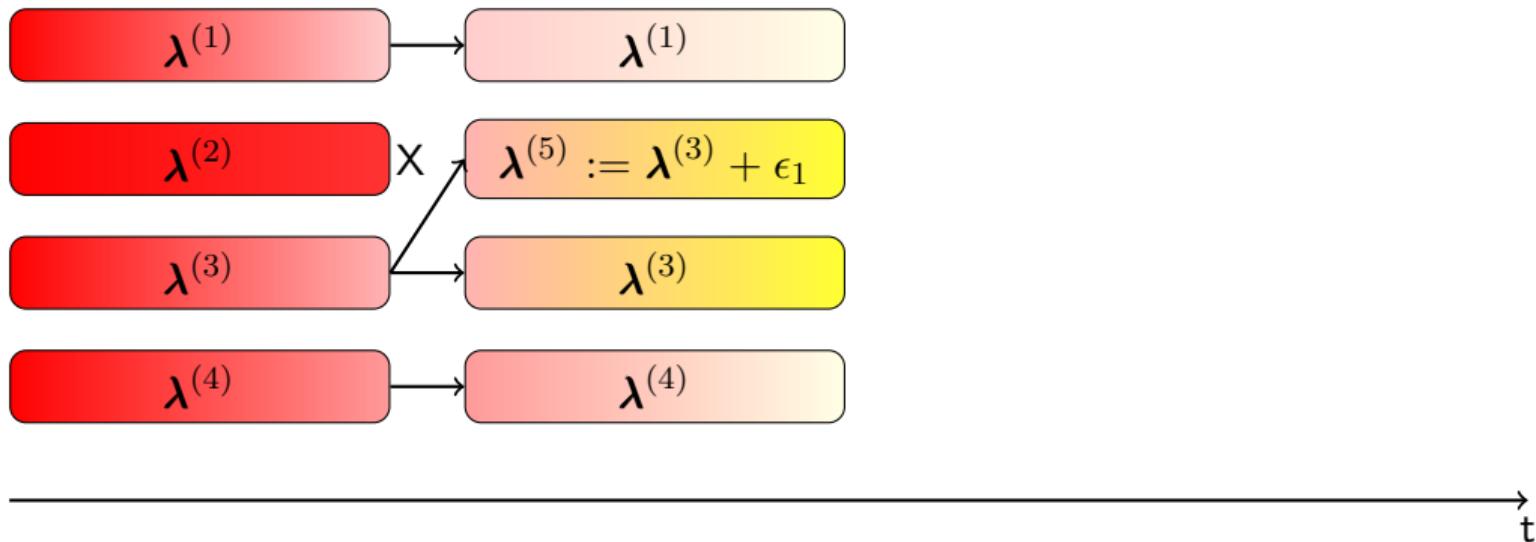
- Sample many hyperparameter configurations  $\lambda^{(i)}$  and evaluate them all in parallel
- Pure exploration on a large population of configurations
- No dynamic adaptation

# Population-based Training [Jaderberg et al. 2017]



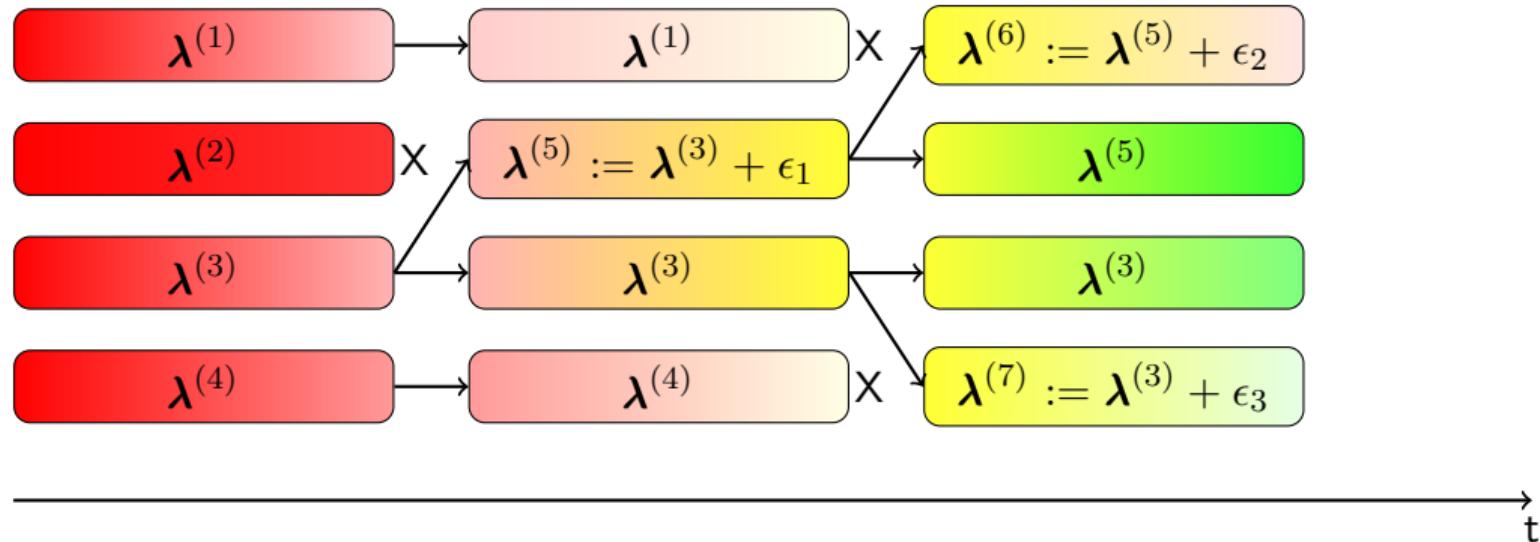
- The color indicates the performance over time

# Population-based Training [Jaderberg et al. 2017]



- The color indicates the performance over time

# Population-based Training [Jaderberg et al. 2017]



- The color indicates the performance over time

# Population-based Training [Jaderberg et al. 2017; Liang et al. 2020]

General workflow of PBT:

- ① Sample initial population
  - ▶ Each population member is a combination of hyperparameter setting  $\lambda$  and (partially trained) model

# Population-based Training [Jaderberg et al. 2017; Liang et al. 2020]

General workflow of PBT:

- ① Sample initial population
  - ▶ Each population member is a combination of hyperparameter setting  $\lambda$  and (partially trained) model
- ② Train population for a bit

# Population-based Training [Jaderberg et al. 2017; Liang et al. 2020]

General workflow of PBT:

- ① Sample initial population
  - ▶ Each population member is a combination of hyperparameter setting  $\lambda$  and (partially trained) model
- ② Train population for a bit
- ③ Tournament selection to drop poorly performing population members

# Population-based Training [Jaderberg et al. 2017; Liang et al. 2020]

General workflow of PBT:

- ① Sample initial population
  - ▶ Each population member is a combination of hyperparameter setting  $\lambda$  and (partially trained) model
- ② Train population for a bit
- ③ Tournament selection to drop poorly performing population members
- ④ Use mutation (and cross-over) to generate off-springs
  - ▶ Change the hyperparameter settings, but inherits the partially trained model (+ perturbation)

# Population-based Training [Jaderberg et al. 2017; Liang et al. 2020]

General workflow of PBT:

- ① Sample initial population
  - ▶ Each population member is a combination of hyperparameter setting  $\lambda$  and (partially trained) model
- ② Train population for a bit
- ③ Tournament selection to drop poorly performing population members
- ④ Use mutation (and cross-over) to generate off-springs
  - ▶ Change the hyperparameter settings, but inherits the partially trained model (+ perturbation)
- ~~ New population consists of so-far best performing ones and new off-springs

# Population-based Training [Jaderberg et al. 2017; Liang et al. 2020]

General workflow of PBT:

- ① Sample initial population
  - ▶ Each population member is a combination of hyperparameter setting  $\lambda$  and (partially trained) model
- ② Train population for a bit
- ③ Tournament selection to drop poorly performing population members
- ④ Use mutation (and cross-over) to generate off-springs
  - ▶ Change the hyperparameter settings, but inherits the partially trained model (+ perturbation)
- ~~ New population consists of so-far best performing ones and new off-springs
- ⑤ Go to 2.

# Properties of Population-based Training (PBT)

- PBT returns an already trained model (e.g., DNN or RL policy)

## Properties of Population-based Training (PBT)

- PBT returns an already trained model (e.g., DNN or RL policy)
- PBT uses evolutionary computing to determine well-performing hyperparameter settings

## Properties of Population-based Training (PBT)

- PBT returns an already trained model (e.g., DNN or RL policy)
- PBT uses evolutionary computing to determine well-performing hyperparameter settings
- Since hyperparameter settings changes while training the models, PBT relates to dynamic algorithm configuration

# Properties of Population-based Training (PBT)

- PBT returns an already trained model (e.g., DNN or RL policy)
- PBT uses evolutionary computing to determine well-performing hyperparameter settings
- Since hyperparameter settings changes while training the models, PBT relates to dynamic algorithm configuration
- Since each population member (i.e., model) can be trained independently, PBT can be efficiently parallelized
  - ↝ Drawback: requires substantial parallel compute resources

# Combining Population-based Training and Bayesian Optimization

- Bayesian Optimization (BO) is well known for its sample efficiency

# Combining Population-based Training and Bayesian Optimization

- Bayesian Optimization (BO) is well known for its sample efficiency
- Idea: Can we use BO to guide PBT?

# Combining Population-based Training and Bayesian Optimization

- Bayesian Optimization (BO) is well known for its sample efficiency
- Idea: Can we use BO to guide PBT?
  - ~> Less parallel compute resources are required(?)

# Combining Population-based Training and Bayesian Optimization

- Bayesian Optimization (BO) is well known for its sample efficiency
- Idea: Can we use BO to guide PBT?
  - ~~ Less parallel compute resources are required(?)
  - ~~ Scales better to higher dimensional spaces(?)

# PBT + BO: Outline

- ① Sample initial population
  - ▶ Each population member is a combination of hyperparameter setting  $\lambda$  and (partially trained) model
- ② Train population for a bit
- ③ Tournament selection to drop poorly performing population members
- ④ Use **Bayesian optimization** to select new hyperparameter settings
  - ▶ Change the hyperparameter settings, but inherits the partially trained model (+ perturbation)
- ~~ New population consists of so-far best performing ones and new off-springs
- ⑤ Go to 2.

## PBT + BO: Parallel Evaluation

- Challenge I: PBT runs in parallel asynchronously

## PBT + BO: Parallel Evaluation

- Challenge I: PBT runs in parallel asynchronously
  - ↝ BO has to take into account that other hyperparameter settings are being evaluated already

# PBT + BO: Parallel Evaluation

- Challenge I: PBT runs in parallel asynchronously
  - ↝ BO has to take into account that other hyperparameter settings are being evaluated already
- Several ideas on how to parallelize BO

# PBT + BO: Parallel Evaluation

- Challenge I: PBT runs in parallel asynchronously
  - ↝ BO has to take into account that other hyperparameter settings are being evaluated already
- Several ideas on how to parallelize BO
  - ▶ Randomize the model training or optimization of the acquisition function

# PBT + BO: Parallel Evaluation

- Challenge I: PBT runs in parallel asynchronously
  - ~~> BO has to take into account that other hyperparameter settings are being evaluated already
- Several ideas on how to parallelize BO
  - ▶ Randomize the model training or optimization of the acquisition function
  - ▶ Thompson sampling to use only a single explanation of the data  
(in proportion to its likelihood)

# PBT + BO: Parallel Evaluation

- Challenge I: PBT runs in parallel asynchronously
  - ~~> BO has to take into account that other hyperparameter settings are being evaluated already
- Several ideas on how to parallelize BO
  - ▶ Randomize the model training or optimization of the acquisition function
  - ▶ Thompson sampling to use only a single explanation of the data  
(in proportion to its likelihood)
  - ▶ Hallucinate performance of other hyperparameter settings in optimistically, pessimistically or in expectation of the current surrogate model

## PBT + BO: Parallel Evaluation [Parker-Holder et al. 2020]

- Challenge II: The cost depends on the previous  $\lambda^{(1)}, \lambda^{(2)}, \dots, \lambda^{(t-1)}$

## PBT + BO: Parallel Evaluation [Parker-Holder et al. 2020]

- Challenge II: The cost depends on the previous  $\lambda^{(1)}, \lambda^{(2)}, \dots, \lambda^{(t-1)}$
- BO-Surrogate model predicts the cost improvement over time:

$$c_{\text{PBT}}^{(t)}(\lambda) = \frac{c^{(t)}(\lambda) - c^{(t-1)}(\lambda)}{\Delta t}$$

where  $c^{(t)}(\lambda)$  is the cost for a given hyperparameter setting at time step  $t$ .

## PBT + BO: Parallel Evaluation [Parker-Holder et al. 2020]

- Challenge II: The cost depends on the previous  $\lambda^{(1)}, \lambda^{(2)}, \dots, \lambda^{(t-1)}$
- BO-Surrogate model predicts the cost improvement over time:

$$c_{\text{PBT}}^{(t)}(\lambda) = \frac{c^{(t)}(\lambda) - c^{(t-1)}(\lambda)}{\Delta t}$$

where  $c^{(t)}(\lambda)$  is the cost for a given hyperparameter setting at time step  $t$ .

- Remark: Also add  $c^{(t-1)}$  as an input to the BO-surrogate model to ease the task of predicting the improvement

# AutoML: Dynamic Configuration & Learning

## Learning to Learn: Supervised

Bernd Bischl   Frank Hutter   Lars Kotthoff  
Marius Lindauer   Joaquin Vanschoren

# Learning to Learn

## Idea

- Learn algorithms directly, e.g., how to search in the weight space
- First idea: learn weight updates of a neural network

# Learning to Learn

## Idea

- Learn algorithms directly, e.g., how to search in the weight space
- First idea: learn weight updates of a neural network

Learning to learn by gradient descent by gradient descent

[Andrychowicz et al. 2016]

Weight updates (note:  $\theta$  denote DNN weights):

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla f(\theta^{(t)})$$

# Learning to Learn

## Idea

- Learn algorithms directly, e.g., how to search in the weight space
- First idea: learn weight updates of a neural network

Learning to learn by gradient descent by gradient descent

[Andrychowicz et al. 2016]

Weight updates (note:  $\theta$  denote DNN weights):

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla f(\theta^{(t)})$$

Even more general:

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}(\nabla f(\theta^{(t)}), \phi)$$

where  $g$  is the optimizer and  $\phi$  are the parameters of the optimizer  $g$ .

# Learning to Learn

## Idea

- Learn algorithms directly, e.g., how to search in the weight space
- First idea: learn weight updates of a neural network

Learning to learn by gradient descent by gradient descent

[Andrychowicz et al. 2016]

Weight updates (note:  $\theta$  denote DNN weights):

$$\theta^{(t+1)} = \theta^{(t)} - \alpha^{(t)} \nabla f(\theta^{(t)})$$

Even more general:

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}(\nabla f(\theta^{(t)}), \phi)$$

where  $g$  is the optimizer and  $\phi$  are the parameters of the optimizer  $g$ .

~~ Goal: Optimize  $f$  wrt  $\theta$  by learning  $g$  (resp.  $\phi$ )

## Learning to Learn: Objective [Andrychowicz et al. 2016]

$$L(\phi) = \mathbb{E} [f(\theta^*(f, \phi))]$$

where  $L$  is a loss function and  $\theta^*(f, \phi)$  are the optimized weights  $\theta^*$  by using the optimizer parameterized with  $\phi$  on function  $f$ .

## Learning to Learn: Objective [Andrychowicz et al. 2016]

$$L(\phi) = \mathbb{E} [f(\theta^*(f, \phi))]$$

where  $L$  is a loss function and  $\theta^*(f, \phi)$  are the optimized weights  $\theta^*$  by using the optimizer parameterized with  $\phi$  on function  $f$ .

$$L(\phi) = \mathbb{E} \left[ \sum_{t=1}^T w^{(t)} f(\theta^{(t)}) \right]$$

## Learning to Learn: Objective [Andrychowicz et al. 2016]

$$L(\phi) = \mathbb{E} [f(\theta^*(f, \phi))]$$

where  $L$  is a loss function and  $\theta^*(f, \phi)$  are the optimized weights  $\theta^*$  by using the optimizer parameterized with  $\phi$  on function  $f$ .

$$L(\phi) = \mathbb{E} \left[ \sum_{t=1}^T w^{(t)} f(\theta^{(t)}) \right]$$

where  $w_t$  are arbitrary weights associated with each time step and

## Learning to Learn: Objective [Andrychowicz et al. 2016]

$$L(\phi) = \mathbb{E} [f(\theta^*(f, \phi))]$$

where  $L$  is a loss function and  $\theta^*(f, \phi)$  are the optimized weights  $\theta^*$  by using the optimizer parameterized with  $\phi$  on function  $f$ .

$$L(\phi) = \mathbb{E} \left[ \sum_{t=1}^T w^{(t)} f(\theta^{(t)}) \right]$$

where  $w_t$  are arbitrary weights associated with each time step and

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}$$

$$\begin{pmatrix} g^{(t)} \\ h^{(t+1)} \end{pmatrix} = m(\nabla_\theta f(\theta^{(t)}), h^{(t)}, \phi)$$

## Learning to Learn: Objective [Andrychowicz et al. 2016]

$$L(\phi) = \mathbb{E} [f(\theta^*(f, \phi))]$$

where  $L$  is a loss function and  $\theta^*(f, \phi)$  are the optimized weights  $\theta^*$  by using the optimizer parameterized with  $\phi$  on function  $f$ .

$$L(\phi) = \mathbb{E} \left[ \sum_{t=1}^T w^{(t)} f(\theta^{(t)}) \right]$$

where  $w_t$  are arbitrary weights associated with each time step and

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}$$

$$\begin{pmatrix} g^{(t)} \\ h^{(t+1)} \end{pmatrix} = m(\nabla_\theta f(\theta^{(t)}), h^{(t)}, \phi)$$

↝ Goal: Learn  $m$  via  $\phi$  by using gradient descent by optimizing  $L$

## Learning to Learn: Objective [Andrychowicz et al. 2016]

$$L(\phi) = \mathbb{E} [f(\theta^*(f, \phi))]$$

where  $L$  is a loss function and  $\theta^*(f, \phi)$  are the optimized weights  $\theta^*$  by using the optimizer parameterized with  $\phi$  on function  $f$ .

$$L(\phi) = \mathbb{E} \left[ \sum_{t=1}^T w^{(t)} f(\theta^{(t)}) \right]$$

where  $w_t$  are arbitrary weights associated with each time step and

$$\theta^{(t+1)} = \theta^{(t)} + g^{(t)}$$

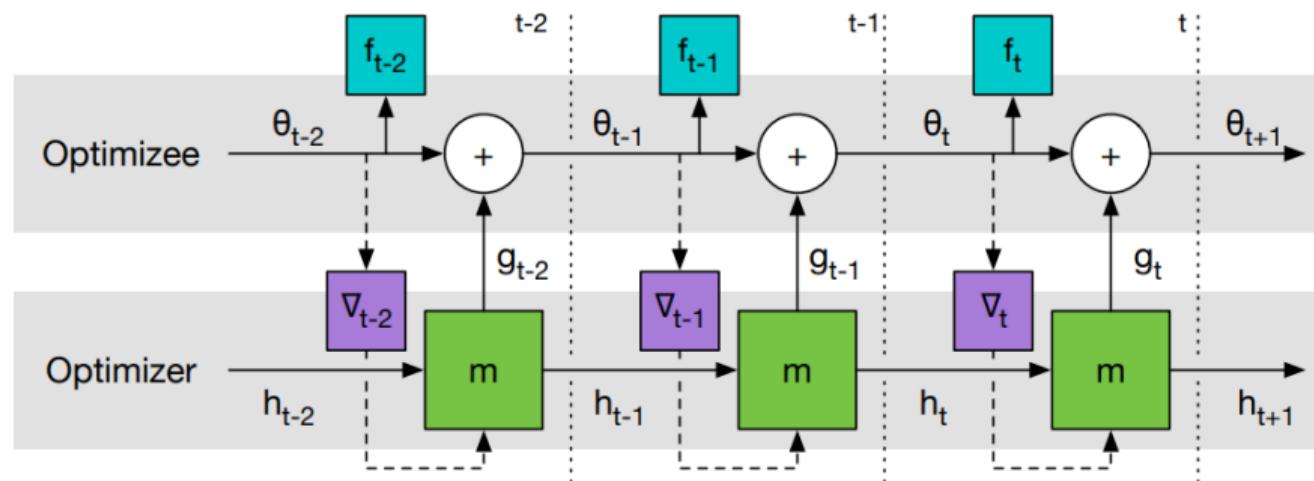
$$\begin{pmatrix} g^{(t)} \\ h^{(t+1)} \end{pmatrix} = m(\nabla_\theta f(\theta^{(t)}), h^{(t)}, \phi)$$

- ~~> Goal: Learn  $m$  via  $\phi$  by using gradient descent by optimizing  $L$
- ~~> “Learning to learn gradient descent by gradient descent”

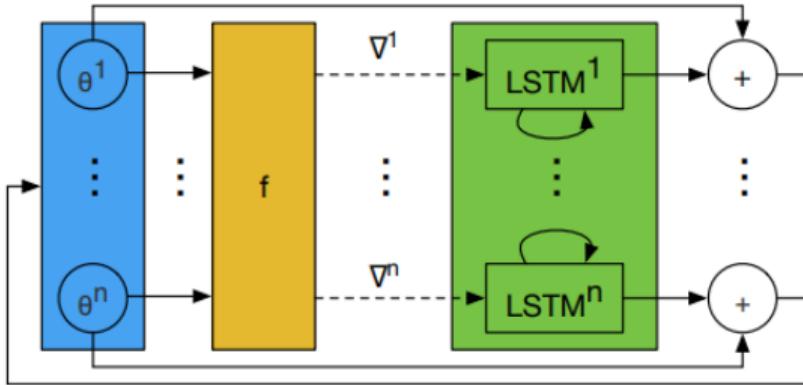
# Learning to Learn: LSTM approach [Andrychowicz et al. 2016]

**Optimizee** Target network to be trained

**Optimizer** LSTM with hidden state  $h_t$  that predicts weight updates  $g_t$

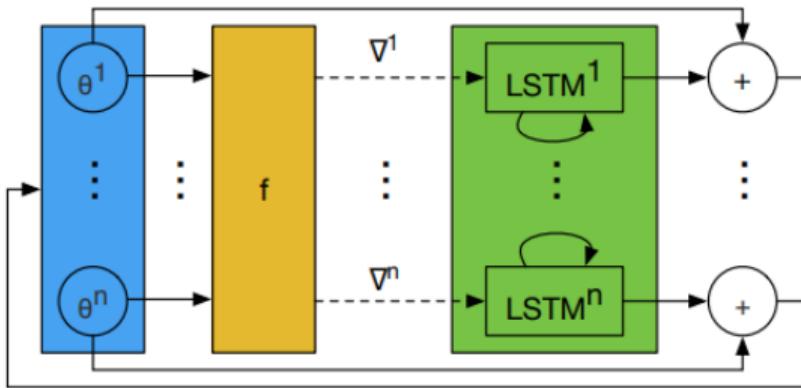


# Learning to Learn: Coordinatewise LSTM optimizer [Andrychowicz et al. 2016]



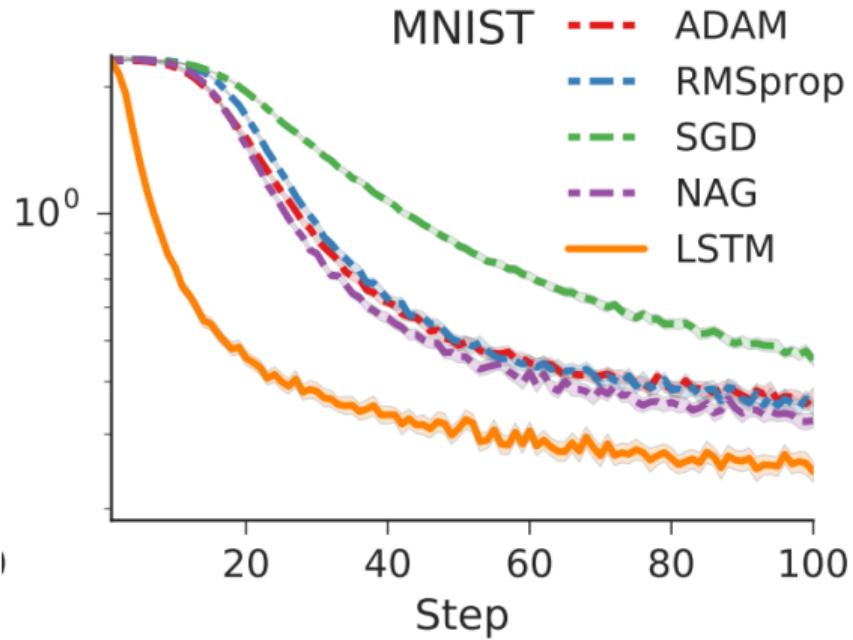
- One LSTM for each coordinate (i.e., weight)
- All LSTMs have shared parameters  $\phi$
- Each coordinate has its own separate hidden state

# Learning to Learn: Coordinatewise LSTM optimizer [Andrychowicz et al. 2016]



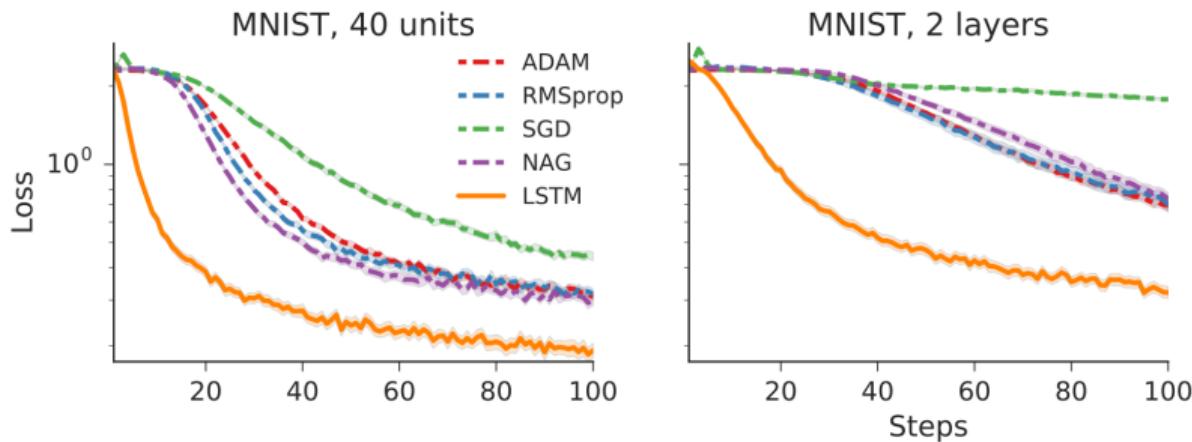
- One LSTM for each coordinate (i.e., weight)
  - All LSTMs have shared parameters  $\phi$
  - Each coordinate has its own separate hidden state
- ↝ We can train the LSTM on  $k$  weights and apply it to larger DNNs with  $k'$  weights, where  $k \leq k'$

# Learning to Learn with LSTM: Results [Andrychowicz et al. 2016]



# Learning to Learn with LSTM: Results [Andrychowicz et al. 2016]

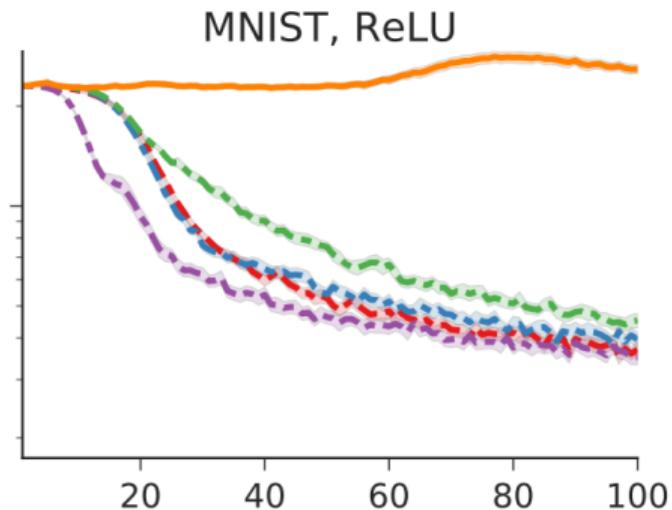
Changing the original architecture of the DNN:



~ learnt optimizer is robust against some architectural changes

# Learning to Learn with LSTM: Results [Andrychowicz et al. 2016]

Changing the activation function to ReLU:



↷ fails on other activation functions

## Black Box Optimization Setting

$$\mathbf{x}^* \in \arg \min_{\mathbf{x} \in \mathcal{X}} f(\mathbf{x})$$

- ① Given the current state of knowledge  $h^{(t)}$  propose a query point  $\mathbf{x}^{(t)}$
- ② Observe the response  $y^{(t)}$
- ③ Update any internal statistics to produce  $h^{(t+1)}$

## Learning Black Box Optimization

Essentially, a similar idea as before:

$$\begin{aligned} h^{(t)}, \mathbf{x}^{(t)} &= \text{RNN}_{\phi}(h^{(t-1)}, \mathbf{x}^{(t-1)}, y^{(t)}) \\ y^{(t)} &\sim p(y|\mathbf{x}^{(t)}) \end{aligned}$$

- Using recurrent neural network (RNN) to predict next  $x_t$ .
- $h^{(t)}$  is the internal hidden state

## Learning Black-box Optimization: Loss Functions [Chen et al. 2017]

- Sum loss: Provides more information than final loss

$$L_{\text{sum}}(\phi) = \mathbb{E}_{f,y^{(1:T-1)}} \left[ \sum_{t=1}^T f(\mathbf{x}^{(t)}) \right]$$

# Learning Black-box Optimization: Loss Functions [Chen et al. 2017]

- Sum loss: Provides more information than final loss

$$L_{\text{sum}}(\phi) = \mathbb{E}_{f,y^{(1:T-1)}} \left[ \sum_{t=1}^T f(\mathbf{x}^{(t)}) \right]$$

- EI loss: Try to learn behavior of Bayesian optimizer based on expected improvement (EI)
  - ▶ requires model (e.g., GP)

$$L_{\text{EI}}(\phi) = -\mathbb{E}_{f,y^{(1:T-1)}} \left[ \sum_{t=1}^T \text{EI}(\mathbf{x}^{(t)} | y^{(1:t-1)}) \right]$$

# Learning Black-box Optimization: Loss Functions [Chen et al. 2017]

- Sum loss: Provides more information than final loss

$$L_{\text{sum}}(\phi) = \mathbb{E}_{f,y^{(1:T-1)}} \left[ \sum_{t=1}^T f(\mathbf{x}^{(t)}) \right]$$

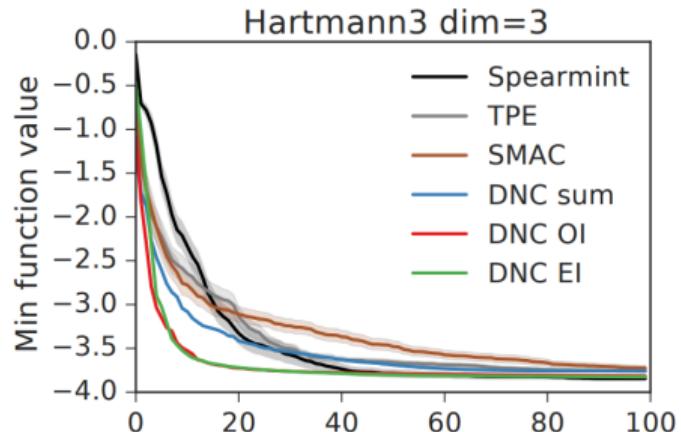
- EI loss: Try to learn behavior of Bayesian optimizer based on expected improvement (EI)
  - ▶ requires model (e.g., GP)

$$L_{\text{EI}}(\phi) = -\mathbb{E}_{f,y^{(1:T-1)}} \left[ \sum_{t=1}^T \text{EI}(\mathbf{x}^{(t)} | y^{(1:t-1)}) \right]$$

- Observed Improvement Loss:

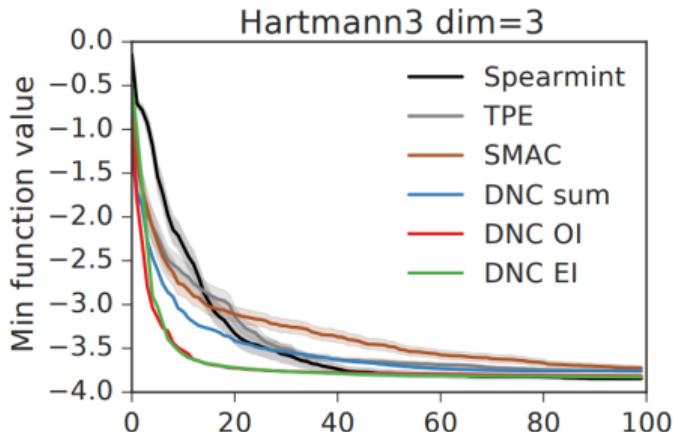
$$L_{\text{OI}}(\phi) = \mathbb{E}_{f,y^{(1:T-1)}} \left[ \sum_{t=1}^T \min \left\{ f(\mathbf{x}^{(t)}) - \min_{i < t} (f(\mathbf{x}^{(i)})), 0 \right\} \right]$$

# Learning Black-box Optimization: Results [Chen et al. 2017]



- Hartmann3 is an artificial function with 3 dimensions

# Learning Black-box Optimization: Results [Chen et al. 2017]



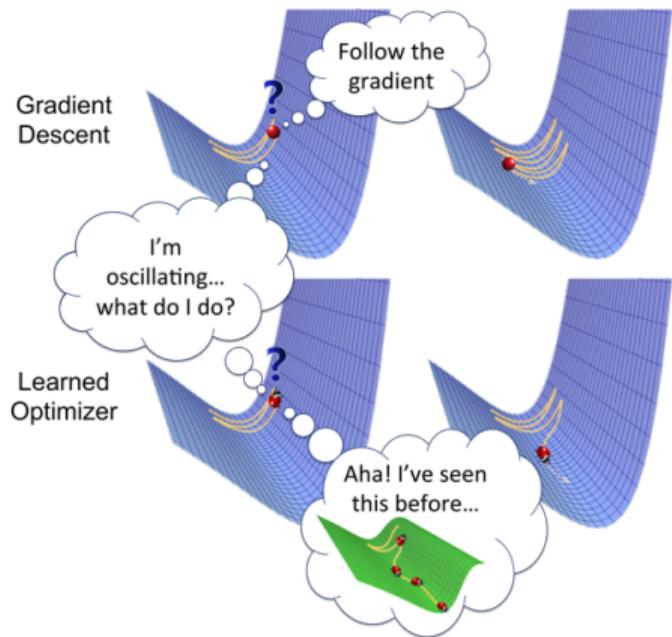
- Hartmann3 is an artificial function with 3 dimensions
  - ↝  $L_{OI}$  and  $L_{EI}$  perform best
  - ↝  $L_{OI}$  easier to compute than  $L_{EI}$  because we need a predictive model to compute EI

# AutoML: Dynamic Configuration & Learning

## Learning to Learn: Reinforcement Learning

Bernd Bischl   Frank Hutter   Lars Kotthoff  
Marius Lindauer   Joaquin Vanschoren

# Learning to Optimize via Reinforcement Learning [Li and Malik. 2017]



Source: <https://bair.berkeley.edu/blog/2017/09/12/learning-to-optimize-with-rl/>

## Reinforcement Learning for Learning to Optimize

**State** current location, objective values and gradients evaluated at the current and past locations

## Reinforcement Learning for Learning to Optimize

**State** current location, objective values and gradients evaluated at the current and past locations

**Action** Step update  $\Delta x$

## Reinforcement Learning for Learning to Optimize

**State** current location, objective values and gradients evaluated at the current and past locations

**Action** Step update  $\Delta \mathbf{x}$

**Transition**  $\mathbf{x}^{(t)} \leftarrow \mathbf{x}^{(t-1)} + \Delta \mathbf{x}$

## Reinforcement Learning for Learning to Optimize

**State** current location, objective values and gradients evaluated at the current and past locations

**Action** Step update  $\Delta \mathbf{x}$

**Transition**  $\mathbf{x}^{(t)} \leftarrow \mathbf{x}^{(t-1)} + \Delta \mathbf{x}$

**Cost/Reward** Objective value at the current location

- Since the RL agent will optimize the cumulative cost, this is equivalent to  $L_{\text{sum}}$  [Chen et al. 2017] ( $\gamma = 0$ )
- encourages the policy to reach the minimum of the objective function as quickly as possible

## Reinforcement Learning for Learning to Optimize

**State** current location, objective values and gradients evaluated at the current and past locations

**Action** Step update  $\Delta \mathbf{x}$

**Transition**  $\mathbf{x}^{(t)} \leftarrow \mathbf{x}^{(t-1)} + \Delta \mathbf{x}$

**Cost/Reward** Objective value at the current location

- Since the RL agent will optimize the cumulative cost, this is equivalent to  $L_{\text{sum}}$  [Chen et al. 2017] ( $\gamma = 0$ )
- encourages the policy to reach the minimum of the objective function as quickly as possible

**Policy** DNN predicting  $\mu_d$  of Gaussian (with constant variance  $\sigma^2$ ) for dimension  $d$ ; sample  $\Delta \mathbf{x}_d \sim \mathcal{N}(\mu_d, \sigma^2)$

## Reinforcement Learning for Learning to Optimize

**State** current location, objective values and gradients evaluated at the current and past locations

**Action** Step update  $\Delta \mathbf{x}$

**Transition**  $\mathbf{x}^{(t)} \leftarrow \mathbf{x}^{(t-1)} + \Delta \mathbf{x}$

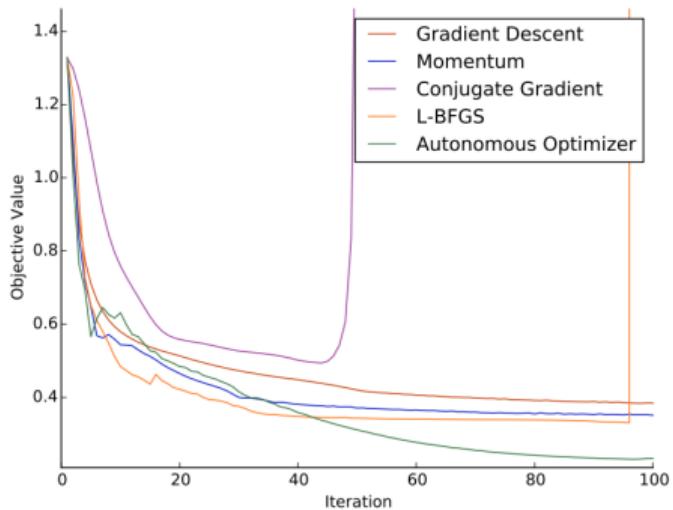
**Cost/Reward** Objective value at the current location

- Since the RL agent will optimize the cumulative cost, this is equivalent to  $L_{\text{sum}}$  [Chen et al. 2017] ( $\gamma = 0$ )
- encourages the policy to reach the minimum of the objective function as quickly as possible

**Policy** DNN predicting  $\mu_d$  of Gaussian (with constant variance  $\sigma^2$ ) for dimension  $d$ ; sample  $\Delta \mathbf{x}_d \sim \mathcal{N}(\mu_d, \sigma^2)$

**Training Set** randomly generated objective functions

# Learning to Optimize via Reinforcement Learning Results [Li and Malik. 2017]



- 2-layer DNN with ReLUs
- Training datasets for training RL agent:  
four multivariate Gaussians and sampling 25 points from each  
~~ hard toy problem

## Learning Acquisition Functions [Volpp et al. 2019]

- Instead of learning everything, it might be sufficient to learn hand-design heuristics

# Learning Acquisition Functions [Volpp et al. 2019]

- Instead of learning everything, it might be sufficient to learn hand-design heuristics
- In Bayesian Optimization (BO), the most critical hand-design heuristic is the acquisition function
  - ▶ trade-off between exploitation and exploration
  - ▶ Depending on the problem at hand, you might need a different acquisition function

# Learning Acquisition Functions [Volpp et al. 2019]

- Instead of learning everything, it might be sufficient to **learn hand-design heuristics**
- In Bayesian Optimization (BO), the most critical hand-design heuristic is the acquisition function
  - ▶ trade-off between exploitation and exploration
  - ▶ Depending on the problem at hand, you might need a different acquisition function
  - ▶ Choices:
    - ★ probability of improvement (PI)
    - ★ expected improvement (EI)
    - ★ upper confidence bounds (UCB)
    - ★ entropy search (ES) – quite expensive!
    - ★ knowledge gradient (KG)
    - ★ ...

# Learning Acquisition Functions [Volpp et al. 2019]

- Instead of learning everything, it might be sufficient to **learn hand-design heuristics**
- In Bayesian Optimization (BO), the most critical hand-design heuristic is the acquisition function
  - ▶ trade-off between exploitation and exploration
  - ▶ Depending on the problem at hand, you might need a different acquisition function
  - ▶ Choices:
    - ★ probability of improvement (PI)
    - ★ expected improvement (EI)
    - ★ upper confidence bounds (UCB)
    - ★ entropy search (ES) – quite expensive!
    - ★ knowledge gradient (KG)
    - ★ ...
- **Idea:** Learn a *neural acquisition function* from data  
~~ Replace acquisition function

# Bayesian Optimization: Algorithm

---

**Algorithm 1** Bayesian Optimization (BO)

---

**Input** : Search Space  $\mathcal{X}$ , black box function  $f$ , acquisition function  $\alpha$ , maximal number of function evaluations  $T$ 

1  $\mathcal{D}^{(0)} \leftarrow \text{initial\_design}(\mathcal{X})$ ;

for  $t = 1, 2, \dots, T - |\mathcal{D}_0|$  do

2      $\hat{c} : \mathbf{x} \mapsto c(\mathbf{x}) \leftarrow \text{fit predictive model on } \mathcal{D}^{(t-1)}$ ;

select  $\mathbf{x}^{(t)}$  by optimizing  $\mathbf{x}^{(t)} \in \arg \max_{\mathbf{x} \in \mathcal{X}} \alpha(\mathbf{x}; \mathcal{D}^{(t-1)}, \hat{c})$ ;

Query  $y^{(t)} := f(\mathbf{x}^{(t)})$ ;

Add observation to data  $\mathcal{D}^{(t)} := \mathcal{D}^{(t-1)} \cup \{\langle \mathbf{x}^{(t)}, y^{(t)} \rangle\}$ ;

3 **return** Best  $\mathbf{x}$  according to  $D$  or  $\hat{c}$

---

## Neural Acquisition Function [Volpp et al. 2019]

Although the acquisition function  $\alpha$  depends on the history  $\mathcal{D}^{(t-1)}$  and the predictive model  $\hat{c}$ ,  $\alpha$  mainly makes use of the predictive mean  $\mu$  and variance  $\sigma^2$ .

## Neural Acquisition Function [Volpp et al. 2019]

Although the acquisition function  $\alpha$  depends on the history  $\mathcal{D}^{(t-1)}$  and the predictive model  $\hat{c}$ ,  $\alpha$  mainly makes use of the predictive mean  $\mu$  and variance  $\sigma^2$ .

Neural acquisition function (AF):

$$\alpha_\theta(\mathbf{x}) = \alpha_\theta(\mu^{(t)}(\mathbf{x}), \sigma^{(t)}(\mathbf{x}), \mathbf{x}, t, T)$$

where  $\theta$  are the parameters of a neural network, and  $\mu, \sigma, \mathbf{x}, t, T$  are its inputs.

## RL to train Neural AF [Volpp et al. 2019]

Policy  $\pi_\theta$ : Neural acquisition function  $\alpha_\theta$

## RL to train Neural AF [Volpp et al. 2019]

Policy  $\pi_\theta$ : Neural acquisition function  $\alpha_\theta$

Episode: run of  $\pi$  on  $f \in \mathcal{F}'$

- $\mathcal{F}$  is a set of functions we can sample functions from

## RL to train Neural AF [Volpp et al. 2019]

Policy  $\pi_\theta$ : Neural acquisition function  $\alpha_\theta$

Episode: run of  $\pi$  on  $f \in \mathcal{F}'$

- $\mathcal{F}$  is a set of functions we can sample functions from

State  $s^{(t)}$ :  $\mu^{(t)}$  and  $\sigma^{(t)}$  on a set of points  $\xi^{(t)}$ , and  $t$  and  $T$

## RL to train Neural AF [Volpp et al. 2019]

Policy  $\pi_\theta$ : Neural acquisition function  $\alpha_\theta$

Episode: run of  $\pi$  on  $f \in \mathcal{F}'$

- $\mathcal{F}$  is a set of functions we can sample functions from

State  $s^{(t)}$ :  $\mu^{(t)}$  and  $\sigma^{(t)}$  on a set of points  $\xi^{(t)}$ , and  $t$  and  $T$

Action  $a^{(t)}$ : Sampled point  $\mathbf{x}^{(t)} \in \xi^{(t)}$

## RL to train Neural AF [Volpp et al. 2019]

Policy  $\pi_\theta$ : Neural acquisition function  $\alpha_\theta$

Episode: run of  $\pi$  on  $f \in \mathcal{F}'$

- $\mathcal{F}$  is a set of functions we can sample functions from

State  $s^{(t)}$ :  $\mu^{(t)}$  and  $\sigma^{(t)}$  on a set of points  $\xi^{(t)}$ , and  $t$  and  $T$

Action  $a^{(t)}$ : Sampled point  $\mathbf{x}^{(t)} \in \xi^{(t)}$

Reward  $r^{(t)}$ : negative simple regret:  $r^{(t)} = f(\mathbf{x}^*) - f(\hat{\mathbf{x}})$

- assumes that we can estimate the optimal  $\mathbf{x}^*$  for *training* functions

## RL to train Neural AF [Volpp et al. 2019]

Policy  $\pi_\theta$ : Neural acquisition function  $\alpha_\theta$

Episode: run of  $\pi$  on  $f \in \mathcal{F}'$

- $\mathcal{F}$  is a set of functions we can sample functions from

State  $s^{(t)}$ :  $\mu^{(t)}$  and  $\sigma^{(t)}$  on a set of points  $\xi^{(t)}$ , and  $t$  and  $T$

Action  $a^{(t)}$ : Sampled point  $\mathbf{x}^{(t)} \in \xi^{(t)}$

Reward  $r^{(t)}$ : negative simple regret:  $r^{(t)} = f(\mathbf{x}^*) - f(\hat{\mathbf{x}})$

- assumes that we can estimate the optimal  $\mathbf{x}^*$  for *training* functions

Transition probability : Noisy evaluation of  $f$  and the predictive model update

- The state is described by a discrete set of points  $\xi^{(t)} = \{\xi_n\}_{n=1}^N$

- The state is described by a discrete set of points  $\xi^{(t)} = \{\xi_n\}_{n=1}^N$
- We feed these points through the predictive model and the neural AF to obtain  $\alpha_\theta(\xi_n) = \alpha_\theta(\mu^{(t)}(\xi_n), \sigma^{(t)}(\xi_n), \xi_n, t, T)$ ,

- The state is described by a discrete set of points  $\xi^{(t)} = \{\xi_n\}_{n=1}^N$
- We feed these points through the predictive model and the neural AF to obtain  $\alpha_\theta(\xi_n) = \alpha_\theta(\mu^{(t)}(\xi_n), \sigma^{(t)}(\xi_n), \xi_n, t, T)$ ,
- $\alpha_\theta(\xi_i)$  are interpreted as the logits of categorical distribution, s.t.

$$\pi_\alpha(\cdot \mid s^{(t)}) = \text{Cat} [\alpha_\theta(\xi_1), \dots, \alpha_\theta(\xi_N)]$$

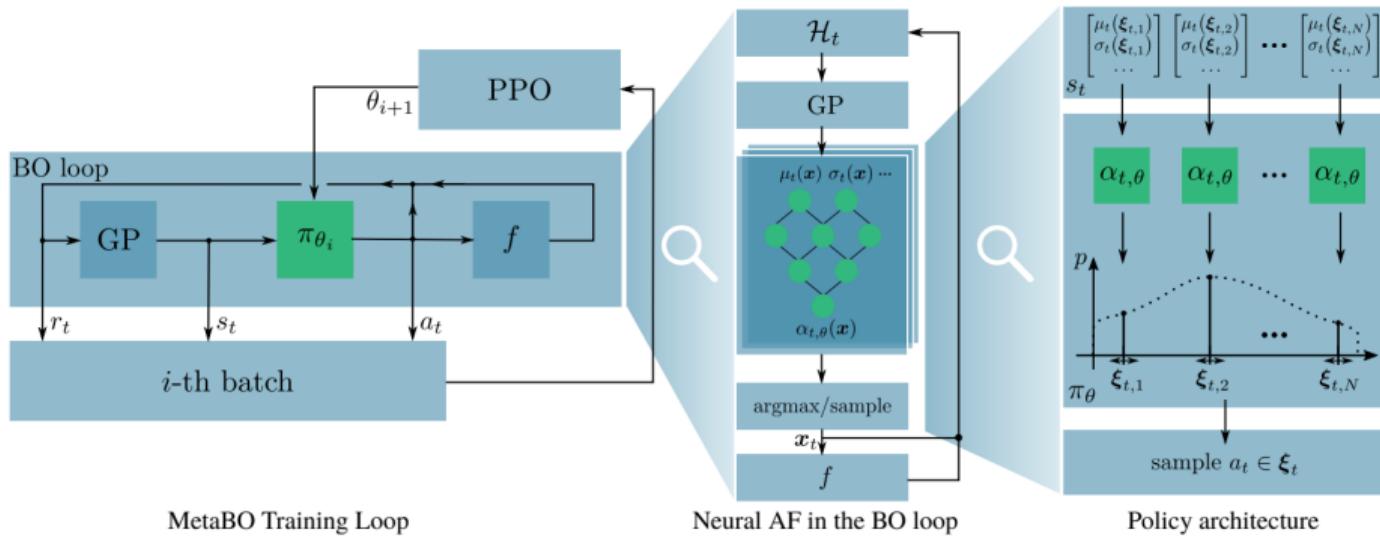
- The state is described by a discrete set of points  $\xi^{(t)} = \{\xi_n\}_{n=1}^N$
- We feed these points through the predictive model and the neural AF to obtain  $\alpha_\theta(\xi_n) = \alpha_\theta(\mu^{(t)}(\xi_n), \sigma^{(t)}(\xi_n), \xi_n, t, T)$ ,
- $\alpha_\theta(\xi_i)$  are interpreted as the logits of categorical distribution, s.t.

$$\pi_\alpha(\cdot \mid s^{(t)}) = \text{Cat} [\alpha_\theta(\xi_1), \dots, \alpha_\theta(\xi_N)]$$

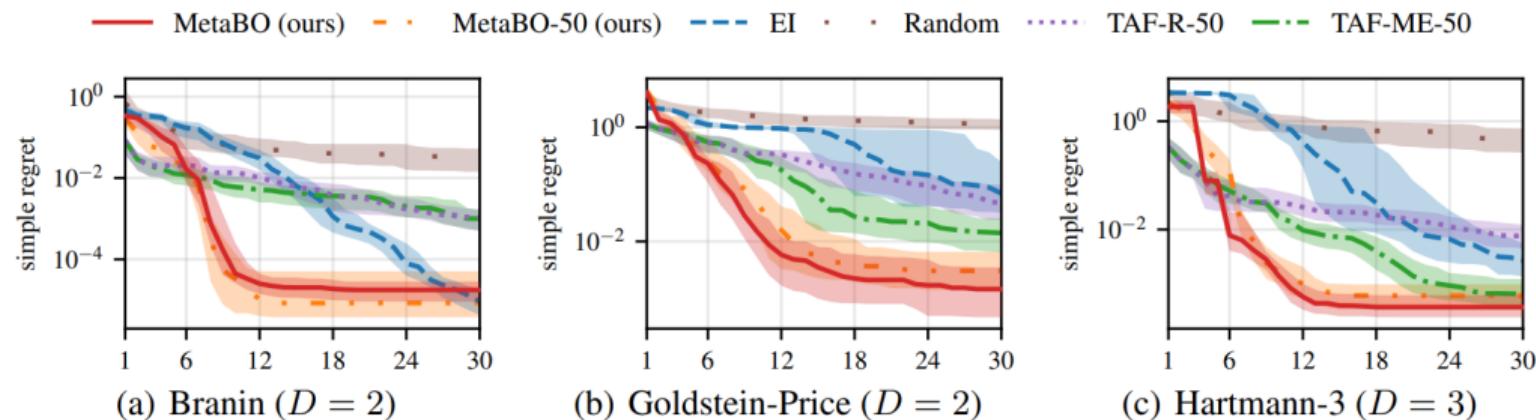
- Due to curse of dimensionality, we need a two step approach for  $\xi^{(t)}$ 
  - ➊ sample  $\xi_{\text{global}}$  using a coarse Sobol grid
  - ➋ sample  $\xi_{\text{local}}$  using local optimization starting from the best samples in  $\xi_{\text{global}}$

↝  $\xi^{(t)} = \xi_{\text{global}} \cup \xi_{\text{local}}$

# Learning Acquisition Functions: Overview [Volpp et al. 2019]

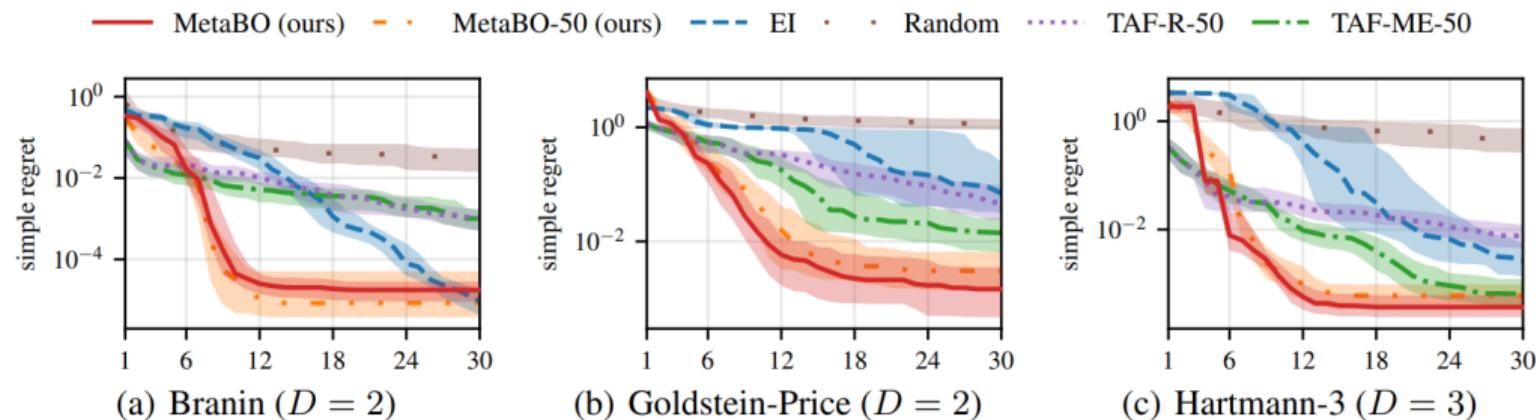


# Results on Artificial Functions [Volpp et al. 2019]



- Approach by [Volpp et al. 2019] called MetaBO
- MetaBO performs better than other acquisition functions (EI, GP-UCB, PI) and other baselines (Random, TAF)

# Results on Artificial Functions [Volpp et al. 2019]



- Approach by [Volpp et al. 2019] called MetaBO
- MetaBO performs better than other acquisition functions (EI, GP-UCB, PI) and other baselines (Random, TAF)

**Assumption:** You have a family of functions at hand that resembles your target function.

# AutoML: Interpretability

## Overview: Automated Empirical Analysis

Bernd Bischl   Frank Hutter   Lars Kotthoff  
Marius Lindauer   Joaquin Vanschoren

# Idea

- Big challenge of ML: Interpretability
  - ▶ In some applications, it is required to "understand" a prediction
  - ▶ Users have less trust in systems, they can't understand

# Idea

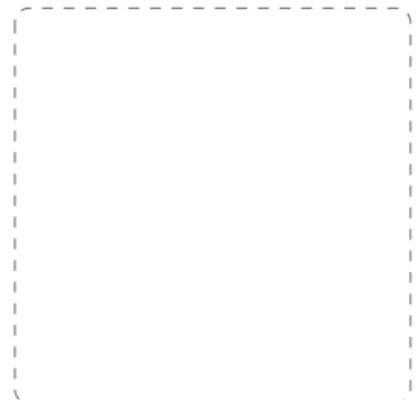
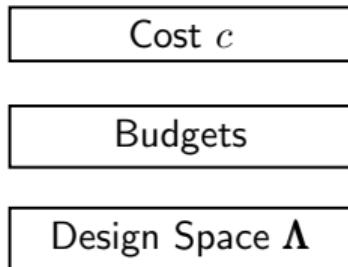
- Big challenge of ML: Interpretability
  - ▶ In some applications, it is required to "understand" a prediction
  - ▶ Users have less trust in systems, they can't understand
- AutoML is even worse?
  - ▶ AutoML is a black-box that automates the design of another blackbox (ML)
  - ▶ Also ML-developers have an basic understanding of the design of their ML pipelines
- Automated empirical interpretability helps to
  - ▶ understand the finally returned ML system
  - ▶ understand the AutoML process

# Approach

- Insights:

- ▶ AutoML is yet another optimization problem
- ▶ (Most) AutoML approach are iterative in nature

~~~ AutoML generates a lot of empirical data

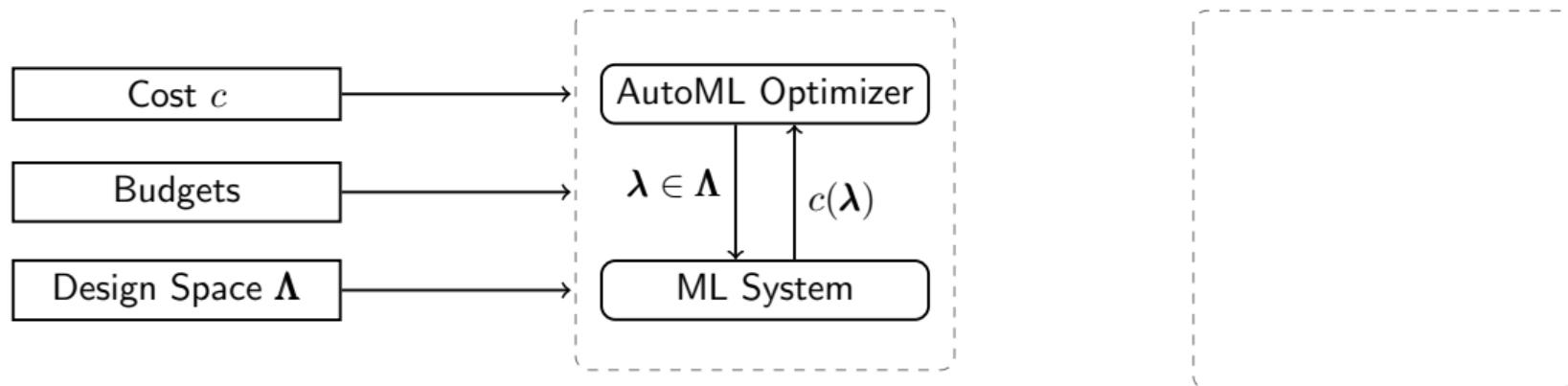


Approach

- Insights:

- ▶ AutoML is yet another optimization problem
- ▶ (Most) AutoML approach are iterative in nature

~~~ AutoML generates a lot of empirical data

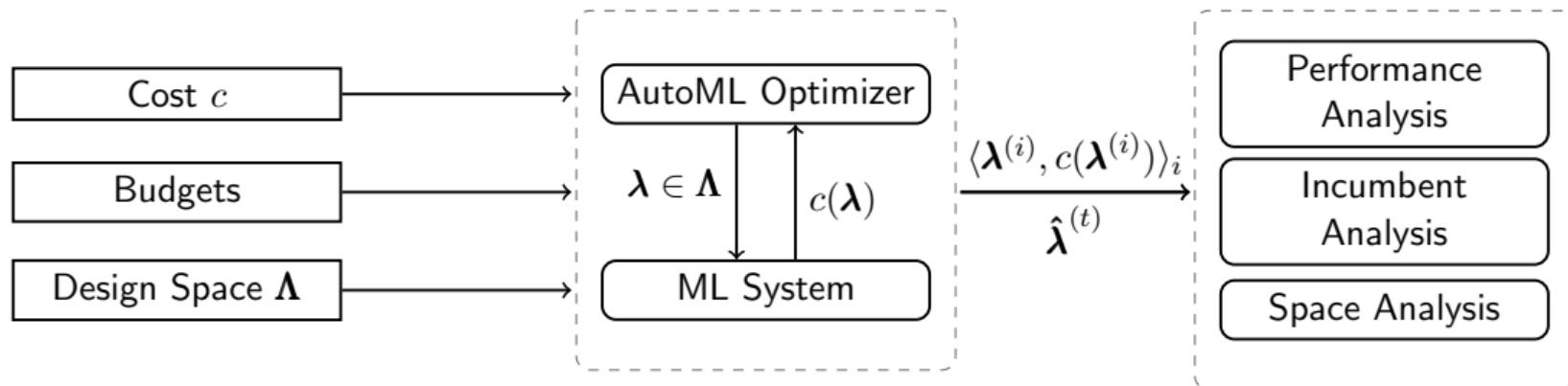


# Approach

- Insights:

- ▶ AutoML is yet another optimization problem
- ▶ (Most) AutoML approach are iterative in nature

↝ AutoML generates a lot of empirical data

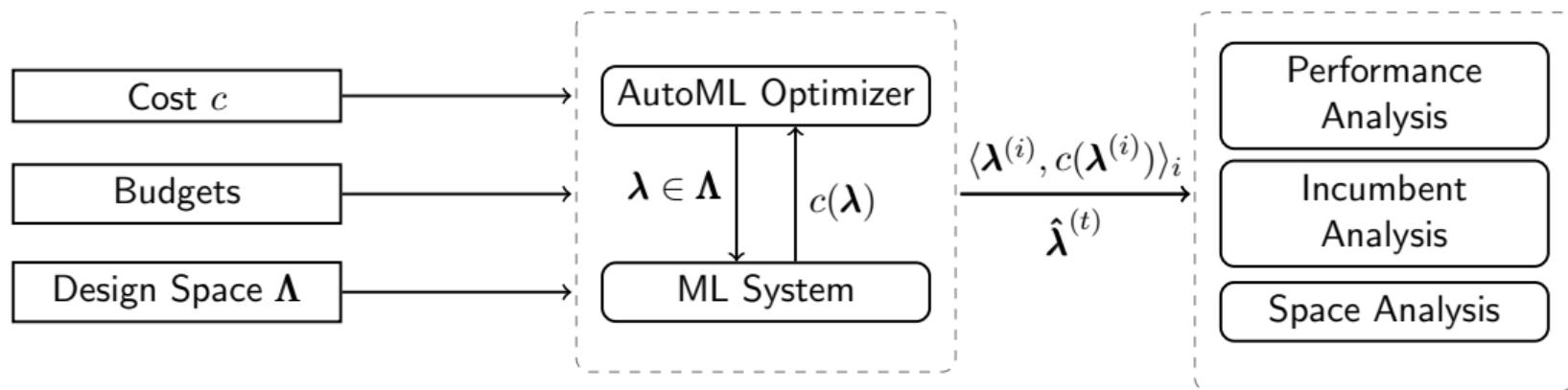


# Approach

- Insights:

- ▶ AutoML is yet another optimization problem
- ▶ (Most) AutoML approach are iterative in nature

~~ AutoML generates a lot of empirical data



~~ Let's use this data to learn something about our AutoML problem

## Basic Examples

- Visualize final incumbent  $\hat{\lambda}$ 
  - ▶ ML pipeline with its components
  - ▶ Neural architecture

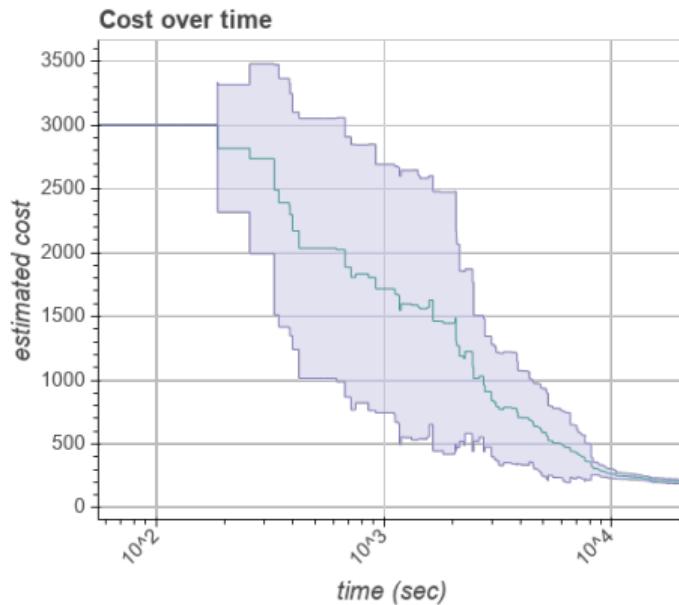
## Basic Examples

- Visualize final incumbent  $\hat{\lambda}$ 
  - ▶ ML pipeline with its components
  - ▶ Neural architecture
- Compare what changed between  $\lambda_{\text{def}}$  and  $\hat{\lambda}$

## Basic Examples

- Visualize final incumbent  $\hat{\lambda}$ 
  - ▶ ML pipeline with its components
  - ▶ Neural architecture
- Compare what changed between  $\lambda_{\text{def}}$  and  $\hat{\lambda}$
- Show  $\hat{\lambda}$  on different budgets (if you used a multi-fidelity approach)

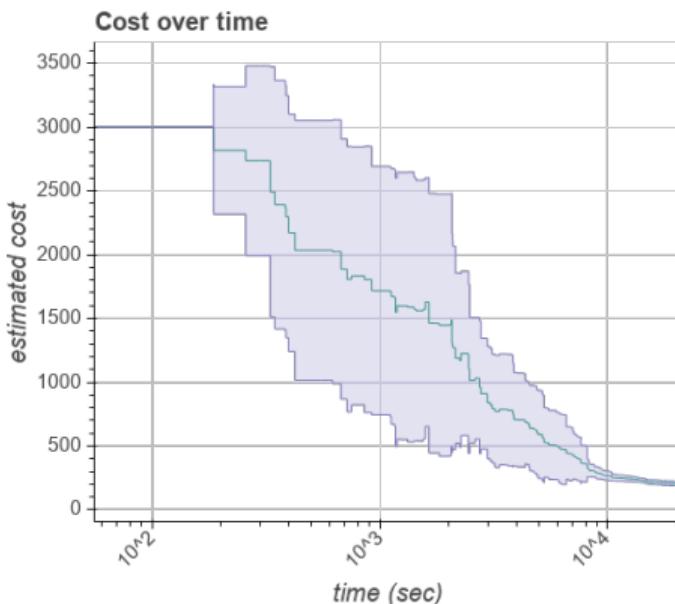
# Cost Over Time



- Study how your AutoML tool improves cost (or loss) over time

Source: [Lindauer et al. 2019]

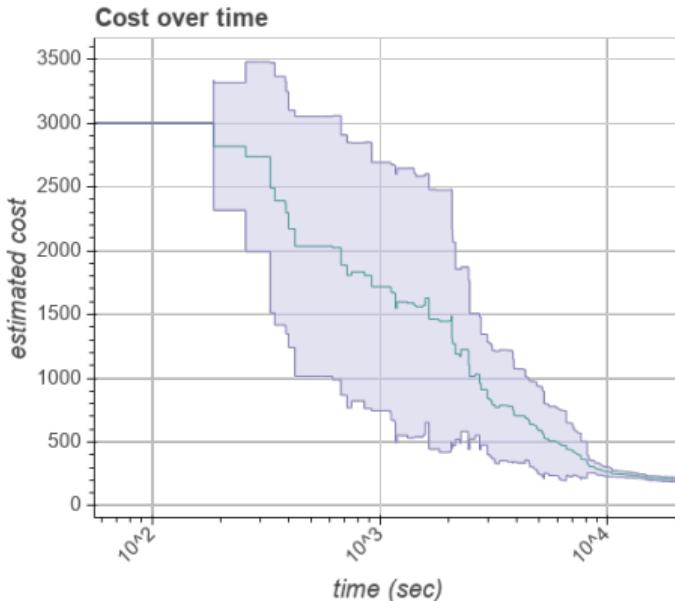
# Cost Over Time



- Study how your AutoML tool improves cost (or loss) over time
- Allows to identify whether
  - ▶ you need less time next time or

Source: [Lindauer et al. 2019]

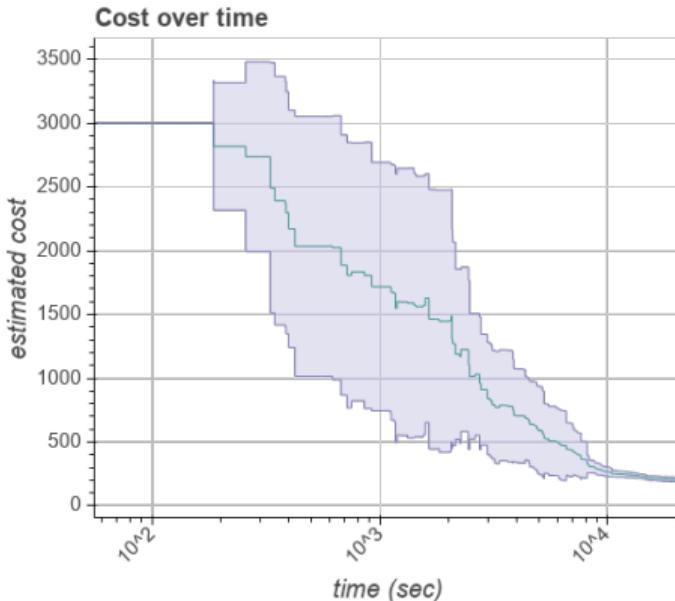
# Cost Over Time



- Study how your AutoML tool improves cost (or loss) over time
- Allows to identify whether
  - ▶ you need less time next time or
  - ▶ the AutoML system is still improving; so you should give it more time

Source: [Lindauer et al. 2019]

# Cost Over Time



- Study how your AutoML tool improves cost (or loss) over time
- Allows to identify whether
  - ▶ you need less time next time or
  - ▶ the AutoML system is still improving; so you should give it more time
- Notes:
  - ▶ Plot on log-scale to see details in the beginning
  - ▶ If you done several runs, plot distribution (e.g., median and 25/75%-quartiles)

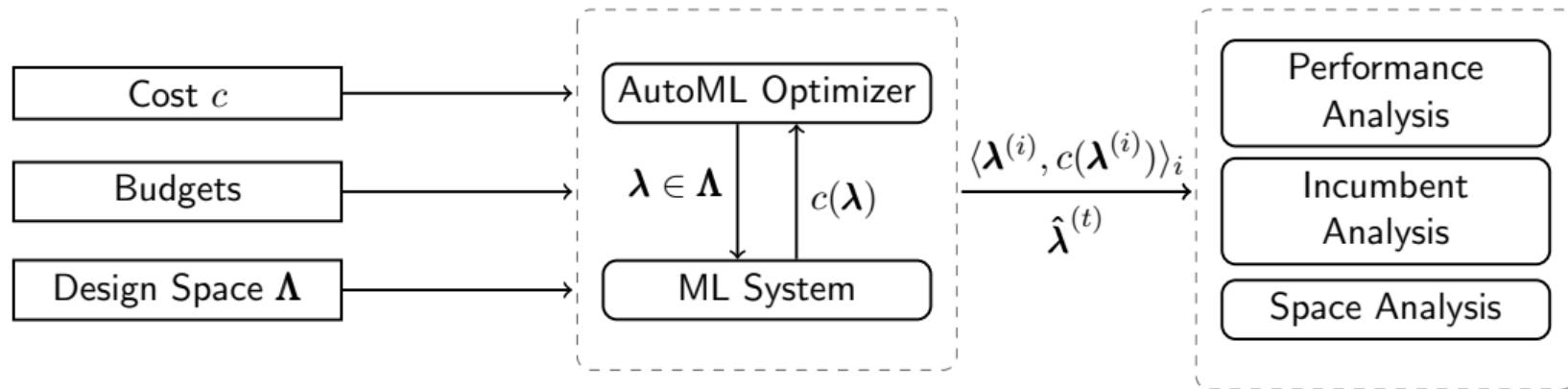
Source: [Lindauer et al. 2019]

# AutoML: Interpretability

## Studying the AutoML Optimization Process

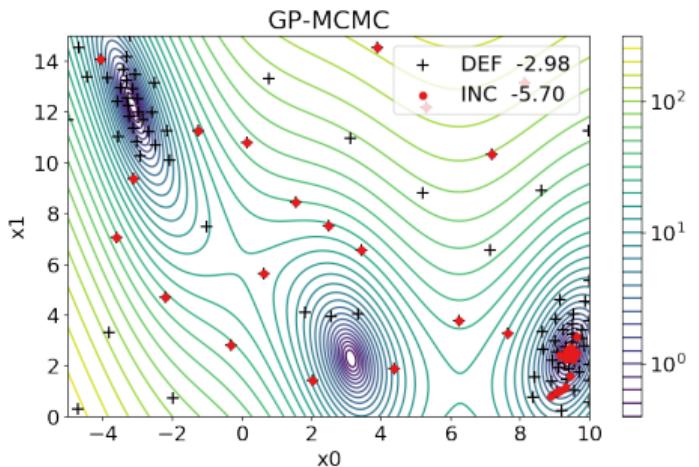
Bernd Bischl   Frank Hutter   Lars Kotthoff  
Marius Lindauer   Joaquin Vanschoren

# Idea



→ focus on how the AutoML optimizer samples from the design space  $\Lambda$

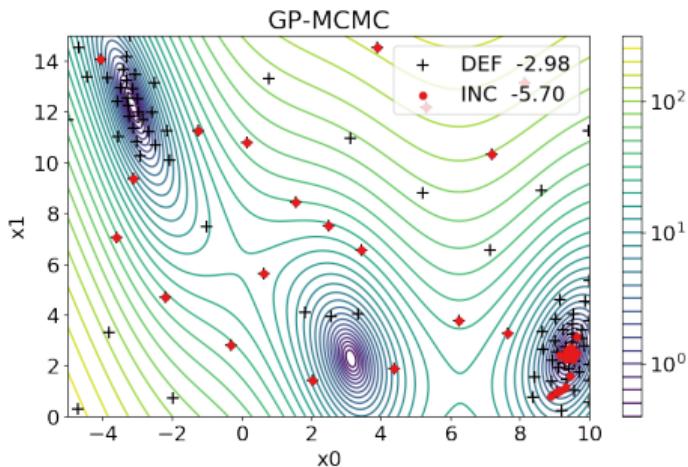
# Analyzing the Sampling Behavior



- Plot of a 1D or 2D function

Source: [Lindauer et al. 2019]

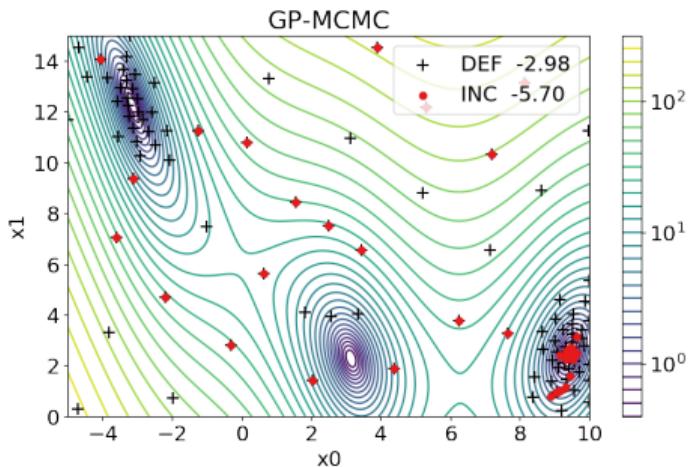
# Analyzing the Sampling Behavior



- Plot of a 1D or 2D function
- Background shows the ground truth (real function values)

Source: [Lindauer et al. 2019]

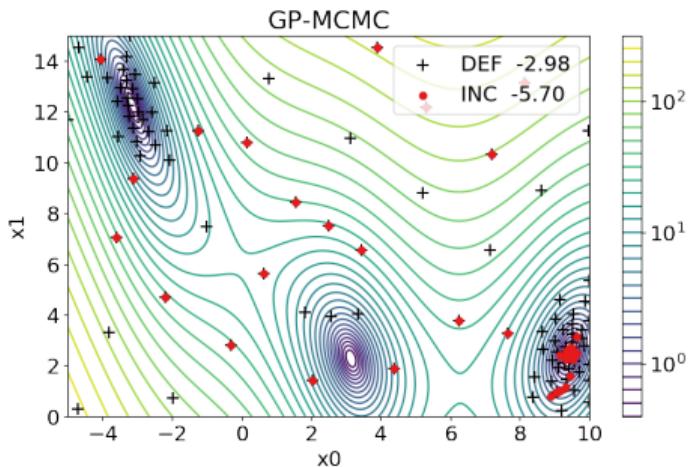
# Analyzing the Sampling Behavior



- Plot of a 1D or 2D function
- Background shows the ground truth (real function values)
- Dots are sampled points in the search space

Source: [Lindauer et al. 2019]

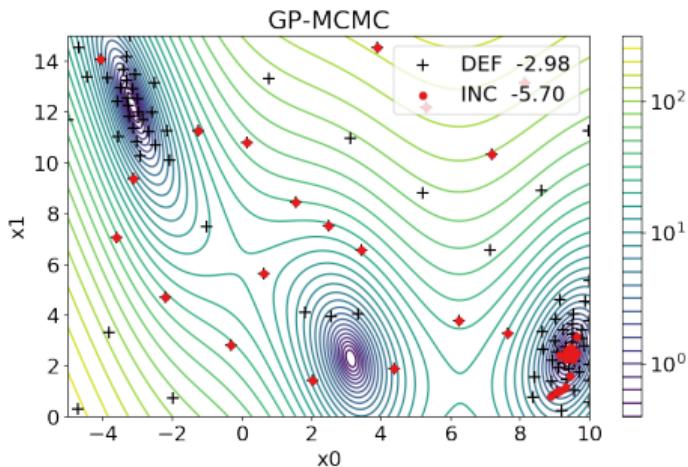
# Analyzing the Sampling Behavior



- Plot of a 1D or 2D function
- Background shows the ground truth (real function values)
- Dots are sampled points in the search space
- Typical approach in Bayesian Optimization community

Source: [Lindauer et al. 2019]

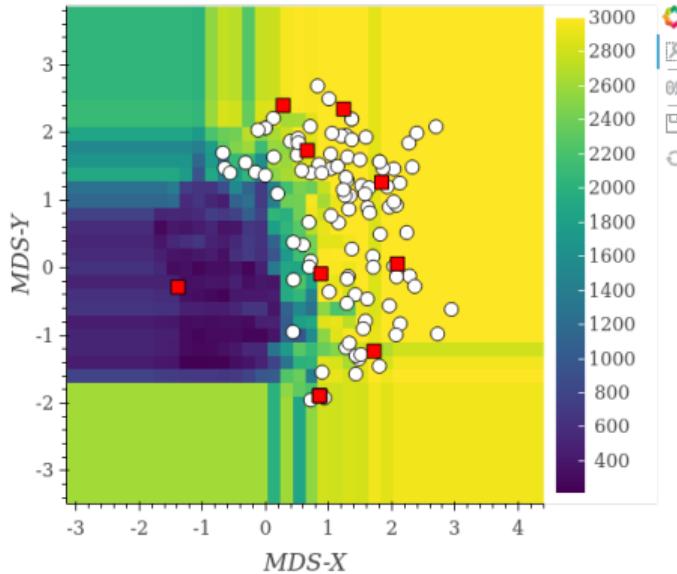
# Analyzing the Sampling Behavior



- Plot of a 1D or 2D function
  - Background shows the ground truth (real function values)
  - Dots are sampled points in the search space
  - Typical approach in Bayesian Optimization community
- ↝ Impossible for higher dimensional problems?

Source: [Lindauer et al. 2019]

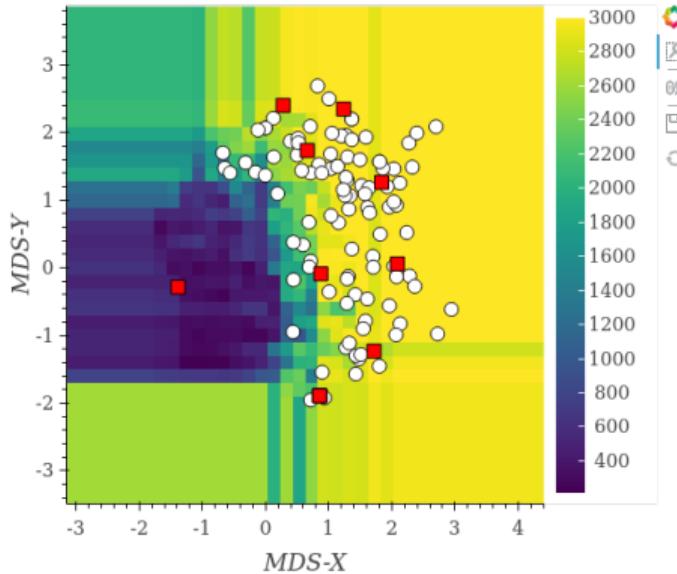
# Analyzing the Sampling Behavior in $N$ -D



- Same idea as before  
but we have to project  $N$ -D into 2-D

Source: [Lindauer et al. 2019]

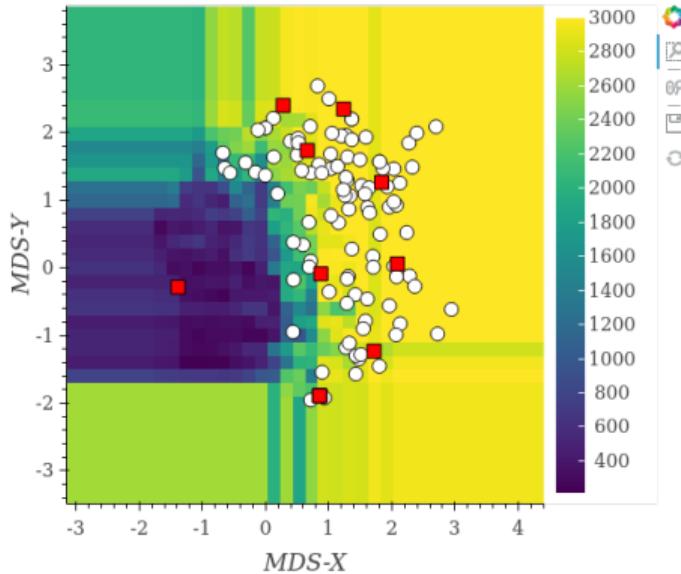
# Analyzing the Sampling Behavior in $N$ -D



- Same idea as before  
but we have to project  $N$ -D into 2-D
  - ➊ Use an MDS to project down to 2-D

Source: [Lindauer et al. 2019]

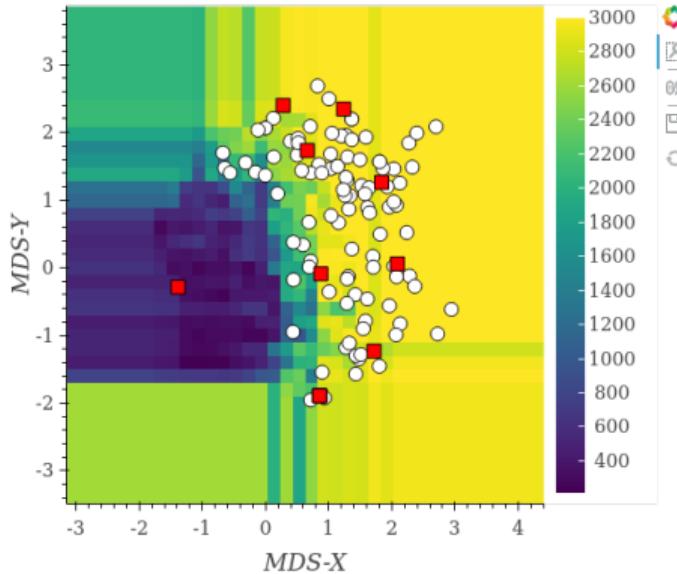
# Analyzing the Sampling Behavior in $N$ -D



- Same idea as before  
but we have to project  $N$ -D into 2-D
  - ① Use an MDS to project down to 2-D
  - ② Each dot is single hyperparameter configuration

Source: [Lindauer et al. 2019]

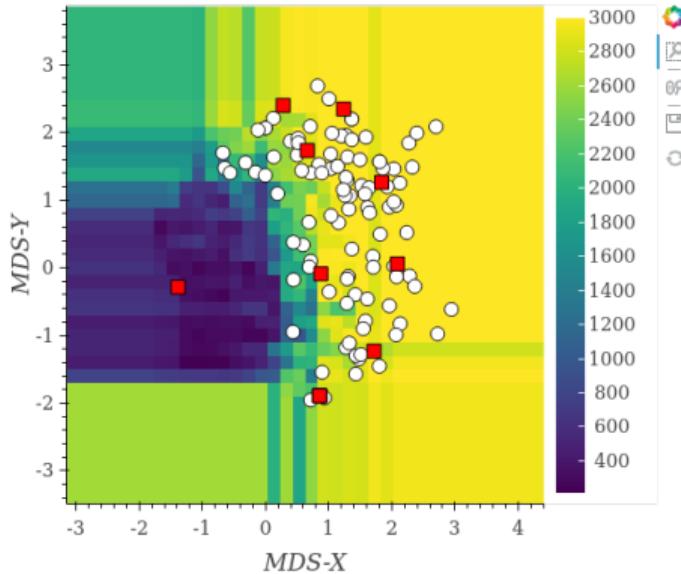
# Analyzing the Sampling Behavior in $N$ -D



- Same idea as before  
but we have to project  $N$ -D into 2-D
  - ① Use an MDS to project down to 2-D
  - ② Each dot is single hyperparameter configuration
  - ③ Red squares are intermediate incumbents

Source: [Lindauer et al. 2019]

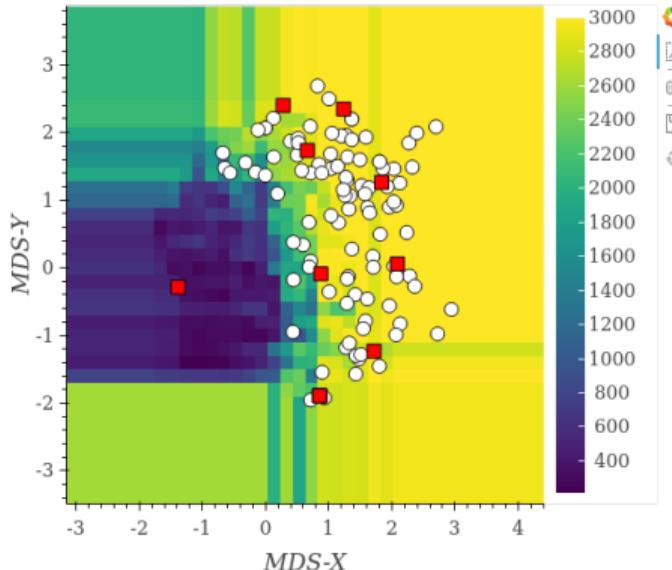
# Analyzing the Sampling Behavior in $N$ -D



- Same idea as before  
but we have to project  $N$ -D into 2-D
  - ① Use an MDS to project down to 2-D
  - ② Each dot is single hyperparameter configuration
  - ③ Red squares are intermediate incumbents
  - ④ The background is colored wrt a performance-estimate  
(e.g., reusing model fitted during BO)

Source: [Lindauer et al. 2019]

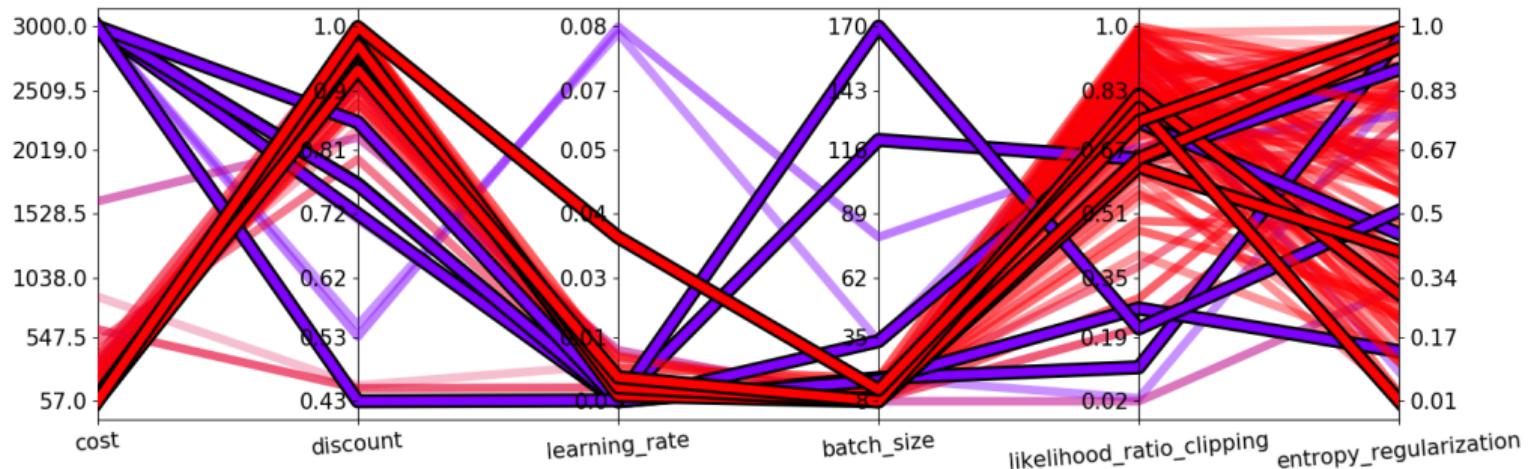
# Analyzing the Sampling Behavior in $N$ -D



- Same idea as before  
but we have to project  $N$ -D into 2-D
  - ① Use an MDS to project down to 2-D
  - ② Each dot is single hyperparameter configuration
  - ③ Red squares are intermediate incumbents
  - ④ The background is colored wrt a performance-estimate  
(e.g., reusing model fitted during BO)
  - ⑤ Extension: Animation by showing how points get added over time

Source: [Lindauer et al. 2019]

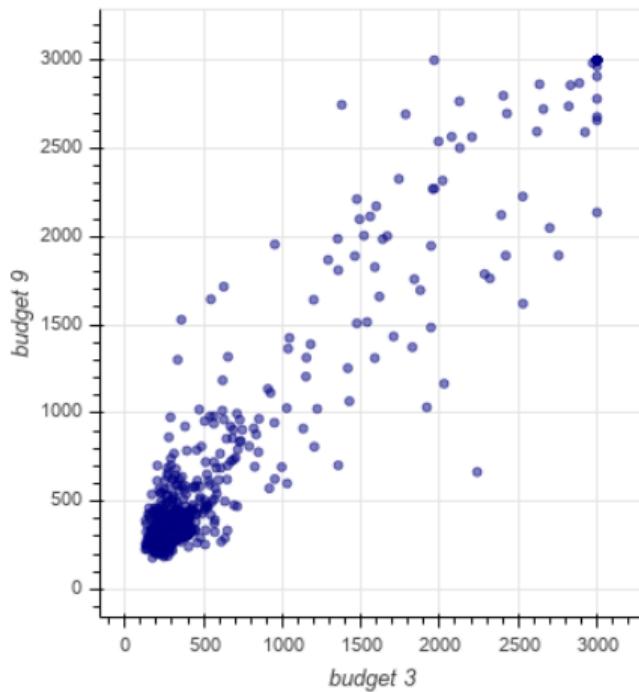
# Parallel Coordinate Plot [Golovin et al. 2017]



Source: [Lindauer et al. 2019]

- Each coordinate is one hyperparameter;
- Except the most left one: cost or loss

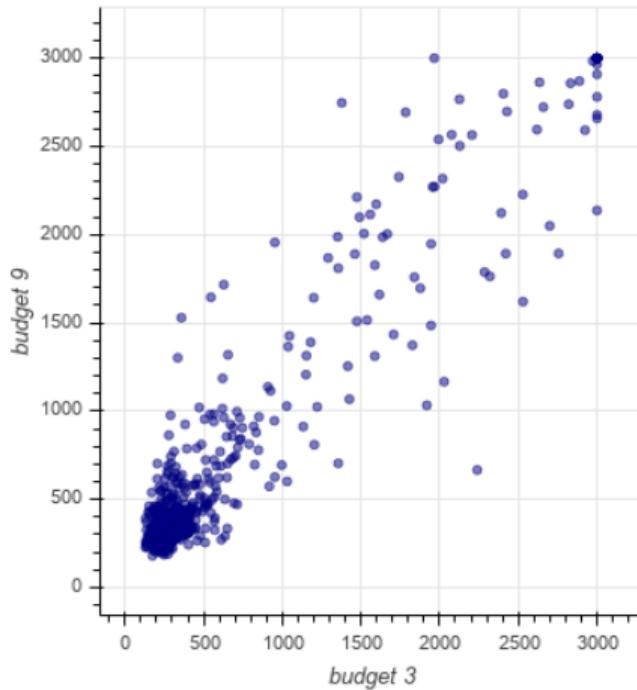
# Multi-Fidelity Checks



- Challenge of multi-fidelity approaches:
  - ▶ How to choose the fidelities (a.k.a. budgets)

Source: [Lindauer et al. 2019]

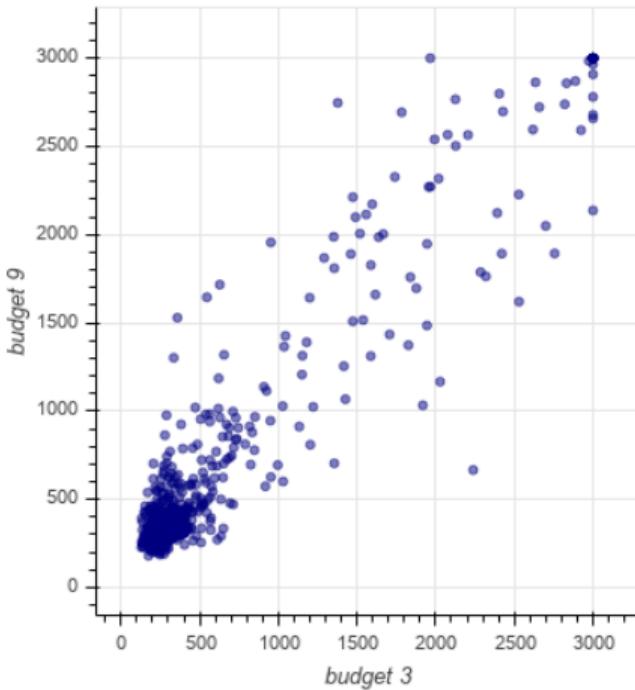
# Multi-Fidelity Checks



- Challenge of multi-fidelity approaches:
  - ▶ How to choose the fidelities (a.k.a. budgets)
- Important Property:
  - ▶ Decisions on small budgets should be reasonable for higher budgets

Source: [Lindauer et al. 2019]

# Multi-Fidelity Checks



- Challenge of multi-fidelity approaches:
  - ▶ How to choose the fidelities (a.k.a. budgets)
- Important Property:
  - ▶ Decisions on small budgets should be reasonable for higher budgets
- Analysis:
  - ① Scatter plot of performance on Budget X vs. Budget Y
  - ② Each dot is sampled hyperparameter configuration
  - ③ Compute rank correlation (here: 0.69)

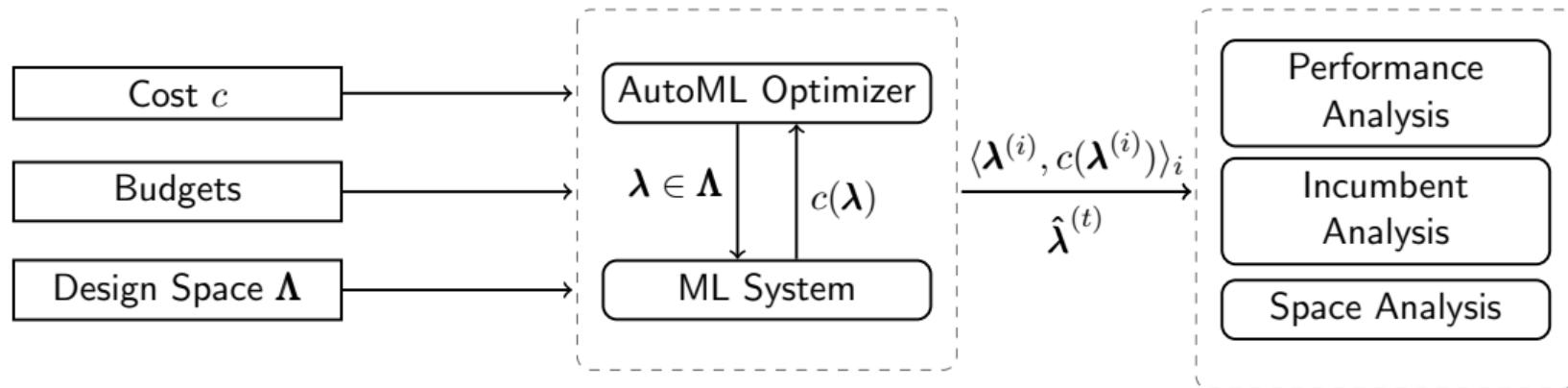
Source: [Lindauer et al. 2019]

# AutoML: Interpretability

## Incumbent Analysis and Local Hyperparameter Importance

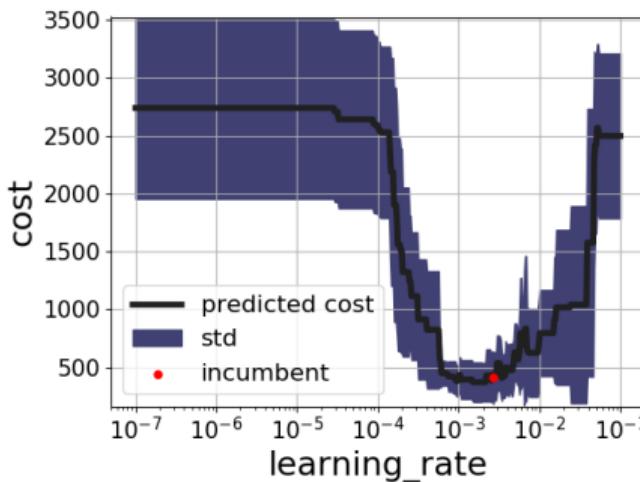
Bernd Bischl   Frank Hutter   Lars Kotthoff  
Marius Lindauer   Joaquin Vanschoren

# Idea



↗ focus on why is the eventually returned configuration a good choice

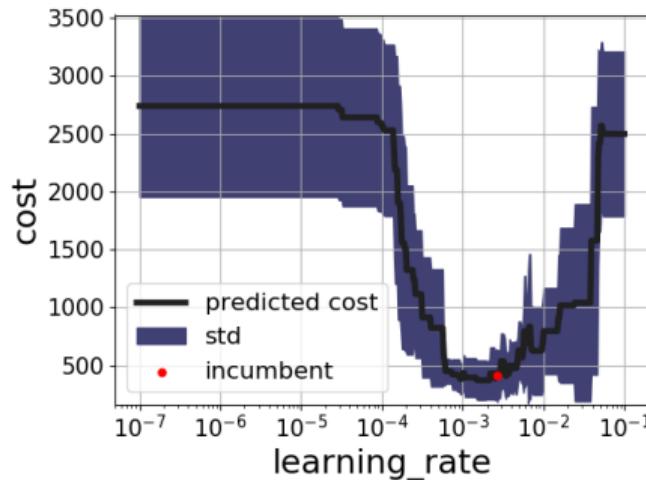
## Local Importance [Biedenkapp et al. 2018]



- Typical question of users:
  - ▶ How would the performance change if we change hyperparameter  $\lambda_i$ ?

Source: [Lindauer et al. 2019]

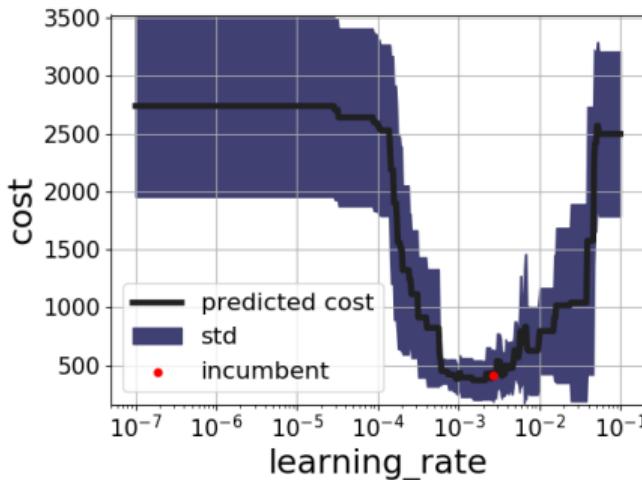
## Local Importance [Biedenkapp et al. 2018]



- Typical question of users:
  - ▶ How would the performance change if we change hyperparameter  $\lambda_i$ ?
- Problem: Running full study is often too expensive
  - ▶ Each run of an ML-system is potential expensive

Source: [Lindauer et al. 2019]

## Local Importance [Biedenkapp et al. 2018]



- Typical question of users:
  - ▶ How would the performance change if we change hyperparameter  $\lambda_i$ ?
- Problem: Running full study is often too expensive
  - ▶ Each run of an ML-system is potential expensive
- Key Ideas:
  - ▶ Re-use probabilistic models as trained in BO
  - ▶ Plot performance change around  $\hat{\lambda}^{(t)}$  along each dimension

Source: [Lindauer et al. 2019]

# Quantifying Local Importance [Biedenkapp et al. 2018]

$$\text{VAR}_{\boldsymbol{\lambda}}(i) = \sum_{v \in \Lambda_i} (\mathbb{E}_{v \sim \Lambda_i} [L(\boldsymbol{\lambda})] - L(\boldsymbol{\lambda}[\lambda_i := v]))^2 \quad (1)$$

## Quantifying Local Importance [Biedenkapp et al. 2018]

$$\text{VAR}_{\lambda}(i) = \sum_{v \in \Lambda_i} (\mathbb{E}_{v \sim \Lambda_i} [L(\lambda)] - L(\lambda[\lambda_i := v]))^2 \quad (1)$$

$$\text{LPI}(i | \lambda) = \frac{\text{VAR}_{\lambda}(i)}{\sum_j \text{VAR}_{\lambda}(j)} \quad (2)$$

# Quantifying Local Importance [Biedenkapp et al. 2018]

$$\text{VAR}_{\lambda}(i) = \sum_{v \in \Lambda_i} (\mathbb{E}_{v \sim \Lambda_i} [L(\lambda)] - L(\lambda[\lambda_i := v]))^2 \quad (1)$$

$$\text{LPI}(i | \lambda) = \frac{\text{VAR}_{\lambda}(i)}{\sum_j \text{VAR}_{\lambda}(j)} \quad (2)$$

- ~ While fixing all other hyperparameters to the incumbent value, the hyperparameter with the highest variance is the most important one

## Ablation Study for Importance

- Users often start from some kind of default configuration
  - ① As given in the documentation
  - ② Or as always used in the last time

# Ablation Study for Importance

- Users often start from some kind of default configuration
  - ① As given in the documentation
  - ② Or as always used in the last time
- Key Idea: Going from the default to the automatically optimized configuration, which choices where important?

$$\lambda^{(\text{start})} = [1, 1, 0, 100]$$

$$\lambda^{(\text{end})} = [0.98, 2.42, 1, 42]$$

# Ablation Study for Importance

- Users often start from some kind of default configuration
  - ① As given in the documentation
  - ② Or as always used in the last time
- Key Idea: Going from the default to the automatically optimized configuration, which choices were important?

$$\begin{aligned}\lambda^{(\text{start})} &= [1, 1, 0, 100] \\ \lambda^{(\text{end})} &= [0.98, 2.42, 1, 42]\end{aligned}$$

- Cheap approach: Assess  $\lambda^{(\text{end})}$  with each hyperparameter value from  $\lambda^{(\text{start})}$

# Ablation Study for Importance

- Users often start from some kind of default configuration
  - ① As given in the documentation
  - ② Or as always used in the last time
- Key Idea: Going from the default to the automatically optimized configuration, which choices were important?

$$\begin{aligned}\lambda^{(\text{start})} &= [1, 1, 0, 100] \\ \lambda^{(\text{end})} &= [0.98, 2.42, 1, 42]\end{aligned}$$

- Cheap approach: Assess  $\lambda^{(\text{end})}$  with each hyperparameter value from  $\lambda^{(\text{start})}$
- Expensive approach: Try all mixtures of  $\lambda^{(\text{end})}$  and  $\lambda^{(\text{start})}$ 
  - ▶ Only feasible for small spaces and fairly cheap ML systems

# Ablation Study for Importance

- Users often start from some kind of default configuration
  - ① As given in the documentation
  - ② Or as always used in the last time
- Key Idea: Going from the default to the automatically optimized configuration, which choices were important?

$$\begin{aligned}\lambda^{(\text{start})} &= [1, 1, 0, 100] \\ \lambda^{(\text{end})} &= [0.98, 2.42, 1, 42]\end{aligned}$$

- Cheap approach: Assess  $\lambda^{(\text{end})}$  with each hyperparameter value from  $\lambda^{(\text{start})}$
- Expensive approach: Try all mixtures of  $\lambda^{(\text{end})}$  and  $\lambda^{(\text{start})}$ 
  - ▶ Only feasible for small spaces and fairly cheap ML systems
- Trade-off: Find a way from  $\lambda^{(\text{start})}$  to  $\lambda^{(\text{end})}$  in a greedy fashion [Fawcett and Hoos. 2016]

# Greedy Ablation Study

Given:

$$\begin{aligned}\lambda^{(\text{start})} &= [1, 1, 0, 100] & L_{\text{start}} &= 20\% \\ \lambda^{(\text{end})} &= [0.98, 2.42, 1, 42] & L_{\text{end}} &= 4\%\end{aligned}$$

# Greedy Ablation Study

Given:

$$\begin{aligned}\lambda^{(\text{start})} &= [1, 1, 0, 100] & L_{\text{start}} &= 20\% \\ \lambda^{(\text{end})} &= [0.98, 2.42, 1, 42] & L_{\text{end}} &= 4\%\end{aligned}$$

1st Iteration:

$$\lambda^{(1)} = [\textcolor{blue}{0.98}, 1, 0, 100] \quad L_1 = 19\%$$

# Greedy Ablation Study

Given:

$$\begin{aligned}\lambda^{(\text{start})} &= [1, 1, 0, 100] & L_{\text{start}} &= 20\% \\ \lambda^{(\text{end})} &= [0.98, 2.42, 1, 42] & L_{\text{end}} &= 4\%\end{aligned}$$

1st Iteration:

$$\begin{aligned}\lambda^{(1)} &= [0.98, 1, 0, 100] & L_1 &= 19\% \\ \lambda^{(2)} &= [1, 2.42, 0, 100] & L_2 &= 20\%\end{aligned}$$

# Greedy Ablation Study

Given:

$$\begin{aligned}\boldsymbol{\lambda}^{(\text{start})} &= [1, 1, 0, 100] & L_{\text{start}} &= 20\% \\ \boldsymbol{\lambda}^{(\text{end})} &= [0.98, 2.42, 1, 42] & L_{\text{end}} &= 4\%\end{aligned}$$

1st Iteration:

$$\begin{aligned}\boldsymbol{\lambda}^{(1)} &= [0.98, 1, 0, 100] & L_1 &= 19\% \\ \boldsymbol{\lambda}^{(2)} &= [1, 2.42, 0, 100] & L_2 &= 20\% \\ \boldsymbol{\lambda}^{(3)} &= [1, 1, 1, 100] & L_3 &= 7\%\end{aligned}$$

# Greedy Ablation Study

Given:

$$\begin{aligned}\lambda^{(\text{start})} &= [1, 1, 0, 100] & L_{\text{start}} &= 20\% \\ \lambda^{(\text{end})} &= [0.98, 2.42, 1, 42] & L_{\text{end}} &= 4\%\end{aligned}$$

1st Iteration:

$$\begin{aligned}\lambda^{(1)} &= [0.98, 1, 0, 100] & L_1 &= 19\% \\ \lambda^{(2)} &= [1, 2.42, 0, 100] & L_2 &= 20\% \\ \lambda^{(3)} &= [1, 1, 1, 100] & L_3 &= 7\% \\ \lambda^{(4)} &= [1, 1, 0, 42] & L_4 &= 16\%\end{aligned}$$

# Greedy Ablation Study

Given:

$$\begin{aligned}\lambda^{(\text{start})} &= [1, 1, 0, 100] & L_{\text{start}} &= 20\% \\ \lambda^{(\text{end})} &= [0.98, 2.42, 1, 42] & L_{\text{end}} &= 4\%\end{aligned}$$

1st Iteration:

$$\begin{aligned}\lambda^{(1)} &= [0.98, 1, 0, 100] & L_1 &= 19\% \\ \lambda^{(2)} &= [1, 2.42, 0, 100] & L_2 &= 20\% \\ \lambda^{(3)} &= [1, 1, 1, 100] & L_3 &= 7\% \\ \lambda^{(4)} &= [1, 1, 0, 42] & L_4 &= 16\%\end{aligned}$$

~~~ 1st step:  $\lambda_2$  – flipping hyperparameter 3

Greedy Ablation Study

Given:

$$\begin{aligned}\boldsymbol{\lambda}^{(\text{start})} &= [1, 1, 0, 100] & L_{\text{start}} &= 20\% \\ \boldsymbol{\lambda}^{(s1)} &= [1, 1, 1, 100] & L &= 7\% \\ \boldsymbol{\lambda}^{(\text{end})} &= [0.98, 2.42, 1, 42] & L_{\text{end}} &= 4\%\end{aligned}$$

2nd Iteration:

$$\boldsymbol{\lambda}^{(1)} = [0.98, 1, 1, 100] \quad L_1 = 6\%$$

Greedy Ablation Study

Given:

$$\begin{aligned}\boldsymbol{\lambda}^{(\text{start})} &= [1, 1, 0, 100] & L_{\text{start}} &= 20\% \\ \boldsymbol{\lambda}^{(s1)} &= [1, 1, 1, 100] & L &= 7\% \\ \boldsymbol{\lambda}^{(\text{end})} &= [0.98, 2.42, 1, 42] & L_{\text{end}} &= 4\%\end{aligned}$$

2nd Iteration:

$$\begin{aligned}\boldsymbol{\lambda}^{(1)} &= [0.98, 1, 1, 100] & L_1 &= 6\% \\ \boldsymbol{\lambda}^{(2)} &= [1, 2.42, 1, 100] & L_2 &= 7\%\end{aligned}$$

Greedy Ablation Study

Given:

$$\begin{aligned}\boldsymbol{\lambda}^{(\text{start})} &= [1, 1, 0, 100] & L_{\text{start}} &= 20\% \\ \boldsymbol{\lambda}^{(s1)} &= [1, 1, 1, 100] & L &= 7\% \\ \boldsymbol{\lambda}^{(\text{end})} &= [0.98, 2.42, 1, 42] & L_{\text{end}} &= 4\%\end{aligned}$$

2nd Iteration:

$$\begin{aligned}\boldsymbol{\lambda}^{(1)} &= [0.98, 1, 1, 100] & L_1 &= 6\% \\ \boldsymbol{\lambda}^{(2)} &= [1, 2.42, 1, 100] & L_2 &= 7\% \\ \boldsymbol{\lambda}^{(3)} &= [1, 1, 1, 42] & L_3 &= 5\%\end{aligned}$$

~~~ 2nd step:  $\lambda_3$  – flipping hyperparameter 4

# Greedy Ablation Study

Given:

$$\begin{aligned}\boldsymbol{\lambda}^{(\text{start})} &= [1, 1, 0, 100] & L_{\text{start}} &= 20\% \\ \boldsymbol{\lambda}^{(s1)} &= [1, 1, 1, 100] & L &= 7\% \\ \boldsymbol{\lambda}^{(s2)} &= [1, 1, 1, 42] & L &= 5\% \\ \boldsymbol{\lambda}^{(\text{end})} &= [0.98, 2.42, 1, 42] & L_{\text{end}} &= 4\%\end{aligned}$$

3rd Iteration:

$$\begin{aligned}\boldsymbol{\lambda}^{(1)} &= [0.98, 1, 1, 100] & L_1 &= 4\% \\ \boldsymbol{\lambda}^{(2)} &= [1, 2.42, 1, 100] & L_2 &= 5\%\end{aligned}$$

~~~ 2nd step:  $\lambda_3$  – flipping hyperparameter 1

Greedy Ablation Study

Ablation Path:

$$\lambda^{(\text{start})} = [1, 1, 0, 100] \quad L_{\text{start}} = 20\%$$

$$\lambda^{(s1)} = [1, 1, 1, 100] \quad L = 7\%$$

$$\lambda^{(s1)} = [1, 1, 1, 42] \quad L = 5\%$$

$$\lambda^{(s3)} = [0.98, 1, 1, 42] \quad L = 4\%$$

$$\lambda^{(s4)} = [0.98, 2.42, 1, 42] \quad L = 4\%$$

$$\lambda^{(\text{end})} = [0.98, 2.42, 1, 42] \quad L_{\text{end}} = 4\%$$

Greedy Ablation Pseudo Code

Algorithm 1 Greedy Ablation

Input : Algorithm \mathcal{A} with configuration space Λ , start configuration $\lambda^{(\text{start})}$, end configuration $\lambda^{(\text{end})}$, cost metric c

$\lambda \leftarrow \lambda^{(\text{start})};$
 $P \leftarrow [];$

Greedy Ablation Pseudo Code

Algorithm 2 Greedy Ablation

Input : Algorithm \mathcal{A} with configuration space Λ , start configuration $\lambda^{(\text{start})}$, end configuration $\lambda^{(\text{end})}$, cost metric c

$\lambda \leftarrow \lambda^{(\text{start})};$

$P \leftarrow [];$

foreach $t \in \{1 \dots |\Lambda|\}$ **do**

Greedy Ablation Pseudo Code

Algorithm 3 Greedy Ablation

Input : Algorithm \mathcal{A} with configuration space Λ , start configuration $\lambda^{(\text{start})}$, end configuration $\lambda^{(\text{end})}$, cost metric c

```
 $\lambda \leftarrow \lambda^{(\text{start})};$ 
 $P \leftarrow [];$ 
foreach  $t \in \{1 \dots |\Lambda|\}$  do
    foreach  $\delta \in \Delta(\lambda, \lambda^{(\text{end})})$  do
         $\lambda'_\delta \leftarrow \text{apply } \delta \text{ to } \lambda;$ 
        evaluate  $c(\lambda'_\delta);$ 
```

Greedy Ablation Pseudo Code

Algorithm 4 Greedy Ablation

Input : Algorithm \mathcal{A} with configuration space Λ , start configuration $\lambda^{(\text{start})}$, end configuration $\lambda^{(\text{end})}$, cost metric c

$\lambda \leftarrow \lambda^{(\text{start})};$

$P \leftarrow [];$

foreach $t \in \{1 \dots |\Lambda|\}$ **do**

foreach $\delta \in \Delta(\lambda, \lambda^{(\text{end})})$ **do**

$\lambda'_\delta \leftarrow \text{apply } \delta \text{ to } \lambda;$

 evaluate $c(\lambda'_\delta);$

Determine most important change $\delta^* \in \arg \min_{\delta \in \Delta(\lambda, \lambda^{(\text{end})})} c(\lambda_\delta);$

$\lambda \leftarrow \text{apply } \delta^* \text{ to } \lambda;$

$P.append(\delta^*);$

Greedy Ablation Pseudo Code

Algorithm 5 Greedy Ablation

Input : Algorithm \mathcal{A} with configuration space Λ , start configuration $\lambda^{(\text{start})}$, end configuration $\lambda^{(\text{end})}$, cost metric c

$\lambda \leftarrow \lambda^{(\text{start})};$

$P \leftarrow [];$

foreach $t \in \{1 \dots |\Lambda|\}$ **do**

foreach $\delta \in \Delta(\lambda, \lambda^{(\text{end})})$ **do**

$\lambda'_\delta \leftarrow \text{apply } \delta \text{ to } \lambda;$

 evaluate $c(\lambda'_\delta);$

Determine most important change $\delta^* \in \arg \min_{\delta \in \Delta(\lambda, \lambda^{(\text{end})})} c(\lambda_\delta);$

$\lambda \leftarrow \text{apply } \delta^* \text{ to } \lambda;$

 P.append(δ^*);

return *Ablation path P*

Remarks on Ablation

- Even this greedy ablation requires $\mathcal{O}(n^2)$ steps

Remarks on Ablation

- Even this greedy ablation requires $\mathcal{O}(n^2)$ steps
- ↝ We can also speedup that up by using surrogate models
[Biedenkapp et al. 2017]

Remarks on Ablation

- Even this greedy ablation requires $\mathcal{O}(n^2)$ steps
- ~ We can also speedup that up by using surrogate models
[Biedenkapp et al. 2017]
- Common observations:
 - ➊ Some hyperparameters might not matter (λ_2 in the example)

Remarks on Ablation

- Even this greedy ablation requires $\mathcal{O}(n^2)$ steps
- ~ We can also speedup that up by using surrogate models
[Biedenkapp et al. 2017]
- Common observations:
 - ① Some hyperparameters might not matter (λ_2 in the example)
 - ② Often only a few of the hyperparameters have an big impact

Remarks on Ablation

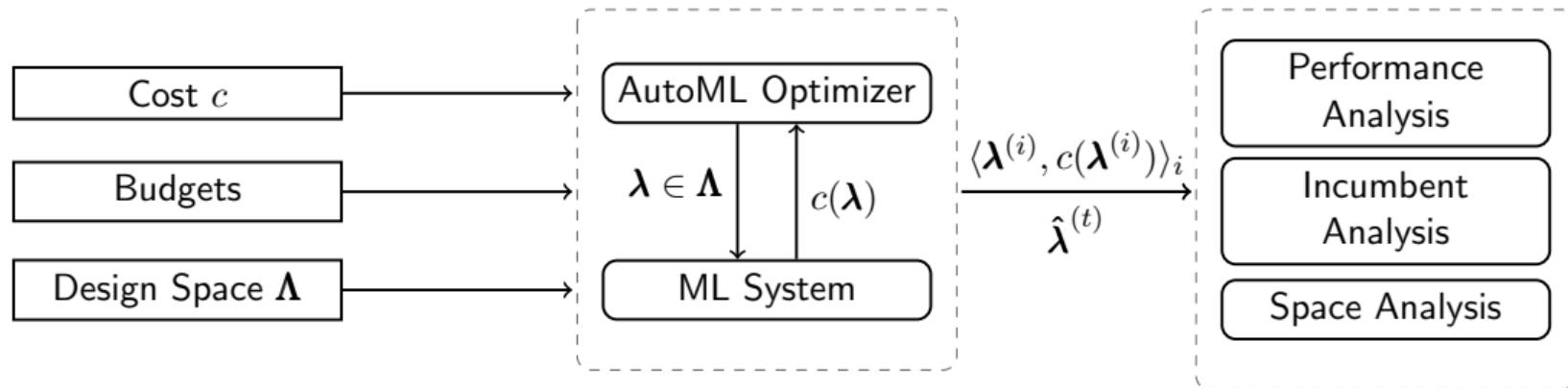
- Even this greedy ablation requires $\mathcal{O}(n^2)$ steps
- ~ We can also speedup that up by using surrogate models
[Biedenkapp et al. 2017]
- Common observations:
 - 1 Some hyperparameters might not matter (λ_2 in the example)
 - 2 Often only a few of the hyperparameters have an big impact
 - 3 You have plateaus in your ablation path because of interaction effects

AutoML: Interpretability

Global Hyperparameter Importance

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Idea



→ focus on which hyperparameters are important across the entire search space

Importance Analysis of Surrogate Model

- Key Idea: Surrogate models ($\Lambda \rightarrow \mathbb{R}$) learn from observations how to predict the performance of a hyperparameter configuration $\lambda \in \Lambda$

Importance Analysis of Surrogate Model

- Key Idea: Surrogate models ($\Lambda \rightarrow \mathbb{R}$) learn from observations how to predict the performance of a hyperparameter configuration $\lambda \in \Lambda$
- ~~ These models can be used to figure out which hyperparameter was important

Importance Analysis of Surrogate Model

- Key Idea: Surrogate models ($\Lambda \rightarrow \mathbb{R}$) learn from observations how to predict the performance of a hyperparameter configuration $\lambda \in \Lambda$
 - ~ These models can be used to figure out which hyperparameter was important
- For example:
 - ▶ Use forward selection [Hutter et al. 2013]
 - ▶ Use automatic feature relevance determination of the model (e.g., of a surrogate model based on random forest)

Importance Analysis of Surrogate Model

- Key Idea: Surrogate models ($\Lambda \rightarrow \mathbb{R}$) learn from observations how to predict the performance of a hyperparameter configuration $\lambda \in \Lambda$
 - ~ These models can be used to figure out which hyperparameter was important
- For example:
 - ▶ Use forward selection [Hutter et al. 2013]
 - ▶ Use automatic feature relevance determination of the model (e.g., of a surrogate model based on random forest)
- Advantages:
 - ▶ Very cheap to do, since we only have to query the surrogate model several times

Importance Analysis of Surrogate Model

- Key Idea: Surrogate models ($\Lambda \rightarrow \mathbb{R}$) learn from observations how to predict the performance of a hyperparameter configuration $\lambda \in \Lambda$
 - ~ These models can be used to figure out which hyperparameter was important
- For example:
 - ▶ Use forward selection [Hutter et al. 2013]
 - ▶ Use automatic feature relevance determination of the model (e.g., of a surrogate model based on random forest)
- Advantages:
 - ▶ Very cheap to do, since we only have to query the surrogate model several times
- Potential drawback:
 - ▶ The surrogate model might overfit to different subsets of the hyperparameters (if we don't provide sufficient data)

Global Importance Analysis

- Key idea: What is the importance of a hyperparameter by marginalizing over all other hyperparameter effects?

Global Importance Analysis

- Key idea: What is the importance of a hyperparameter by marginalizing over all other hyperparameter effects?
- Key Insight: We can use a surrogate model to compute these effects

Global Importance Analysis

- Key idea: What is the importance of a hyperparameter by marginalizing over all other hyperparameter effects?
- Key Insight: We can use a surrogate model to compute these effects

fANOVA [Sobobl. 1993]

Write performance predictions as a sum of components:

$$\hat{y}(\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_n) = \hat{f}_0 + \sum_{i=1}^n \hat{f}_i(\boldsymbol{\lambda}_i) + \sum_{i \neq j} \hat{f}_{ij}(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_j) + \dots$$

$$\begin{aligned}\hat{y}(\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_n) = & \quad \text{average response} + \text{main effects} + \\ & \quad \text{2-D interaction effects} + \text{higher order effects}\end{aligned}$$

Global Importance Analysis

- Key idea: What is the importance of a hyperparameter by marginalizing over all other hyperparameter effects?
- Key Insight: We can use a surrogate model to compute these effects

fANOVA [Sobobl. 1993]

Write performance predictions as a sum of components:

$$\begin{aligned}\hat{y}(\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_n) &= \hat{f}_0 + \sum_{i=1}^n \hat{f}_i(\boldsymbol{\lambda}_i) + \sum_{i \neq j} \hat{f}_{ij}(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_j) + \dots \\ \hat{y}(\boldsymbol{\lambda}_1, \dots, \boldsymbol{\lambda}_n) &= \text{average response} + \text{main effects} + \\ &\quad \text{2-D interaction effects} + \text{higher order effects}\end{aligned}$$

Variance Decomposition

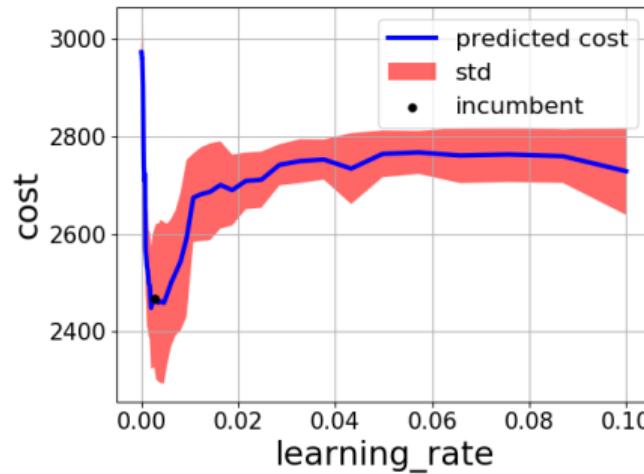
$$V = \frac{1}{||\boldsymbol{\Lambda}||} \int_{\boldsymbol{\lambda}_1} \dots \int_{\boldsymbol{\lambda}_n} [(\hat{y}(\boldsymbol{\lambda}) - \hat{f}_0)^2] d\boldsymbol{\lambda}_1 \dots d\boldsymbol{\lambda}_n$$

fANOVA Analysis

- The fANOVA and variance decomposition can be done efficiently in linear time if the surrogate model is a random forest [Hutter et al. 2014]

fANOVA Analysis

- The fANOVA and variance decomposition can be done efficiently in linear time if the surrogate model is a random forest [Hutter et al. 2014]

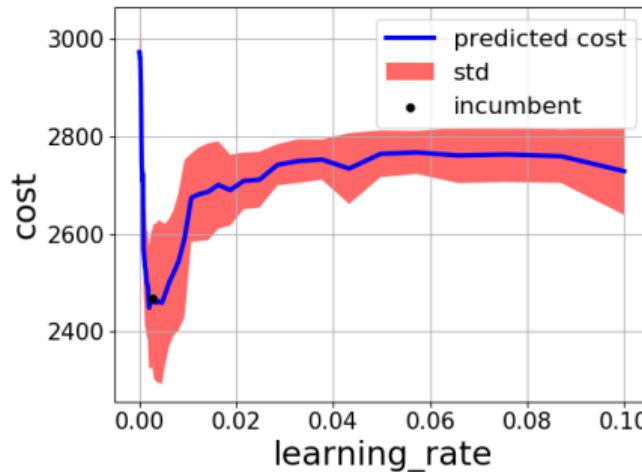


- predicted cost is marginalized over all other hyperparameter effects

Source: [Lindauer et al. 2019]

fANOVA Analysis

- The fANOVA and variance decomposition can be done efficiently in linear time if the surrogate model is a random forest [Hutter et al. 2014]



- predicted cost is marginalized over all other hyperparameter effects
- Warning:** The optimum on these curves does not have to be the global optimum across all hyperparameters

Source: [Lindauer et al. 2019]

fANOVA Analysis

- How much of the variance can be explained by a hyperparameter (or combinations of hyperparamaters) marginalized over all other parameters?

Table: Exemplary analysis of PPO on cartpole

| Hyperparameter | Explained Variance |
|----------------------------|--------------------|
| Discount rate | 19.3 % |
| Batch size | 15.7 % |
| Learning rate | 3.7 % |
| Likelihood ration clipping | 3.4% |
| ... | |

fANOVA Analysis

- How much of the variance can be explained by a hyperparameter (or combinations of hyperparameters) marginalized over all other parameters?

Table: Exemplary analysis of PPO on cartpole

| Hyperparameter | Explained Variance |
|--|--------------------|
| Discount rate | 19.3 % |
| Batch size | 15.7 % |
| Learning rate | 3.7 % |
| Likelihood ration clipping | 3.4% |
| ... | |
| discount rate & batch size | 10.4% |
| discount rate & likelihood ration clipping | 4.4% |
| ... | |

Remarks on fANOVA

- Given compute higher-order interaction effects
 - ▶ Often too expensive for more than 2 or 3 dimensions

Remarks on fANOVA

- Given compute higher-order interaction effects
 - ▶ Often too expensive for more than 2 or 3 dimensions
- Implicit assumption: the surrogate model models the space fairly well

Remarks on fANOVA

- Given compute higher-order interaction effects
 - ▶ Often too expensive for more than 2 or 3 dimensions
- Implicit assumption: the surrogate model models the space fairly well
- Global analysis and local analysis of hyperparameter importance does not always agree
[Biedenkapp et al. 2018]

Remarks on fANOVA

- Given compute higher-order interaction effects
 - ▶ Often too expensive for more than 2 or 3 dimensions
- Implicit assumption: the surrogate model models the space fairly well
- Global analysis and local analysis of hyperparameter importance does not always agree
[Biedenkapp et al. 2018]
- ~~> You should run both to get a good understanding of why an AutoML tool chose a configuration

AutoML: Beyond AutoML

Overview: Algorithm Configuration

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Generalization of HPO

- hyperparameter optimization (HPO) is not limited to ML

Generalization of HPO

- hyperparameter optimization (HPO) is not limited to ML
- in fact, you can optimize the performance of any algorithm by means of HPO if

Generalization of HPO

- hyperparameter optimization (HPO) is not limited to ML
- in fact, you can optimize the performance of any algorithm by means of HPO if
 - ① the algorithm at hand has parameters that influence its performance

Generalization of HPO

- hyperparameter optimization (HPO) is not limited to ML
- in fact, you can optimize the performance of any algorithm by means of HPO if
 - ① the algorithm at hand has parameters that influence its performance
 - ② you care about the empirical performance of an algorithm

Generalization of HPO

- hyperparameter optimization (HPO) is not limited to ML
- in fact, you can optimize the performance of any algorithm by means of HPO if
 - ① the algorithm at hand has parameters that influence its performance
 - ② you care about the empirical performance of an algorithm
- a limitation of HPO is that we assume that we care only about a single task (i.e., dataset or input to the algorithm)

Generalization of HPO

- hyperparameter optimization (HPO) is not limited to ML
 - in fact, you can optimize the performance of any algorithm by means of HPO if
 - ① the algorithm at hand has parameters that influence its performance
 - ② you care about the empirical performance of an algorithm
 - a limitation of HPO is that we assume that we care only about a single task (i.e., dataset or input to the algorithm)
- ↝ Can we find an algorithm's configuration that performs well and robustly across a set of tasks?

Generalization of HPO

- hyperparameter optimization (HPO) is not limited to ML
 - in fact, you can optimize the performance of any algorithm by means of HPO if
 - ① the algorithm at hand has parameters that influence its performance
 - ② you care about the empirical performance of an algorithm
 - a limitation of HPO is that we assume that we care only about a single task (i.e., dataset or input to the algorithm)
- ↝ Can we find an algorithm's configuration that performs well and robustly across a set of tasks?
- ▶ A hyperparameter configuration for a set of datasets

Generalization of HPO

- hyperparameter optimization (HPO) is not limited to ML
 - in fact, you can optimize the performance of any algorithm by means of HPO if
 - ① the algorithm at hand has parameters that influence its performance
 - ② you care about the empirical performance of an algorithm
 - a limitation of HPO is that we assume that we care only about a single task (i.e., dataset or input to the algorithm)
- ↝ Can we find an algorithm's configuration that performs well and robustly across a set of tasks?
- ▶ A hyperparameter configuration for a set of datasets
 - ▶ A parameter configuration of a SAT solver for a set of SAT instances

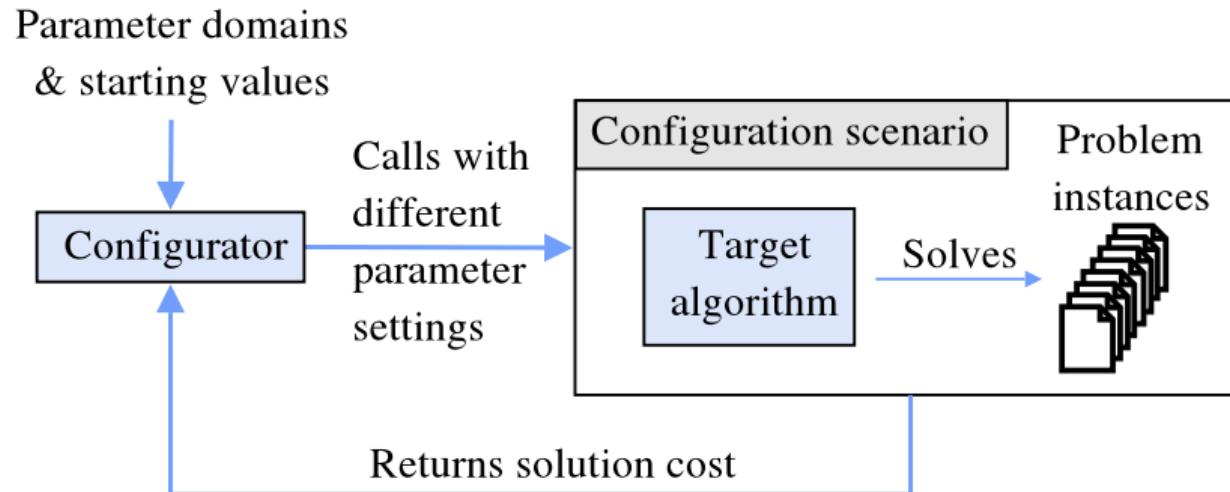
Generalization of HPO

- hyperparameter optimization (HPO) is not limited to ML
 - in fact, you can optimize the performance of any algorithm by means of HPO if
 - ① the algorithm at hand has parameters that influence its performance
 - ② you care about the empirical performance of an algorithm
 - a limitation of HPO is that we assume that we care only about a single task (i.e., dataset or input to the algorithm)
- ↝ Can we find an algorithm's configuration that performs well and robustly across a set of tasks?
- ▶ A hyperparameter configuration for a set of datasets
 - ▶ A parameter configuration of a SAT solver for a set of SAT instances
 - ▶ A parameter configuration of an AI planning solver for a set of planning problems
 - ▶ ...

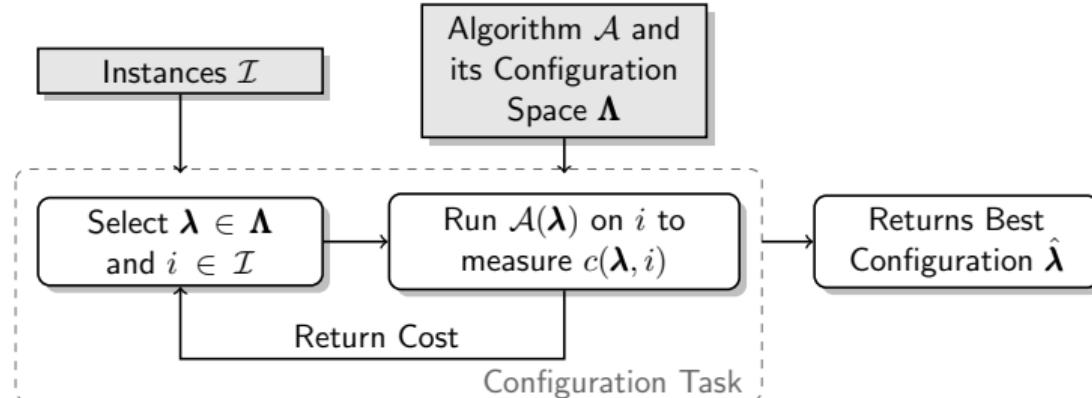
Generalization of HPO

- hyperparameter optimization (HPO) is not limited to ML
- in fact, you can optimize the performance of any algorithm by means of HPO if
 - ① the algorithm at hand has parameters that influence its performance
 - ② you care about the empirical performance of an algorithm
- a limitation of HPO is that we assume that we care only about a single task (i.e., dataset or input to the algorithm)
 - ~ Can we find an algorithm's configuration that performs well and robustly across a set of tasks?
 - ▶ A hyperparameter configuration for a set of datasets
 - ▶ A parameter configuration of a SAT solver for a set of SAT instances
 - ▶ A parameter configuration of an AI planning solver for a set of planning problems
 - ▶ ...
 - ~ Algorithm configuration

Algorithm Configuration Visualized



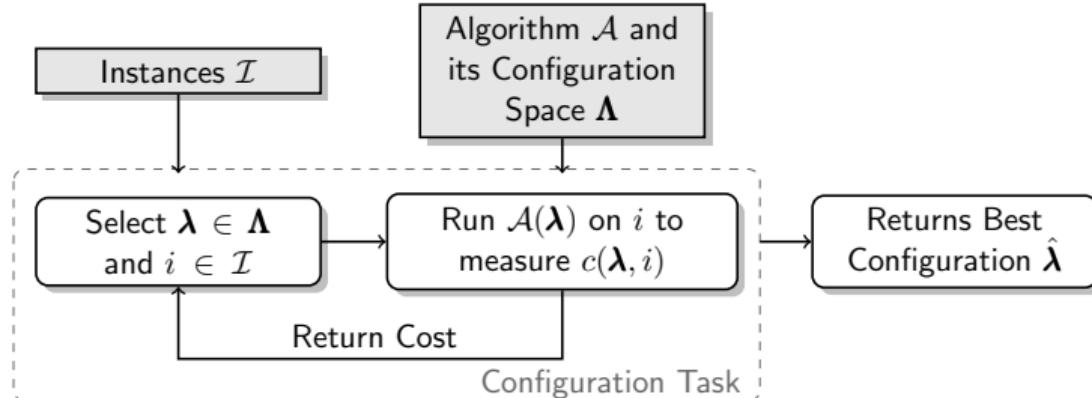
Algorithm Configuration – in More Detail



Definition

Given a parameterized algorithm \mathcal{A} with possible (hyper-)parameter settings Λ ,

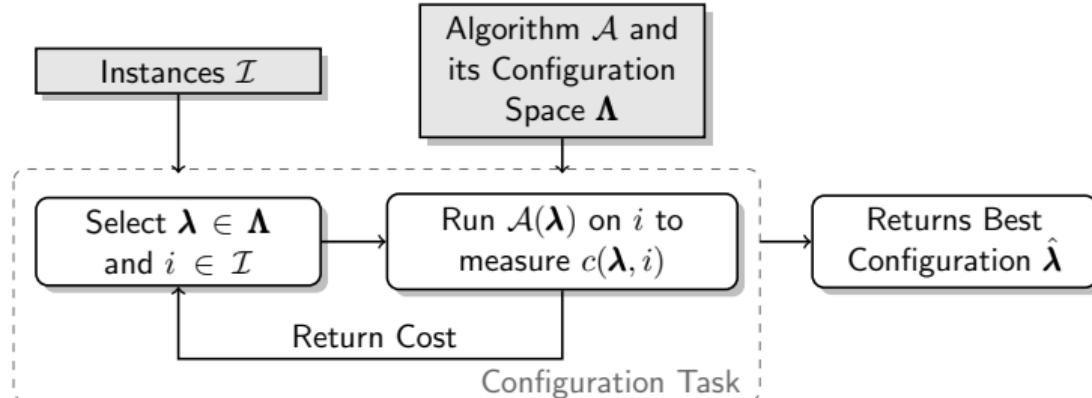
Algorithm Configuration – in More Detail



Definition

Given a parameterized algorithm \mathcal{A} with possible (hyper-)parameter settings Λ , a set of training problem instances \mathcal{I} ,

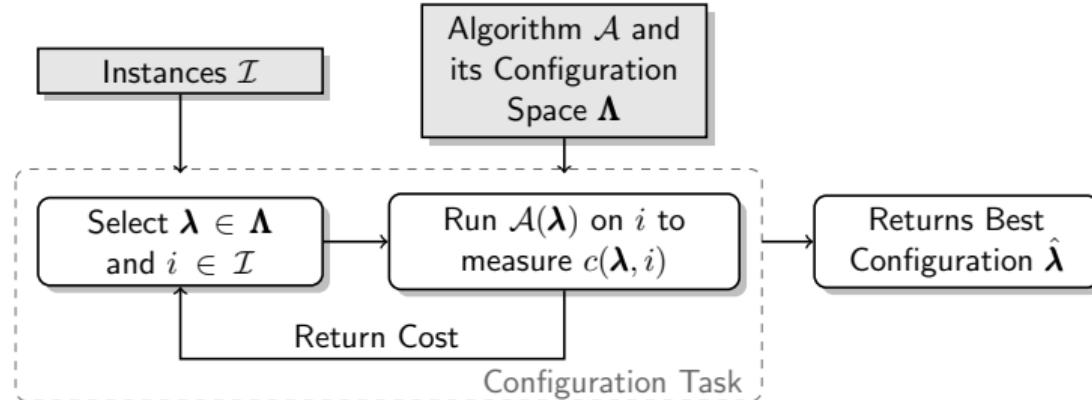
Algorithm Configuration – in More Detail



Definition

Given a parameterized algorithm \mathcal{A} with possible (hyper-)parameter settings Λ , a set of training problem instances \mathcal{I} , and a cost metric $c : \Lambda \times \mathcal{I} \rightarrow \mathbb{R}$,

Algorithm Configuration – in More Detail



Definition

Given a parameterized algorithm \mathcal{A} with possible (hyper-)parameter settings Λ , a set of training problem instances \mathcal{I} , and a cost metric $c : \Lambda \times \mathcal{I} \rightarrow \mathbb{R}$, the algorithm configuration problem is to **find a parameter configuration $\lambda^* \in \Lambda$ that minimizes c across the instances in \mathcal{I}** .

Algorithm Configuration – Full Formal Definition

Definition

An instance of the algorithm configuration problem is a 5-tuple $(\mathcal{A}, \Lambda, \mathcal{D}, \kappa, c)$ where:

- \mathcal{A} is a parameterized algorithm;
- Λ is the (hyper-)parameter configuration space of \mathcal{A} ;
- \mathcal{D} is a distribution over problem instances with domain \mathcal{I} ;

Algorithm Configuration – Full Formal Definition

Definition

An instance of the algorithm configuration problem is a 5-tuple $(\mathcal{A}, \Lambda, \mathcal{D}, \kappa, c)$ where:

- \mathcal{A} is a parameterized algorithm;
- Λ is the (hyper-)parameter configuration space of \mathcal{A} ;
- \mathcal{D} is a distribution over problem instances with domain \mathcal{I} ;
- $\kappa < \infty$ is a cutoff time, after which each run of \mathcal{A} will be terminated if still running

Algorithm Configuration – Full Formal Definition

Definition

An instance of the algorithm configuration problem is a 5-tuple $(\mathcal{A}, \boldsymbol{\Lambda}, \mathcal{D}, \kappa, c)$ where:

- \mathcal{A} is a parameterized algorithm;
- $\boldsymbol{\Lambda}$ is the (hyper-)parameter configuration space of \mathcal{A} ;
- \mathcal{D} is a distribution over problem instances with domain \mathcal{I} ;
- $\kappa < \infty$ is a cutoff time, after which each run of \mathcal{A} will be terminated if still running
- $c : \boldsymbol{\Lambda} \times \mathcal{I} \rightarrow \mathbb{R}$ is a function that measures the observed cost of running $\mathcal{A}(\boldsymbol{\lambda})$ on an instance $i \in \mathcal{I}$ with cutoff time κ

Algorithm Configuration – Full Formal Definition

Definition

An instance of the algorithm configuration problem is a 5-tuple $(\mathcal{A}, \Lambda, \mathcal{D}, \kappa, c)$ where:

- \mathcal{A} is a parameterized algorithm;
- Λ is the (hyper-)parameter configuration space of \mathcal{A} ;
- \mathcal{D} is a distribution over problem instances with domain \mathcal{I} ;
- $\kappa < \infty$ is a cutoff time, after which each run of \mathcal{A} will be terminated if still running
- $c : \Lambda \times \mathcal{I} \rightarrow \mathbb{R}$ is a function that measures the observed cost of running $\mathcal{A}(\lambda)$ on an instance $i \in \mathcal{I}$ with cutoff time κ

The cost of a candidate solution $\lambda \in \Lambda$ is $f(\lambda) = \mathbb{E}_{i \sim \mathcal{D}}(c(\lambda, i))$.

The goal is to find $\lambda^* \in \arg \min_{\lambda \in \Lambda} f(\lambda)$.

Distribution of Instances

We usually have a finite number of instances from a given application

- We want to do well on that type of instances
- Future instances of this type should be solved well

Distribution of Instances

We usually have a finite number of instances from a given application

- We want to do well on that type of instances
- Future instances of this type should be solved well

Like in machine learning

- We split the instances into a **training set** and a **test set**
- We configure algorithms on the training instances
- We only use the test instances afterwards
 - unbiased estimate of generalization performance for unseen instances

Challenges of Algorithm Configuration

- Structured high-dimensional parameter space
 - ▶ categorical vs. continuous parameters
 - ▶ conditionals between parameters

Challenges of Algorithm Configuration

- Structured high-dimensional parameter space
 - ▶ categorical vs. continuous parameters
 - ▶ conditionals between parameters
- Stochastic optimization
 - ▶ Randomized algorithms: optimization across various seeds
 - ▶ Distribution of benchmark instances (often wide range of hardness)
 - ▶ Subsumes so-called *multi-armed bandit problem*

Challenges of Algorithm Configuration

- Structured high-dimensional parameter space
 - ▶ categorical vs. continuous parameters
 - ▶ conditionals between parameters
- Stochastic optimization
 - ▶ Randomized algorithms: optimization across various seeds
 - ▶ Distribution of benchmark instances (often wide range of hardness)
 - ▶ Subsumes so-called *multi-armed bandit problem*
- Generalization across instances
 - ▶ apply algorithm configuration to **homogeneous** instance sets
 - ▶ Instance sets can also be **heterogeneous**,
i.e., no single configuration performs well on all instances
 - ~~ combination of algorithm configuration and selection

Challenges of Algorithm Configuration

- Structured high-dimensional parameter space
 - ▶ categorical vs. continuous parameters
 - ▶ conditionals between parameters
 - Stochastic optimization
 - ▶ Randomized algorithms: optimization across various seeds
 - ▶ Distribution of benchmark instances (often wide range of hardness)
 - ▶ Subsumes so-called *multi-armed bandit problem*
 - Generalization across instances
 - ▶ apply algorithm configuration to **homogeneous** instance sets
 - ▶ Instance sets can also be **heterogeneous**,
i.e., no single configuration performs well on all instances
 - ~~ combination of algorithm configuration and selection
- ~~ Hyperparameter optimization is a subproblem of algorithm configuration
[Eggensperger et al. 2019]

AutoML: Beyond AutoML

Racing for Algorithm Configuration

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

State-of-the-art Algorithm Configuration

SMAC: Sequential Model-based Algorithm Configuration [Hutter et al. 2011]

- Bayesian Optimization +
- aggressive racing +
- adaptive capping (for optimizing runtime)

Algorithm 1 SMAC

Input : instance set \mathcal{I} , Algorithm \mathcal{A} with configuration space Λ , Initial configuration λ_0 , performance metric c , Configuration budget b

run history $\mathcal{D}_{\text{Hist}} \leftarrow$ initial design based on λ_0 ;

// $\mathcal{D}_{\text{Hist}} = (\lambda, i, c(i, \lambda))_i$

while b remains **do**

|

Algorithm 2 SMAC

Input : instance set \mathcal{I} , Algorithm \mathcal{A} with configuration space Λ , Initial configuration λ_0 , performance metric c , Configuration budget b

run history $\mathcal{D}_{\text{Hist}} \leftarrow$ initial design based on λ_0 ;

// $\mathcal{D}_{\text{Hist}} = (\lambda, i, c(i, \lambda))_i$

while b remains **do**

$\hat{c} \leftarrow$ train empirical performance model based on run history $\mathcal{D}_{\text{Hist}}$;

Algorithm 3 SMAC

Input : instance set \mathcal{I} , Algorithm \mathcal{A} with configuration space Λ , Initial configuration λ_0 , performance metric c , Configuration budget b

run history $\mathcal{D}_{\text{Hist}} \leftarrow$ initial design based on λ_0 ;

// $\mathcal{D}_{\text{Hist}} = (\lambda, i, c(i, \lambda))_i$

while b remains **do**

$\hat{c} \leftarrow$ train empirical performance model based on run history $\mathcal{D}_{\text{Hist}}$;

$\Lambda_{\text{challengers}} \leftarrow$ select configurations based on \hat{c} ;

Algorithm 4 SMAC

Input : instance set \mathcal{I} , Algorithm \mathcal{A} with configuration space Λ , Initial configuration λ_0 , performance metric c , Configuration budget b

run history $\mathcal{D}_{\text{Hist}} \leftarrow$ initial design based on λ_0 ;

// $\mathcal{D}_{\text{Hist}} = (\lambda, i, c(i, \lambda))_i$

while b remains **do**

$\hat{c} \leftarrow$ train empirical performance model based on run history $\mathcal{D}_{\text{Hist}}$;

$\Lambda_{\text{challengers}} \leftarrow$ select configurations based on \hat{c} ;

$\hat{\lambda}, \mathcal{D}_{\text{Hist}} \leftarrow$ intensify($\Lambda_{\text{challengers}}$, $\hat{\lambda}$);

// racing and capping

Algorithm 5 SMAC

Input : instance set \mathcal{I} , Algorithm \mathcal{A} with configuration space Λ , Initial configuration λ_0 , performance metric c , Configuration budget b

run history $\mathcal{D}_{\text{Hist}} \leftarrow$ initial design based on λ_0 ;

// $\mathcal{D}_{\text{Hist}} = (\lambda, i, c(i, \lambda))_i$

while b remains **do**

$\hat{c} \leftarrow$ train empirical performance model based on run history $\mathcal{D}_{\text{Hist}}$;

$\Lambda_{\text{challengers}} \leftarrow$ select configurations based on \hat{c} ;

$\hat{\lambda}, \mathcal{D}_{\text{Hist}} \leftarrow$ intensify($\Lambda_{\text{challengers}}$, $\hat{\lambda}$); // racing and capping

return $\hat{\lambda}$

Comparisons on N instances [Hutter et al. 2009]

- **Basic(N)** uses a pretty basic comparison: $\text{better}_N(\lambda', \lambda)$:
 - ▶ Compare λ' and λ based on N instances

Comparisons on N instances [Hutter et al. 2009]

- **Basic(N)** uses a pretty basic comparison: $\text{better}_N(\lambda', \lambda)$:
 - ▶ Compare λ' and λ based on N instances
 - ▶ How does this relate to cross-validation?

Comparisons on N instances [Hutter et al. 2009]

- Basic(N) uses a pretty basic comparison: $\text{better}_N(\lambda', \lambda)$:
 - ▶ Compare λ' and λ based on N instances
 - ▶ How does this relate to cross-validation?
- Problem: How to set N ? Problems of large N ? Small N ?

Comparisons on N instances [Hutter et al. 2009]

- Basic(N) uses a pretty basic comparison: $\text{better}_N(\lambda', \lambda)$:
 - ▶ Compare λ' and λ based on N instances
 - ▶ How does this relate to cross-validation?
- Problem: How to set N ? Problems of large N ? Small N ?
 - ▶ Problem of large N : evaluations are slow
 - ▶ Problem of small N : overfitting to a small set of instances
 - ~~ Tradeoff: Choose N of moderate size

Comparisons on N instances [Hutter et al. 2009]

Question: Which N instances should we use?

- ① N different instances for each configuration
- ② The same set of N instances for the entire run

Comparisons on N instances [Hutter et al. 2009]

Question: Which N instances should we use?

- ① N different instances for each configuration
- ② The same set of N instances for the entire run

Answer: the same N instances, so that we compare apples with apples
(but: using the same instances can also yield overfitting)

Comparisons on N instances [Hutter et al. 2009]

Question: Which N instances should we use?

- ① N different instances for each configuration
- ② The same set of N instances for the entire run

Answer: the same N instances, so that we compare apples with apples
(but: using the same instances can also yield overfitting)

If we sampled different instances for each configuration:

- Some configurations would randomly get easier instances
- Those configurations would look better than they really are

Comparisons on N instances [Hutter et al. 2009]

Question: For randomized algorithms, how should we set the seeds?

- ① Sample a new seed for each algorithm run
- ② Fix the seeds together with the instances

Comparisons on N instances [Hutter et al. 2009]

Question: For randomized algorithms, how should we set the seeds?

- ① Sample a new seed for each algorithm run
- ② Fix the seeds together with the instances

Answer: just like for instances, fix them to compare apples to apples

Comparisons on N instances [Hutter et al. 2009]

Question: For randomized algorithms, how should we set the seeds?

- ① Sample a new seed for each algorithm run
- ② Fix the seeds together with the instances

Answer: just like for instances, fix them to compare apples to apples

In summary, for each run of Basic(N):

pick N (instance, seed) pairs and use them for evaluating each λ .

Comparisons on N instances [Hutter et al. 2009]

Question: For randomized algorithms, how should we set the seeds?

- ① Sample a new seed for each algorithm run
- ② Fix the seeds together with the instances

Answer: just like for instances, fix them to compare apples to apples

In summary, for each run of Basic(N):

pick N (instance, seed) pairs and use them for evaluating each λ .
(Different Basic(N) runs can use different instances and seeds.)

The concept of overfitting

Very related to overfitting in machine learning

- Performance improves on the training set
- Performance does not improve on the test set, and may even degrade

The concept of overtuning

Very related to overfitting in machine learning

- Performance improves on the training set
- Performance does not improve on the test set, and may even degrade

More pronounced for heterogeneous benchmark sets

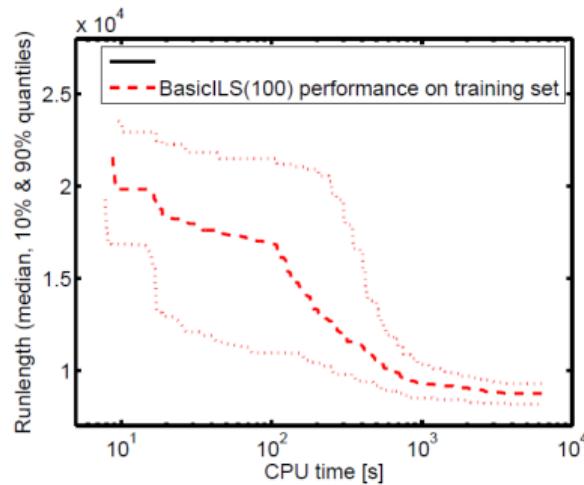
- But it even happens for very homogeneous sets
- Indeed, one can even overfit on a single instance, to the **seeds** used for training

Overtuning Visualized

- Example: minimizing SLS solver runlengths for a single SAT instance
- Training cost, e.g., with $N=100$:
average runlengths across 100 runs with different seeds
- Test cost of $\hat{\lambda}$ here based on 1000 new seeds

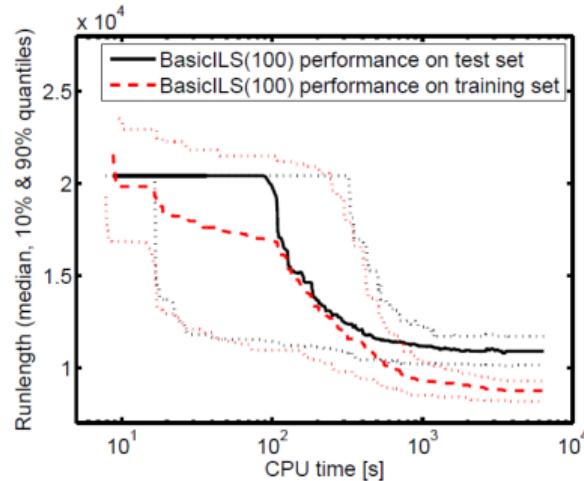
Overtuning Visualized

- Example: minimizing SLS solver runlengths for a single SAT instance
- **Training cost**, e.g., with $N=100$:
average runlengths across 100 runs with different seeds
- **Test cost** of $\hat{\lambda}$ here based on 1000 new seeds



Overtuning Visualized

- Example: minimizing SLS solver runlengths for a single SAT instance
- **Training cost**, e.g., with $N=100$:
average runlengths across 100 runs with different seeds
- **Test cost** of $\hat{\lambda}$ here based on 1000 new seeds

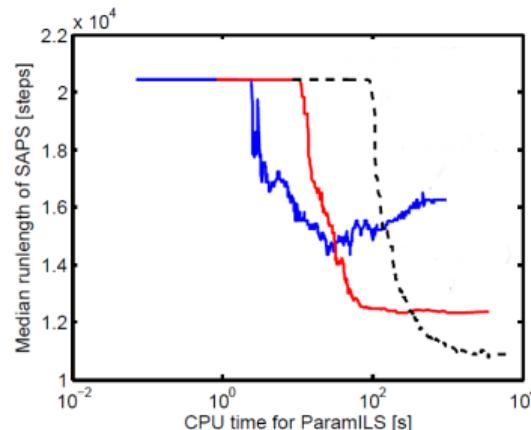


Basic(N) Test Results with Various N

- Example: minimizing SLS solver runlengths for a single SAT instance
- Training cost, e.g., with $N=?$:
average runlengths across N runs with different seeds
- Test cost of $\hat{\lambda}$ here based on 1000 new seeds

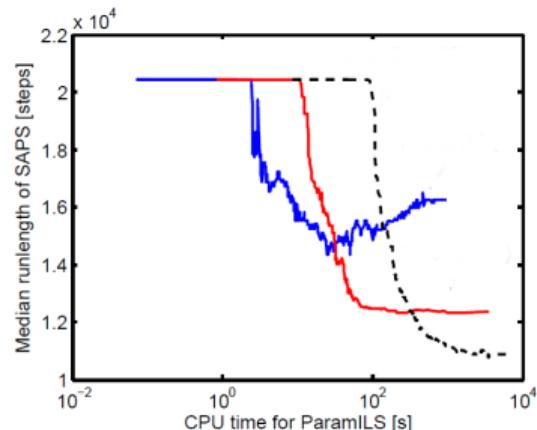
Basic(N) Test Results with Various N

- Example: minimizing SLS solver runlengths for a single SAT instance
- Training cost, e.g., with $N=?$:
average runlengths across N runs with different seeds
- Test cost of $\hat{\lambda}$ here based on 1000 new seeds



Basic(N) Test Results with Various N

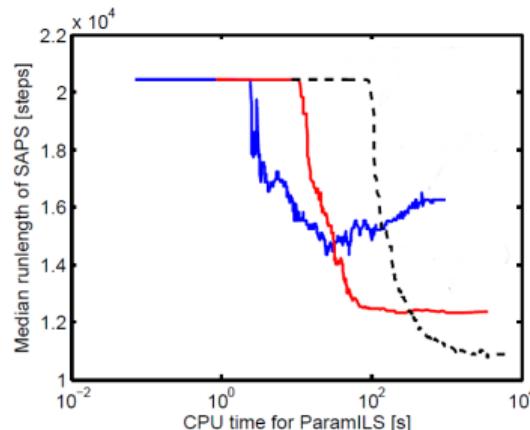
- Example: minimizing SLS solver runlengths for a single SAT instance
- **Training cost**, e.g., with $N=?$:
average runlengths across N runs with different seeds
- **Test cost** of $\hat{\lambda}$ here based on 1000 new seeds



Which of these results corresponds to $N = 1$,
 $N = 10$, and $N = 100$?

Basic(N) Test Results with Various N

- Example: minimizing SLS solver runlengths for a single SAT instance
- **Training cost**, e.g., with $N=?$:
average runlengths across N runs with different seeds
- **Test cost** of $\hat{\lambda}$ here based on 1000 new seeds

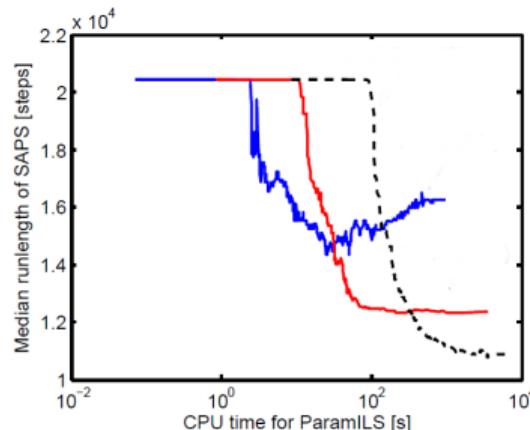


Which of these results corresponds to $N = 1$, $N = 10$, and $N = 100$?

- ➊ $N=1$: blue, $N=10$: red,
 $N=100$ dashed black
- ➋ $N=1$: dashed black,
 $N=10$: red, $N=100$ blue

Basic(N) Test Results with Various N

- Example: minimizing SLS solver runlengths for a single SAT instance
- **Training cost**, e.g., with $N=?$:
average runlengths across N runs with different seeds
- **Test cost** of $\hat{\lambda}$ here based on 1000 new seeds



Which of these results corresponds to $N = 1$, $N = 10$, and $N = 100$?

- ① N=1: blue, N=10: red,
N=100 dashed black
- ② N=1: dashed black,
N=10: red, N=100 blue

Correct Answer: 1

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Intuition: get the best of both worlds

- Perform more runs for good configurations
 - to avoid overtuning
- Quickly reject poor configurations
 - to make progress more quickly

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Intuition: get the best of both worlds

- Perform more runs for good configurations
 - to avoid overtuning
- Quickly reject poor configurations
 - to make progress more quickly

Definition: $N(\lambda)$ and $c_N(\lambda)$

$N(\lambda)$ denotes the number of runs executed for λ so far.

$\hat{c}_N(\lambda)$ denotes the cost estimate of λ based on N runs.

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Intuition: get the best of both worlds

- Perform more runs for good configurations
 - to avoid overtuning
- Quickly reject poor configurations
 - to make progress more quickly

Definition: $N(\lambda)$ and $c_N(\lambda)$

$N(\lambda)$ denotes the number of runs executed for λ so far.

$\hat{c}_N(\lambda)$ denotes the cost estimate of λ based on N runs.

In the beginning: $N(\lambda) = 0$ for every configuration λ

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Definition: domination

$\lambda^{(1)}$ dominates $\lambda^{(2)}$ if

- $N(\lambda^{(1)}) \geq N(\lambda^{(2)})$ and
- $\hat{c}_{N(\lambda^{(2)})}(\lambda^{(1)}) \leq \hat{c}_{N(\lambda^{(2)})}(\lambda^{(2)})$.

I.e.: we have at least as many runs for $\lambda^{(1)}$ and its cost is at least as low.

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Definition: domination

$\lambda^{(1)}$ dominates $\lambda^{(2)}$ if

- $N(\lambda^{(1)}) \geq N(\lambda^{(2)})$ and
- $\hat{c}_{N(\lambda^{(2)})}(\lambda^{(1)}) \leq \hat{c}_{N(\lambda^{(2)})}(\lambda^{(2)})$.

I.e.: we have at least as many runs for $\lambda^{(1)}$ and its cost is at least as low.

better(λ' , $\hat{\lambda}$) in a nutshell

- $\hat{\lambda}$ is the current configuration to beat (incumbent)

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Definition: domination

$\lambda^{(1)}$ dominates $\lambda^{(2)}$ if

- $N(\lambda^{(1)}) \geq N(\lambda^{(2)})$ and
- $\hat{c}_{N(\lambda^{(2)})}(\lambda^{(1)}) \leq \hat{c}_{N(\lambda^{(2)})}(\lambda^{(2)})$.

I.e.: we have at least as many runs for $\lambda^{(1)}$ and its cost is at least as low.

better(λ' , $\hat{\lambda}$) in a nutshell

- $\hat{\lambda}$ is the current configuration to beat (incumbent)
- Perform runs of λ' until either
 - ▶ $\hat{\lambda}$ dominates $\lambda' \rightsquigarrow$ reject λ' , or
 - ▶ λ' dominates $\hat{\lambda} \rightsquigarrow$ change current incumbent ($\hat{\lambda} \leftarrow \lambda'$)

Aggressive Racing (inspired by FocusedILS) [Hutter et al. 2009]

Definition: domination

$\lambda^{(1)}$ dominates $\lambda^{(2)}$ if

- $N(\lambda^{(1)}) \geq N(\lambda^{(2)})$ and
- $\hat{c}_{N(\lambda^{(2)})}(\lambda^{(1)}) \leq \hat{c}_{N(\lambda^{(2)})}(\lambda^{(2)})$.

I.e.: we have at least as many runs for $\lambda^{(1)}$ and its cost is at least as low.

better(λ' , $\hat{\lambda}$) in a nutshell

- $\hat{\lambda}$ is the current configuration to beat (incumbent)
- Perform runs of λ' until either
 - ▶ $\hat{\lambda}$ dominates $\lambda' \rightsquigarrow$ reject λ' , or
 - ▶ λ' dominates $\hat{\lambda} \rightsquigarrow$ change current incumbent ($\hat{\lambda} \leftarrow \lambda'$)
- Over time: perform extra runs of $\hat{\lambda}$ to gain more confidence in it

Toy Example

- Let $\hat{\lambda}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers λ' and λ''

| | | | | | |
|-----------------|-----------|--|-----------|--|-----------|
| | $i^{(1)}$ | | $i^{(2)}$ | | $i^{(3)}$ |
| $\hat{\lambda}$ | 3 | | 2 | | 10 |

Toy Example

- Let $\hat{\lambda}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers λ' and λ''

| | $i^{(1)}$ | $i^{(2)}$ | $i^{(3)}$ |
|-----------------|-----------|-----------|-----------|
| $\hat{\lambda}$ | 3 | 2 | 10 |
| λ' | | | |
| <hr/> | | | |

Toy Example

- Let $\hat{\lambda}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers λ' and λ''

| | $i^{(1)}$ | $i^{(2)}$ | $i^{(3)}$ |
|-----------------|-----------|-----------|-----------|
| $\hat{\lambda}$ | 3 | 2 | 10 |
| λ' | 2 | | |
| <hr/> | | | |

Toy Example

- Let $\hat{\lambda}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers λ' and λ''

| | $i^{(1)}$ | $i^{(2)}$ | $i^{(3)}$ |
|-----------------|-----------|-----------|-----------|
| $\hat{\lambda}$ | 3 | 2 | 10 |
| λ' | 2 | 10 | |
| <hr/> | | | |

Toy Example

- Let $\hat{\lambda}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers λ' and λ''

| | $i^{(1)}$ | $i^{(2)}$ | $i^{(3)}$ |
|--|-----------|-----------|-----------|
| $\hat{\lambda}$ | 3 | 2 | 10 |
| λ' | 2 | 10 | |
| → reject, since $\hat{c}_2(\lambda') = 6 > \hat{c}_2(\hat{\lambda}) = 2.5$ | | | |

Toy Example

- Let $\hat{\lambda}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers λ' and λ''

| | $i^{(1)}$ | $i^{(2)}$ | $i^{(3)}$ |
|--|-----------|-----------|-----------|
| $\hat{\lambda}$ | 3 | 2 | 10 |
| λ' | 2 | 10 | |
| λ'' | 3 | | |
| → reject, since $\hat{c}_2(\lambda') = 6 > \hat{c}_2(\hat{\lambda}) = 2.5$ | | | |

Toy Example

- Let $\hat{\lambda}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers λ' and λ''

| | $i^{(1)}$ | $i^{(2)}$ | $i^{(3)}$ |
|--|-----------|-----------|-----------|
| $\hat{\lambda}$ | 3 | 2 | 10 |
| λ' | 2 | 10 | |
| λ'' | 3 | 1 | |
| → reject, since $\hat{c}_2(\lambda') = 6 > \hat{c}_2(\hat{\lambda}) = 2.5$ | | | |

Toy Example

- Let $\hat{\lambda}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers λ' and λ''

| | $i^{(1)}$ | $i^{(2)}$ | $i^{(3)}$ |
|--|-----------|-----------|-----------|
| $\hat{\lambda}$ | 3 | 2 | 10 |
| λ' | 2 | 10 | |
| λ'' | 3 | 1 | 5 |
| \rightarrow reject, since $\hat{c}_2(\lambda') = 6 > \hat{c}_2(\hat{\lambda}) = 2.5$ | | | |

Toy Example

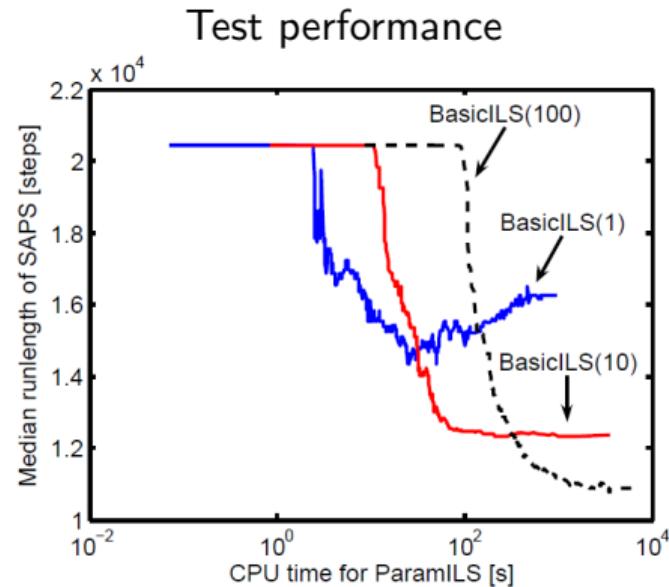
- Let $\hat{\lambda}$ be the incumbent (evaluated on $i^{(1)}, i^{(2)}, i^{(3)}$)
- We'll look at challengers λ' and λ''

| | $i^{(1)}$ | $i^{(2)}$ | $i^{(3)}$ |
|--|-----------|-----------|-----------|
| $\hat{\lambda}$ | 3 | 2 | 10 |
| λ' | 2 | 10 | |
| λ'' | 3 | 1 | 5 |
| → reject, since $\hat{c}_2(\lambda') = 6 > \hat{c}_2(\hat{\lambda}) = 2.5$ | | | |

- new incumbent: $\hat{\lambda} \leftarrow \lambda''$
- Perform an additional run for new $\hat{\lambda}$ to increase confidence over time

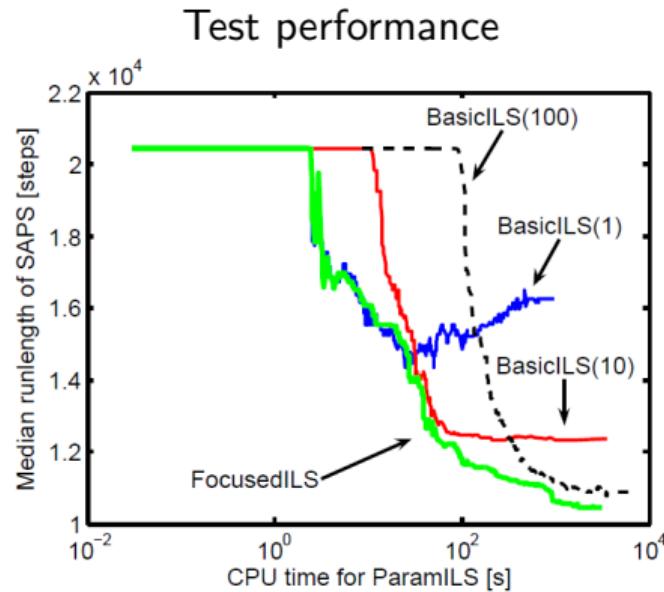
Racing achieves the best of both worlds

Aggressive racing (aka FocusedILS): Fast progress and no overtuning



Racing achieves the best of both worlds

Aggressive racing (aka FocusedILS): Fast progress and no overtuning



Overview of Racing

Input : candidate configurations Λ_{new} , cutoff κ_{max} , previously evaluated runs \mathcal{D}_{Hist} , budget T , incumbent $\hat{\lambda}$

while Λ_{new} *not empty* **do**

$\lambda^{(t)} \leftarrow \text{getNext}(\Lambda_{new});$

Overview of Racing

Input : candidate configurations Λ_{new} , cutoff κ_{max} , previously evaluated runs \mathcal{D}_{Hist} , budget T , incumbent $\hat{\lambda}$

while Λ_{new} *not empty* **do**

$\lambda^{(t)} \leftarrow \text{getNext}(\Lambda_{new});$
 $i, s \leftarrow \text{instance and seed drawn uniformly at random};$
 $c \leftarrow \text{EvaluateRun}(\hat{\lambda}, i, s, \kappa_{max});$
 $\mathcal{D}_{Hist} \leftarrow \mathcal{D}_{Hist} \cup (\hat{\lambda}, i, s, c);$

Overview of Racing

Input : candidate configurations Λ_{new} , cutoff κ_{max} , previously evaluated runs \mathcal{D}_{Hist} , budget T , incumbent $\hat{\lambda}$

while Λ_{new} not empty **do**

$\lambda^{(t)} \leftarrow \text{getNext}(\Lambda_{new})$;

$i, s \leftarrow$ instance and seed drawn uniformly at random;

$c \leftarrow \text{EvaluateRun}(\hat{\lambda}, i, s, \kappa_{max})$;

$\mathcal{D}_{Hist} \leftarrow \mathcal{D}_{Hist} \cup (\hat{\lambda}, i, s, c)$;

while true **do**

$\mathcal{I}^+, \mathbf{s}^+ \leftarrow \text{getAlreadyEvaluatedOn}(\hat{\lambda}, \mathcal{D}_{Hist})$;

$\mathcal{I}^{(t)}, \mathbf{s}^{(t)} \leftarrow \text{getAlreadyEvaluatedOn}(\lambda^{(t)}, \mathcal{D}_{Hist})$;

$i^{(t)}, s^{(t)} \leftarrow$ drawn uniformly at random from $\mathcal{I}^+ \setminus \mathcal{I}^{(t)}$ and $\mathbf{s}^+ \setminus \mathbf{s}^{(t)}$;

Overview of Racing

Input : candidate configurations Λ_{new} , cutoff κ_{max} , previously evaluated runs \mathcal{D}_{Hist} , budget T , incumbent $\hat{\lambda}$

while Λ_{new} not empty **do**

$\lambda^{(t)} \leftarrow \text{getNext}(\Lambda_{new})$;

$i, s \leftarrow$ instance and seed drawn uniformly at random;

$c \leftarrow \text{EvaluateRun}(\hat{\lambda}, i, s, \kappa_{max})$;

$\mathcal{D}_{Hist} \leftarrow \mathcal{D}_{Hist} \cup (\hat{\lambda}, i, s, c)$;

while true **do**

$\mathcal{I}^+, \mathbf{s}^+ \leftarrow \text{getAlreadyEvaluatedOn}(\hat{\lambda}, \mathcal{D}_{Hist})$;

$\mathcal{I}^{(t)}, \mathbf{s}^{(t)} \leftarrow \text{getAlreadyEvaluatedOn}(\lambda^{(t)}, \mathcal{D}_{Hist})$;

$i^{(t)}, s^{(t)} \leftarrow$ drawn uniformly at random from $\mathcal{I}^+ \setminus \mathcal{I}^{(t)}$ and $\mathbf{s}^+ \setminus \mathbf{s}^{(t)}$;

$c_i \leftarrow \text{EvaluateRun}(\lambda^{(t)}, i^{(t)}, s^{(t)}, \kappa_{max})$;

$\mathcal{D}_{Hist} \leftarrow \mathcal{D}_{Hist} \cup (\lambda^{(t)}, i^{(t)}, s^{(t)}, c^{(t)})$;

Overview of Racing

Input : candidate configurations Λ_{new} , cutoff κ_{max} , previously evaluated runs \mathcal{D}_{Hist} , budget T , incumbent $\hat{\lambda}$

while Λ_{new} not empty **do**

- $\lambda^{(t)} \leftarrow \text{getNext}(\Lambda_{new});$
- $i, s \leftarrow \text{instance and seed drawn uniformly at random};$
- $c \leftarrow \text{EvaluateRun}(\hat{\lambda}, i, s, \kappa_{max});$
- $\mathcal{D}_{Hist} \leftarrow \mathcal{D}_{Hist} \cup (\hat{\lambda}, i, s, c);$
- while** true **do**

 - $\mathcal{I}^+, \mathbf{s}^+ \leftarrow \text{getAlreadyEvaluatedOn}(\hat{\lambda}, \mathcal{D}_{Hist});$
 - $\mathcal{I}^{(t)}, \mathbf{s}^{(t)} \leftarrow \text{getAlreadyEvaluatedOn}(\lambda^{(t)}, \mathcal{D}_{Hist});$
 - $i^{(t)}, s^{(t)} \leftarrow \text{drawn uniformly at random from } \mathcal{I}^+ \setminus \mathcal{I}^{(t)} \text{ and } \mathbf{s}^+ \setminus \mathbf{s}^{(t)};$
 - $c_i \leftarrow \text{EvaluateRun}(\lambda^{(t)}, i^{(t)}, s^{(t)}, \kappa_{max});$
 - $\mathcal{D}_{Hist} \leftarrow \mathcal{D}_{Hist} \cup (\lambda^{(t)}, i^{(t)}, s^{(t)}, c^{(t)});$
 - if** average cost of $\lambda^{(t)}$ > average cost of $\hat{\lambda}$ across $\mathcal{I}^{(t)}$ and $\mathbf{s}^{(t)}$ **then**

 - $\sqsubset \text{break};$

Overview of Racing

Input : candidate configurations Λ_{new} , cutoff κ_{max} , previously evaluated runs \mathcal{D}_{Hist} , budget T , incumbent $\hat{\lambda}$

while Λ_{new} not empty **do**

$\lambda^{(t)} \leftarrow \text{getNext}(\Lambda_{new});$
 $i, s \leftarrow \text{instance and seed drawn uniformly at random};$
 $c \leftarrow \text{EvaluateRun}(\hat{\lambda}, i, s, \kappa_{max});$
 $\mathcal{D}_{Hist} \leftarrow \mathcal{D}_{Hist} \cup (\hat{\lambda}, i, s, c);$
while true **do**
 $\mathcal{I}^+, s^+ \leftarrow \text{getAlreadyEvaluatedOn}(\hat{\lambda}, \mathcal{D}_{Hist});$
 $\mathcal{I}^{(t)}, s^{(t)} \leftarrow \text{getAlreadyEvaluatedOn}(\lambda^{(t)}, \mathcal{D}_{Hist});$
 $i^{(t)}, s^{(t)} \leftarrow \text{drawn uniformly at random from } \mathcal{I}^+ \setminus \mathcal{I}^{(t)} \text{ and } s^+ \setminus s^{(t)};$
 $c_i \leftarrow \text{EvaluateRun}(\lambda^{(t)}, i^{(t)}, s^{(t)}, \kappa_{max});$
 $\mathcal{D}_{Hist} \leftarrow \mathcal{D}_{Hist} \cup (\lambda^{(t)}, i^{(t)}, s^{(t)}, c^{(t)});$
 if average cost of $\lambda^{(t)}$ > average cost of $\hat{\lambda}$ across $\mathcal{I}^{(t)}$ and $s^{(t)}$ **then**
 \sqsubset break;
 else if average cost of $\lambda^{(t)}$ < average cost of $\hat{\lambda}$ and $\mathcal{I}^+ = \mathcal{I}^{(t)}$ and $s^+ = s^{(t)}$ **then**
 $\hat{\lambda} \leftarrow \lambda^{(t)};$

Overview of Racing

Input : candidate configurations Λ_{new} , cutoff κ_{max} , previously evaluated runs \mathcal{D}_{Hist} , budget T , incumbent $\hat{\lambda}$

while Λ_{new} not empty **do**

$\lambda^{(t)} \leftarrow \text{getNext}(\Lambda_{new})$;

$i, s \leftarrow$ instance and seed drawn uniformly at random;

$c \leftarrow \text{EvaluateRun}(\hat{\lambda}, i, s, \kappa_{max})$;

$\mathcal{D}_{Hist} \leftarrow \mathcal{D}_{Hist} \cup (\hat{\lambda}, i, s, c)$;

while true **do**

$\mathcal{I}^+, s^+ \leftarrow \text{getAlreadyEvaluatedOn}(\hat{\lambda}, \mathcal{D}_{Hist})$;

$\mathcal{I}^{(t)}, s^{(t)} \leftarrow \text{getAlreadyEvaluatedOn}(\lambda^{(t)}, \mathcal{D}_{Hist})$;

$i^{(t)}, s^{(t)} \leftarrow$ drawn uniformly at random from $\mathcal{I}^+ \setminus \mathcal{I}^{(t)}$ and $s^+ \setminus s^{(t)}$;

$c_i \leftarrow \text{EvaluateRun}(\lambda^{(t)}, i^{(t)}, s^{(t)}, \kappa_{max})$;

$\mathcal{D}_{Hist} \leftarrow \mathcal{D}_{Hist} \cup (\lambda^{(t)}, i^{(t)}, s^{(t)}, c^{(t)})$;

if average cost of $\lambda^{(t)}$ > average cost of $\hat{\lambda}$ across $\mathcal{I}^{(t)}$ and $s^{(t)}$ **then**

└ break;

else if average cost of $\lambda^{(t)}$ < average cost of $\hat{\lambda}$ and $\mathcal{I}^+ = \mathcal{I}^{(t)}$ and $s^+ = s^{(t)}$ **then**

└ $\hat{\lambda} \leftarrow \lambda^{(t)}$;

if time spent exceeds T or Λ_{new} is empty **then**

└ **return** $\hat{\lambda}, \mathcal{D}_{Hist}$

AutoML: Beyond AutoML

Capping of Runtimes

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Adaptive capping [Hutter et al. 2009]

- Assumptions
 - ▶ optimization of runtime
 - ▶ each configuration run has a time limit (e.g., 300 sec)

Adaptive capping [Hutter et al. 2009]

- Assumptions
 - ▶ optimization of runtime
 - ▶ each configuration run has a time limit (e.g., 300 sec)
- E.g., $\hat{\lambda}$ needed 1 sec to solve i_1
 - ▶ Do we need to run λ' for 300 sec?
 - ▶ Terminate evaluation of λ' once guaranteed to be worse than $\hat{\lambda}$

Adaptive capping [Hutter et al. 2009]

- Assumptions
 - ▶ optimization of runtime
 - ▶ each configuration run has a time limit (e.g., 300 sec)
 - E.g., $\hat{\lambda}$ needed 1 sec to solve i_1
 - ▶ Do we need to run λ' for 300 sec?
 - ▶ Terminate evaluation of λ' once guaranteed to be worse than $\hat{\lambda}$
- ⇝ To compare against $\hat{\lambda}$ based on N runs,
we can terminate evaluation of λ' after time $\sum_{k=1}^N c(\hat{\lambda}, i_k)$

Toy-Example: Adaptive capping

runtime cutoff $\kappa = 300$, comparison based on 2 instances

| | i_1 | i_2 |
|-----------------|-------|-------|
| $\hat{\lambda}$ | 4 | 2 |

Toy-Example: Adaptive capping

runtime cutoff $\kappa = 300$, comparison based on 2 instances

| | i_1 | i_2 |
|---------------------------------|------------|-------|
| $\hat{\lambda}$ | 4 | 2 |
| <i>Without adaptive capping</i> | | |
| | λ' | |

Toy-Example: Adaptive capping

runtime cutoff $\kappa = 300$, comparison based on 2 instances

| | i_1 | i_2 |
|---------------------------------|-------|-------|
| $\hat{\lambda}$ | 4 | 2 |
| <i>Without adaptive capping</i> | | |
| λ' | 3 | |

Toy-Example: Adaptive capping

runtime cutoff $\kappa = 300$, comparison based on 2 instances

| | i_1 | i_2 |
|---------------------------------|-------|-------|
| $\hat{\lambda}$ | 4 | 2 |
| <i>Without adaptive capping</i> | | |
| λ' | 3 | 300 |

Toy-Example: Adaptive capping

runtime cutoff $\kappa = 300$, comparison based on 2 instances

| | i_1 | i_2 |
|---------------------------------|---|-------|
| $\hat{\lambda}$ | 4 | 2 |
| <i>Without adaptive capping</i> | | |
| λ' | 3 | 300 |
| | \rightarrow reject λ' (cost: 303) | |

Toy-Example: Adaptive capping

runtime cutoff $\kappa = 300$, comparison based on 2 instances

| | i_1 | i_2 |
|---------------------------------|-------|-------|
| $\hat{\lambda}$ | 4 | 2 |
| <hr/> | | |
| <i>Without adaptive capping</i> | | |
| λ' | 3 | 300 |
| <hr/> | | |
| <i>With adaptive capping</i> | | |
| λ' | | |

→ reject λ' (cost: 303)

Toy-Example: Adaptive capping

runtime cutoff $\kappa = 300$, comparison based on 2 instances

| | i_1 | i_2 |
|---------------------------------|-------|---|
| $\hat{\lambda}$ | 4 | 2 |
| <hr/> | | |
| <i>Without adaptive capping</i> | | |
| λ' | 3 | 300 |
| <hr/> | | |
| <i>With adaptive capping</i> | | |
| λ' | 3 | \rightarrow reject λ' (cost: 303) |

Toy-Example: Adaptive capping

runtime cutoff $\kappa = 300$, comparison based on 2 instances

| | i_1 | i_2 |
|---------------------------------|-------|-------|
| $\hat{\lambda}$ | 4 | 2 |
| <hr/> | | |
| <i>Without adaptive capping</i> | | |
| λ' | 3 | 300 |
| <hr/> | | |
| <i>With adaptive capping</i> | | |
| λ' | 3 | 300 |

Toy-Example: Adaptive capping

runtime cutoff $\kappa = 300$, comparison based on 2 instances

| | i_1 | i_2 |
|---|-------|-------|
| $\hat{\lambda}$ | 4 | 2 |
| <i>Without adaptive capping</i> | | |
| λ' | 3 | 300 |
| \rightarrow reject λ' (cost: 303) | | |
| <i>With adaptive capping</i> | | |
| λ' | 3 | 300 |
| \rightarrow cut off after $\kappa = 4$ seconds, reject λ' (cost: 7) | | |

Toy-Example: Adaptive capping

runtime cutoff $\kappa = 300$, comparison based on 2 instances

| | i_1 | i_2 |
|---|-------|-------|
| $\hat{\lambda}$ | 4 | 2 |
| <i>Without adaptive capping</i> | | |
| λ' | 3 | 300 |
| \rightarrow reject λ' (cost: 303) | | |
| <i>With adaptive capping</i> | | |
| λ' | 3 | 300 |
| \rightarrow cut off after $\kappa = 4$ seconds, reject λ' (cost: 7) | | |

Note: To combine adaptive capping with BO, we need to impute the censored observations caused by adaptive capping. [Hutter et al. 2011]

Overview of Racing and Adaptive Capping

Input : candidate configurations Λ_{new} , cutoff κ_{max} , previously evaluated runs \mathcal{D}_{Hist} , budget T , incumbent $\hat{\lambda}$

while Λ_{new} not empty **do**

$\lambda^{(t)} \leftarrow \text{getNext}(\Lambda_{new})$;

 [... add new run for incumbent ...];

while true **do**

$\mathcal{I}^+, s^+ \leftarrow \text{getAlreadyEvaluatedOn}(\hat{\lambda}, \mathcal{D}_{Hist})$;

$\mathcal{I}^{(t)}, s^{(t)} \leftarrow \text{getAlreadyEvaluatedOn}(\lambda^{(t)}, \mathcal{D}_{Hist})$;

$i^{(t)}, s^{(t)} \leftarrow$ drawn uniformly at random from $\mathcal{I}_+ \setminus \mathcal{I}^{(t)}$ and $s^+ \setminus s^{(t)}$;

$\kappa^{(i)} \leftarrow \text{AdaptCutoff}(\kappa_{max}, \langle (\lambda^{(j)}, c^{(j)}) \rangle_{\lambda^{(j)} = \lambda^+}) \cdot \xi$;

$c_i \leftarrow \text{EvaluateRun}(\lambda^{(t)}, i^{(t)}, s^{(i)}, \kappa^{(i)})$;

$\mathcal{D}_{Hist} \leftarrow \mathcal{D}_{Hist} \cup (\lambda^{(t)}, i^{(t)}, s^{(t)}, c^{(t)})$;

if average cost of $\lambda^{(t)}$ > average cost of $\hat{\lambda}$ across $\mathcal{I}^{(t)}$ and $s^{(t)}$ **then**

 └ break;

else if average cost of $\lambda^{(t)}$ < average cost of $\hat{\lambda}$ and $\mathcal{I}^+ = \mathcal{I}^{(t)}$ and $s^{(t)} = s^+$ **then**

 └ $\hat{\lambda} \leftarrow \lambda^{(t)}$;

if time spent exceeds T or Λ_{new} is empty **then**

 └ **return** $\hat{\lambda}, \mathcal{D}_{Hist}$

AutoML: Beyond AutoML

Structured Procrastination

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Idea

- incumbent driven methods (such as aggressive racing with adaptive capping) provide no theoretical guarantees about runtime

Idea

- incumbent driven methods (such as aggressive racing with adaptive capping) provide no theoretical guarantees about runtime
- task: for a fix set of configuration, identify the one with the best average runtime
- instead of top-down capping, use bottom up capping

Idea

- incumbent driven methods (such as aggressive racing with adaptive capping) provide no theoretical guarantees about runtime
- task: for a fix set of configuration, identify the one with the best average runtime
- instead of top-down capping, use bottom up capping
- start with a minimal cap-time and increase it step by step

Idea

- incumbent driven methods (such as aggressive racing with adaptive capping) provide no theoretical guarantees about runtime
- task: for a fix set of configuration, identify the one with the best average runtime
- instead of top-down capping, use bottom up capping
- start with a minimal cap-time and increase it step by step
- unsuccessful runs (with too small cap-time) are procrastinated to later
 - ~ worst-case runtime guarantees

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 1 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 2 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while b remains **do**

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 3 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while b remains **do**

determine the best $\hat{\lambda}$ according to $R(\lambda, \cdot)$;

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 4 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while b remains **do**

determine the best $\hat{\lambda}$ according to $R(\lambda, \cdot)$;

get first element $(i^{(k)}, \kappa)$ from $Q_{\hat{\lambda}}$;

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 5 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while b remains **do**

 determine the best $\hat{\lambda}$ according to $R(\lambda, \cdot)$;

 get first element $(i^{(k)}, \kappa)$ from $Q_{\hat{\lambda}}$;

 Run $\hat{\lambda}$ on $i^{(k)}$ capped at κ ;

if terminates **then**

$R(\hat{\lambda}, i^{(k)}) := t$;

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 6 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while b remains **do**

 determine the best $\hat{\lambda}$ according to $R(\lambda, \cdot)$;

 get first element $(i^{(k)}, \kappa)$ from $Q_{\hat{\lambda}}$;

 Run $\hat{\lambda}$ on $i^{(k)}$ capped at κ ;

if terminates **then**

$R(\hat{\lambda}, i^{(k)}) := t$;

else

$R(\hat{\lambda}, i^{(k)}) := \kappa$;

 Insert $(i^{(k)}, 2 \cdot \kappa)$ at the end of $Q_{\hat{\lambda}}$;

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 7 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while b remains **do**

 determine the best $\hat{\lambda}$ according to $R(\lambda, \cdot)$;

 get first element $(i^{(k)}, \kappa)$ from $Q_{\hat{\lambda}}$;

 Run $\hat{\lambda}$ on $i^{(k)}$ capped at κ ;

if terminates **then**

$R(\hat{\lambda}, i^{(k)}) := t$;

else

$R(\hat{\lambda}, i^{(k)}) := \kappa$;

 Insert $(i^{(k)}, 2 \cdot \kappa)$ at the end of $Q_{\hat{\lambda}}$;

 Replenish queue $Q_{\hat{\lambda}}$ if too small;

Structured Procrastination: Outline [Kleinberg et al. 2017]

Algorithm 8 Structured Procrastination

Input : finite (small) set of configurations Λ , minimal cap-time κ_0 , sequence of instances $i^{(1)}, \dots, i^{(N)}$

Output : best incumbent configuration $\hat{\lambda}$

for each $\lambda \in \Lambda$ initialize a queue Q_λ with entries $(i^{(k)}, \kappa_0)$;

// small queue in the beginning

initialize a look-up table $R(\lambda, i) = 0$;

// optimistic runtime estimate

while b remains **do**

 determine the best $\hat{\lambda}$ according to $R(\lambda, \cdot)$;

 get first element $(i^{(k)}, \kappa)$ from $Q_{\hat{\lambda}}$;

 Run $\hat{\lambda}$ on $i^{(k)}$ capped at κ ;

if terminates **then**

$R(\hat{\lambda}, i^{(k)}) := t$;

else

$R(\hat{\lambda}, i^{(k)}) := \kappa$;

 Insert $(i^{(k)}, 2 \cdot \kappa)$ at the end of $Q_{\hat{\lambda}}$;

 Replenish queue $Q_{\hat{\lambda}}$ if too small;

return $\hat{\lambda} := \arg \min_{\lambda \in \Lambda} \sum_{k=1}^N R(\lambda, i^{(k)})$

Extensions

- We can derive theoretical optimality guarantees with structured procrastination (SP)

Extensions

- We can derive theoretical optimality guarantees with structured procrastination (SP)
- In practice, plain SP is rather slow and requires the setting of some hyperparameters

Extensions

- We can derive theoretical optimality guarantees with structured procrastination (SP)
- In practice, plain SP is rather slow and requires the setting of some hyperparameters
- Several extensions and similar ideas:
 - ▶ [Kleinberg et al. 2019]
 - ▶ [Weisz et al. 2018]
 - ▶ [Weisz et al. 2019]

AutoML: Beyond AutoML

Best Practices for Algorithm Configuration

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

Pitfalls & Best Practices

The *goals* of automated algorithm configuration include:

- ① reducing the expertise required to use an algorithm

Pitfalls & Best Practices

The **goals** of automated algorithm configuration include:

- ① reducing the expertise required to use an algorithm
- ② less human-time

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

- ① reducing the expertise required to use an algorithm
- ② less human-time
- ③ tuning algorithms to the task at hand

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

- ① reducing the expertise required to use an algorithm
- ② less human-time
- ③ tuning algorithms to the task at hand
- ④ faster development of algorithms

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

- ① reducing the expertise required to use an algorithm
- ② less human-time
- ③ tuning algorithms to the task at hand
- ④ faster development of algorithms
- ⑤ facilitating systematic and reproducible research

BUT:

- ① algorithm configuration can lead to over-tuning

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

- ① reducing the expertise required to use an algorithm
- ② less human-time
- ③ tuning algorithms to the task at hand
- ④ faster development of algorithms
- ⑤ facilitating systematic and reproducible research

BUT:

- ① algorithm configuration can lead to over-tuning
- ② using algorithm configuration requires (at least some) expertise in algorithm configuration

Pitfalls & Best Practices

The goals of automated algorithm configuration include:

- ① reducing the expertise required to use an algorithm
- ② less human-time
- ③ tuning algorithms to the task at hand
- ④ faster development of algorithms
- ⑤ facilitating systematic and reproducible research

BUT:

- ① algorithm configuration can lead to over-tuning
- ② using algorithm configuration requires (at least some) expertise in algorithm configuration
- ③ if done wrong, waste of time and compute resources

Setting up AC

9 Steps to your well-performing algorithm:

- ① Define your performance metric

Setting up AC

9 Steps to your well-performing algorithm:

- ① Define your performance metric
- ② Define instance set

Setting up AC

9 Steps to your well-performing algorithm:

- ① Define your performance metric
- ② Define instance set
- ③ Split your instances in training and test

Setting up AC

9 Steps to your well-performing algorithm:

- ① Define your performance metric
- ② Define instance set
- ③ Split your instances in training and test
- ④ Define the configuration space

Setting up AC

9 Steps to your well-performing algorithm:

- ① Define your performance metric
- ② Define instance set
- ③ Split your instances in training and test
- ④ Define the configuration space
- ⑤ Choose your preferred configurator

Setting up AC

9 Steps to your well-performing algorithm:

- ① Define your performance metric
- ② Define instance set
- ③ Split your instances in training and test
- ④ Define the configuration space
- ⑤ Choose your preferred configurator
- ⑥ Implement an interface between your algorithm and the configurator

Setting up AC

9 Steps to your well-performing algorithm:

- ① Define your performance metric
- ② Define instance set
- ③ Split your instances in training and test
- ④ Define the configuration space
- ⑤ Choose your preferred configurator
- ⑥ Implement an interface between your algorithm and the configurator
- ⑦ Define resource limitations of your algorithm

Setting up AC

9 Steps to your well-performing algorithm:

- ① Define your performance metric
- ② Define instance set
- ③ Split your instances in training and test
- ④ Define the configuration space
- ⑤ Choose your preferred configurator
- ⑥ Implement an interface between your algorithm and the configurator
- ⑦ Define resource limitations of your algorithm
- ⑧ Run the configurator on your algorithm and the training instances

Setting up AC

9 Steps to your well-performing algorithm:

- ① Define your performance metric
- ② Define instance set
- ③ Split your instances in training and test
- ④ Define the configuration space
- ⑤ Choose your preferred configurator
- ⑥ Implement an interface between your algorithm and the configurator
- ⑦ Define resource limitations of your algorithm
- ⑧ Run the configurator on your algorithm and the training instances
- ⑨ Validate the eventually returned configuration on your test instances

Pitfall 1: Trust your algorithm

We have encountered algorithms that

- ignored resource limitations
- returned wrong solutions
- even returned negative runtimes

Pitfall 1: Trust your algorithm

We have encountered algorithms that

- ignored resource limitations
- returned wrong solutions
- even returned negative runtimes

Best Practice 1: Never trust your algorithm

Explicitly check and use external software to:

- ① ensure resource limitations
- ② terminate your algorithm
- ③ verify returned solutions
- ④ measure performance

Pitfall 2: File System

Algorithm configurators ...

- often produce quite some log files (e.g., for each algorithm run)
- are often used on HPC clusters with a shared file system

Pitfall 2: File System

Algorithm configurators ...

- often produce quite some log files (e.g., for each algorithm run)
 - are often used on HPC clusters with a shared file system
- ~~~ It's surprisingly easy to substantially slow down a shared file system

Pitfall 2: File System

Algorithm configurators ...

- often produce quite some log files (e.g., for each algorithm run)
 - are often used on HPC clusters with a shared file system
- ~~~ It's surprisingly easy to substantially slow down a shared file system

Best Practice 2: Don't use the Shared File System

To relieve the file system of a HPC cluster:

- design well which files are required and which are not
- use a local (SSD) disc

Pitfall 3: Over-tuning

It's easy to over-tune to different aspects, incl.:

- training instances
- random seeds
- machine type

Pitfall 3: Over-tuning

It's easy to over-tune to different aspects, incl.:

- training instances
- random seeds
- machine type

In practice, it can be hard to prevent over-tuning, e.g., by

- using larger instance sets
- tuning on the target hardware

Best Practice 3: Check for Over-Tuning

Check for over-tuning by validating your final configuration on

- many random seeds
- a large set of unused test instances
- a different hardware

Pitfall 4: Heterogeneous Instance Sets

Algorithm configurators...

- use some kind of racing to not evaluate each configuration on all instances

Pitfall 4: Heterogeneous Instance Sets

Algorithm configurators...

- use some kind of racing to not evaluate each configuration on all instances
- can be mislead on subsets of instances if the instance set is heterogeneous

Pitfall 4: Heterogeneous Instance Sets

Algorithm configurators...

- use some kind of racing to not evaluate each configuration on all instances
- can be mislead on subsets of instances if the instance set is heterogeneous

~~ returned configurations often perform worse than default configurations
in the validation phase

Pitfall 4: Heterogeneous Instance Sets

Algorithm configurators...

- use some kind of racing to not evaluate each configuration on all instances
- can be misled on subsets of instances if the instance set is heterogeneous

~~ returned configurations often perform worse than default configurations
in the validation phase

Best Practice 4: Ensure Homogeneity

Algorithm configurators should only run on homogeneous instance sets.

Different degrees of homogeneity:

- Strong homogeneity: all instances agree on the ranking of configurations
- Weak homogeneity: all instances agree on the top-performing configurations

More Pitfalls and Best Practices

... can be found in [Eggensperger et al. 2019]

AutoML: Beyond AutoML

Per-Instance Algorithm Configuration

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Homogeneous vs. Heterogeneous Instances

Assumption of AC: Homogeneous Instance Distribution

- Algorithm configuration tools assume that the instance distribution is homogeneous (see video on "Best Practices for AC")
- Important because
 - ▶ there is a well-performing configuration for all (or most) instances
 - ▶ the racing algorithm can make educated decisions on subsets

Homogeneous vs. Heterogeneous Instances

Assumption of AC: Homogeneous Instance Distribution

- Algorithm configuration tools assume that the instance distribution is homogeneous (see video on "Best Practices for AC")
- Important because
 - ▶ there is a well-performing configuration for all (or most) instances
 - ▶ the racing algorithm can make educated decisions on subsets

Violated assumption of AC: Heterogeneous Instance Distribution

- The racing algorithm will make inconsistent (or even wrong) decisions
- There is no single well-performing configuration for all instances

Homogeneous vs. Heterogeneous Instances

Assumption of AC: Homogeneous Instance Distribution

- Algorithm configuration tools assume that the instance distribution is homogeneous (see video on "Best Practices for AC")
- Important because
 - ▶ there is a well-performing configuration for all (or most) instances
 - ▶ the racing algorithm can make educated decisions on subsets

Violated assumption of AC: Heterogeneous Instance Distribution

- The racing algorithm will make inconsistent (or even wrong) decisions
- There is no single well-performing configuration for all instances

~~ What should we do with heterogeneous instance distributions?

Why are systems for heterogeneous instance distributions important?

- ➊ We cannot guarantee homogeneity in practice
 - ➊ Instances might get larger and harder
 - ➋ The underlying task or business case might change

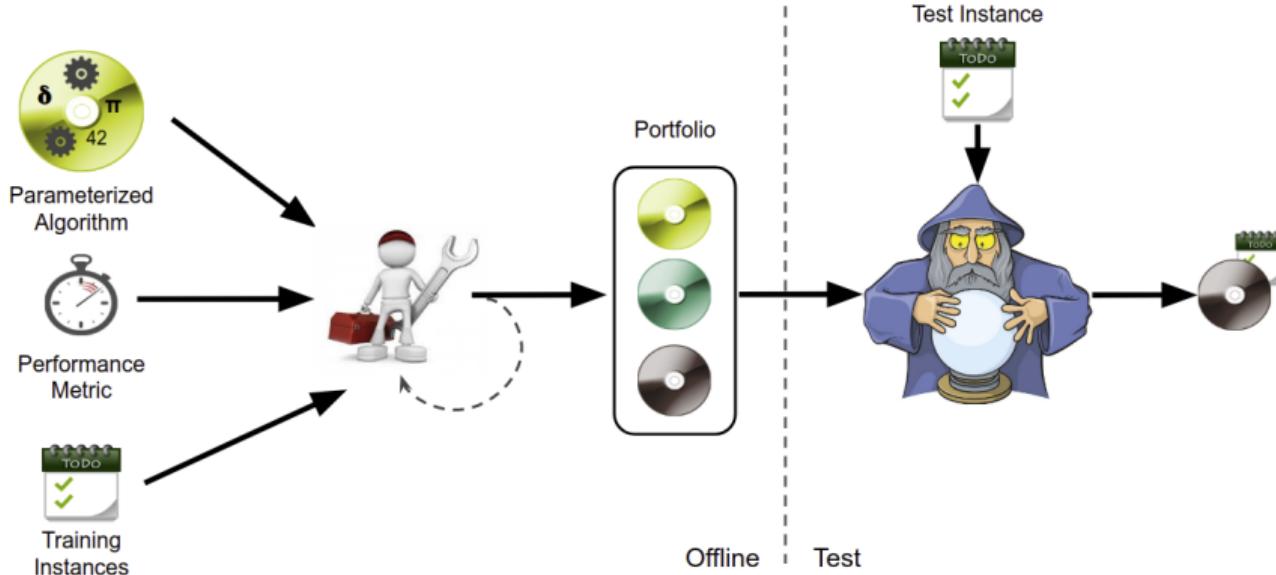
Why are systems for heterogeneous instance distributions important?

- ➊ We cannot guarantee homogeneity in practice
 - ➊ Instances might get larger and harder
 - ➋ The underlying task or business case might change
- ➋ We don't want to do algorithm configuration always from scratch

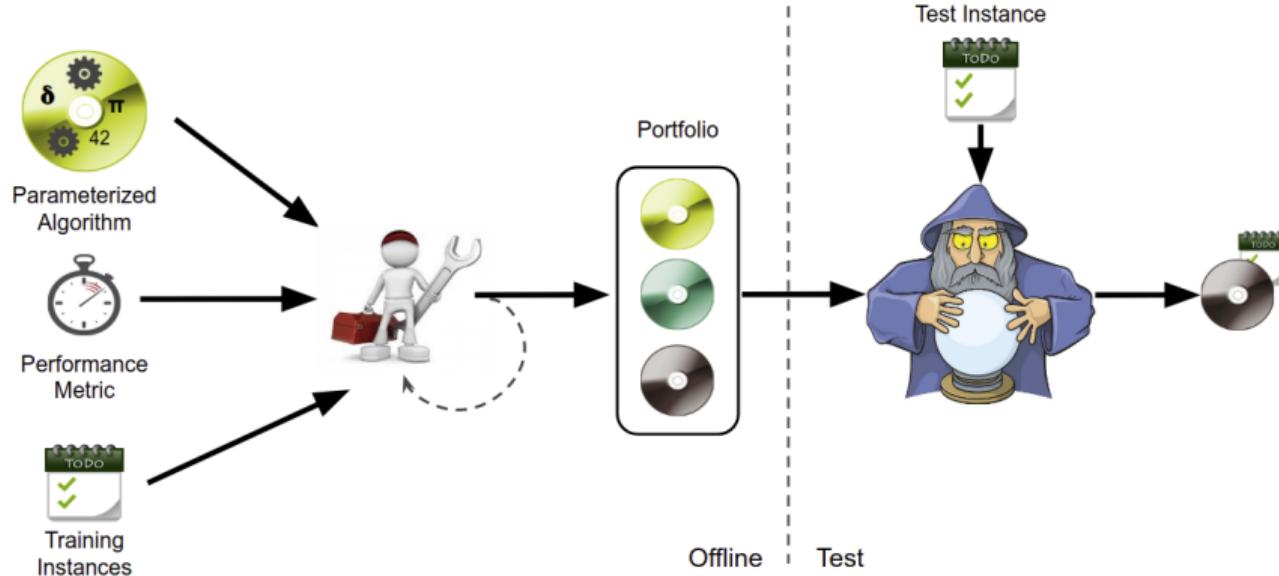
Why are systems for heterogeneous instance distributions important?

- ➊ We cannot guarantee homogeneity in practice
 - ➊ Instances might get larger and harder
 - ➋ The underlying task or business case might change
- ➋ We don't want to do algorithm configuration always from scratch
- ➌ An adaptive configuration system would be the holy grail
 - ~~ hard to achieve

PIAC: Per-Instance Algorithm Configuration



PIAC: Per-Instance Algorithm Configuration



- You can use whichever kind of algorithm selection (wizard) you want
- **Challenge:** Building a portfolio
- **Use case:** Instances are heterogeneous

PIAC: Manual Expert Approach

Basic Assumption

Heterogeneous instance set can be divided into homogeneous subsets

PIAC: Manual Expert Approach

Basic Assumption

Heterogeneous instance set can be divided into homogeneous subsets

Manual Expert

- An expert knows the homogeneous subsets (e.g., origin of instances)
- Determine a well-performing configuration on each subset
→ portfolio of configurations
- Use Algorithm Selection to select a well-performing configuration on each instance

Idea

Training:

- ① Cluster instances into homogeneous subsets
(using g -means in the instance feature space)
- ② Apply algorithm configuration (here GGA) on each instance set

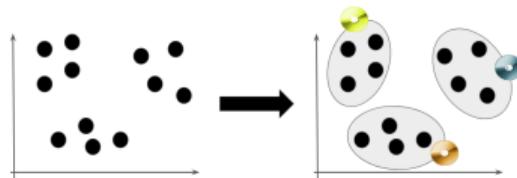
Idea

Training:

- ① Cluster instances into homogeneous subsets
(using g -means in the instance feature space)
- ② Apply algorithm configuration (here GGA) on each instance set

Test:

- ① Determine the nearest cluster (k -NN with $k = 1$) in feature space
- ② Apply optimized configuration of this cluster



Idea

- Iteratively add configurations to a portfolio \mathbf{P} , start with $\mathbf{P} = \emptyset$
- In each iteration, determine configuration that is complementary to \mathbf{P}
 - ~ Maximize marginal contribution to \mathbf{P}

Idea

- Iteratively add configurations to a portfolio \mathbf{P} , start with $\mathbf{P} = \emptyset$
- In each iteration, determine configuration that is complementary to \mathbf{P}
 - ~ Maximize marginal contribution to \mathbf{P}

Marginal contribution of a configuration λ to a portfolio \mathbf{P} :

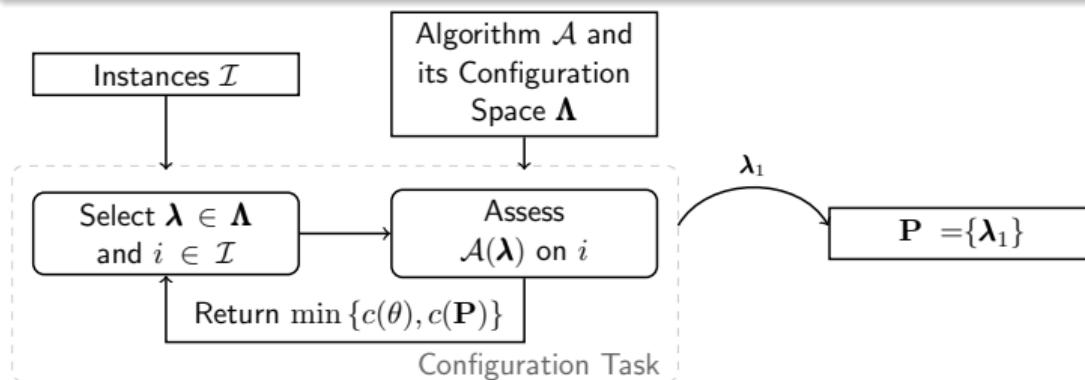
$$c(\mathbf{P}) - c(\mathbf{P} \cup \{\lambda\})$$

Idea

- Iteratively add configurations to a portfolio \mathbf{P} , start with $\mathbf{P} = \emptyset$
- In each iteration, determine configuration that is complementary to \mathbf{P}
 - ~ Maximize marginal contribution to \mathbf{P}

Marginal contribution of a configuration λ to a portfolio \mathbf{P} :

$$c(\mathbf{P}) - c(\mathbf{P} \cup \{\lambda\})$$

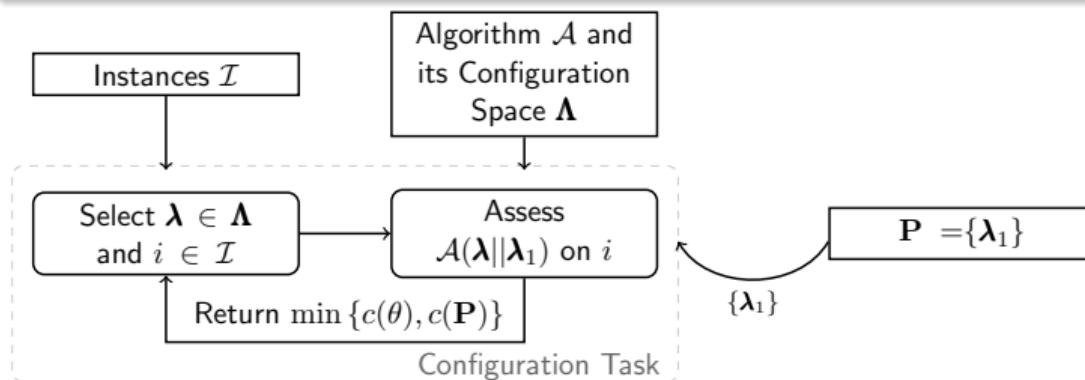


Idea

- Iteratively add configurations to a portfolio \mathbf{P} , start with $\mathbf{P} = \emptyset$
- In each iteration, determine configuration that is complementary to \mathbf{P}
 - ~ Maximize marginal contribution to \mathbf{P}

Marginal contribution of a configuration λ to a portfolio \mathbf{P} :

$$c(\mathbf{P}) - c(\mathbf{P} \cup \{\lambda\})$$

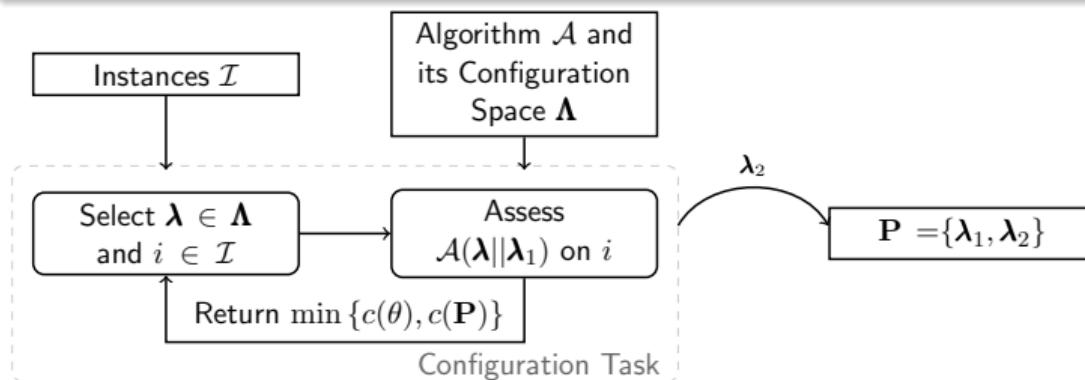


Idea

- Iteratively add configurations to a portfolio \mathbf{P} , start with $\mathbf{P} = \emptyset$
- In each iteration, determine configuration that is complementary to \mathbf{P}
 - ~ Maximize marginal contribution to \mathbf{P}

Marginal contribution of a configuration λ to a portfolio \mathbf{P} :

$$c(\mathbf{P}) - c(\mathbf{P} \cup \{\lambda\})$$

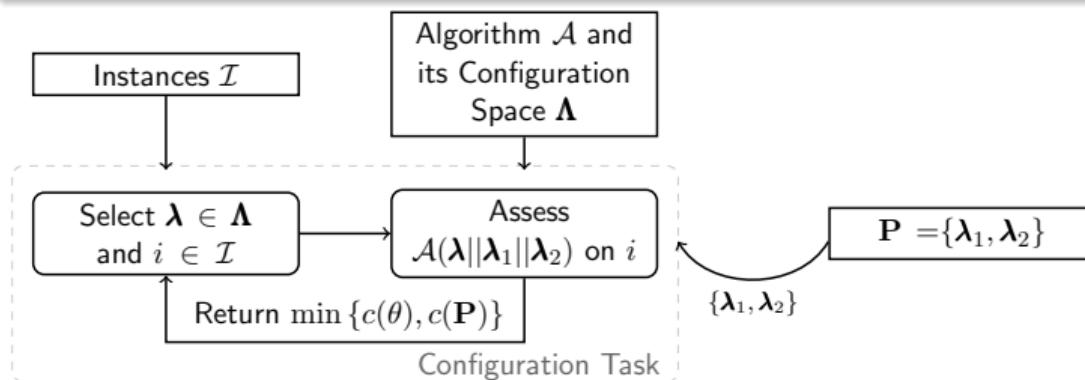


Idea

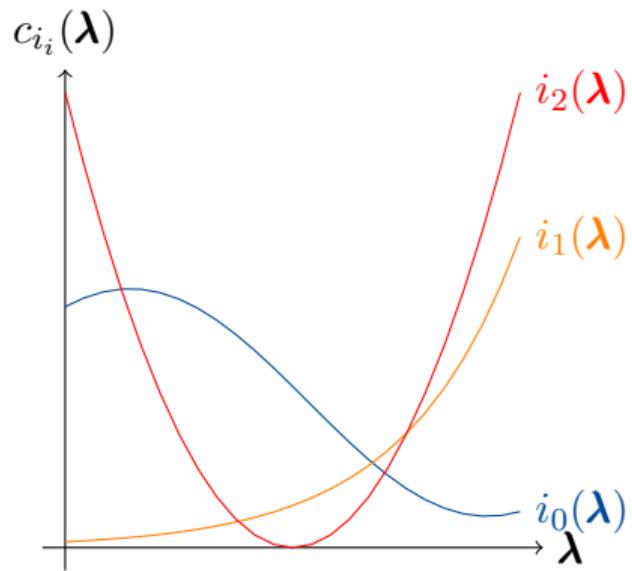
- Iteratively add configurations to a portfolio \mathbf{P} , start with $\mathbf{P} = \emptyset$
- In each iteration, determine configuration that is complementary to \mathbf{P}
 - ~ Maximize marginal contribution to \mathbf{P}

Marginal contribution of a configuration λ to a portfolio \mathbf{P} :

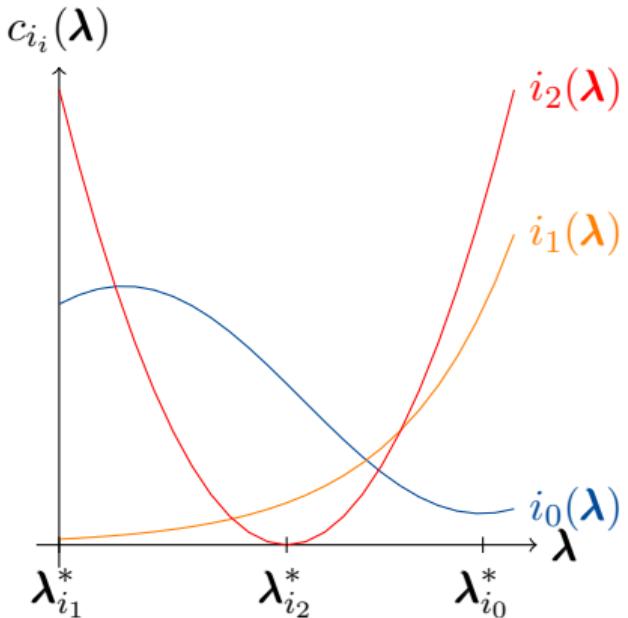
$$c(\mathbf{P}) - c(\mathbf{P} \cup \{\lambda\})$$



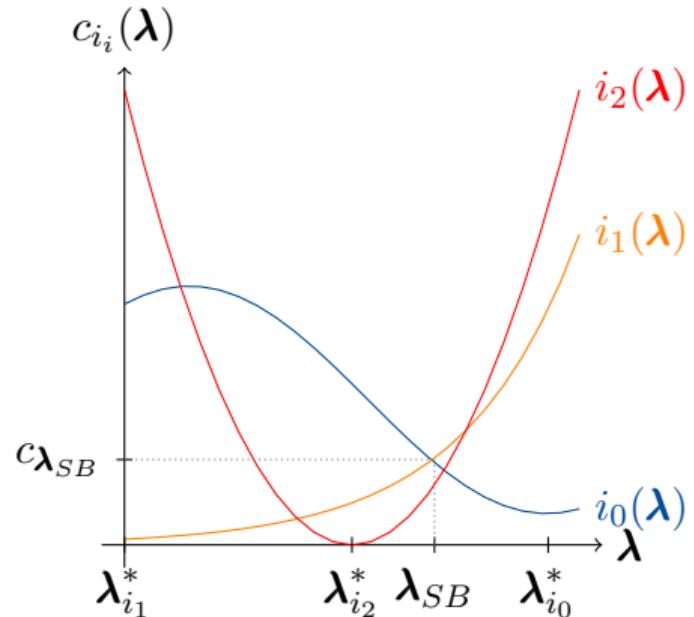
Heterogeneous example



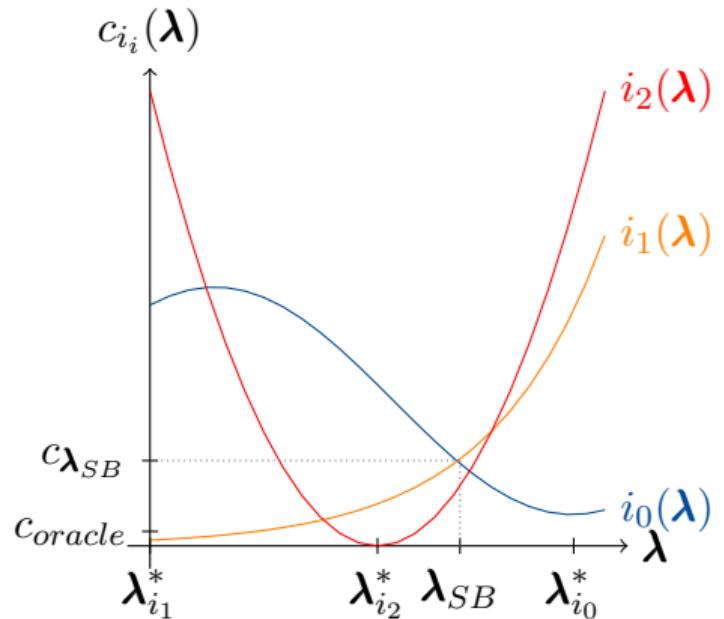
Heterogeneous example



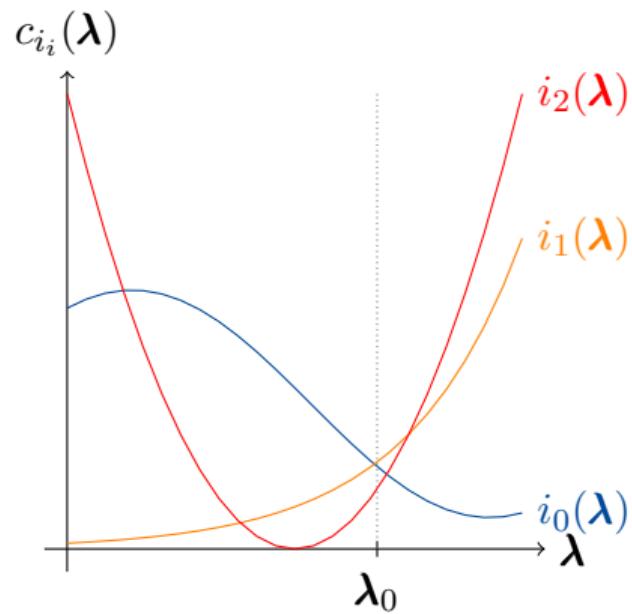
Heterogeneous example



Heterogeneous example

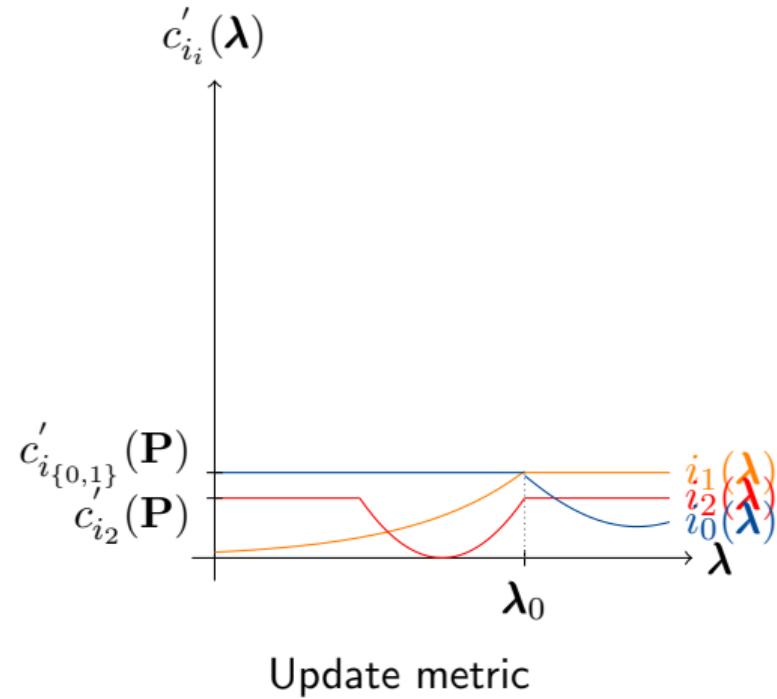


Hydra: Iteration 1

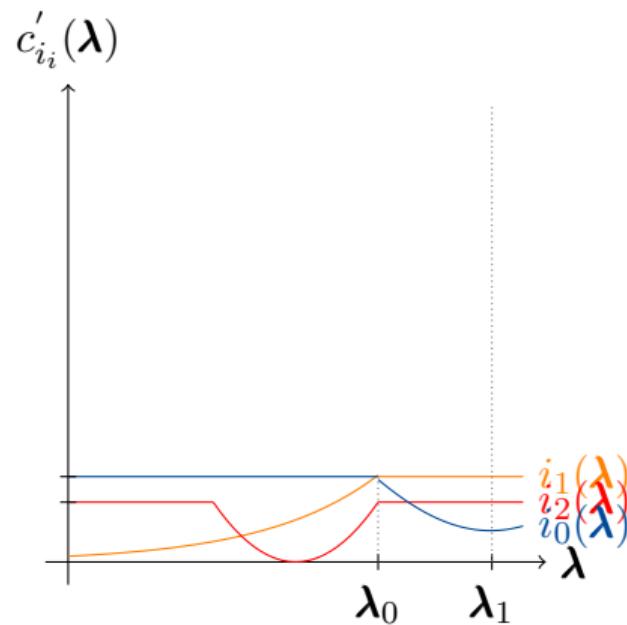


Search initial well performing configuration. Add λ_0 to \mathbf{P}

Hydra: Iteration 1

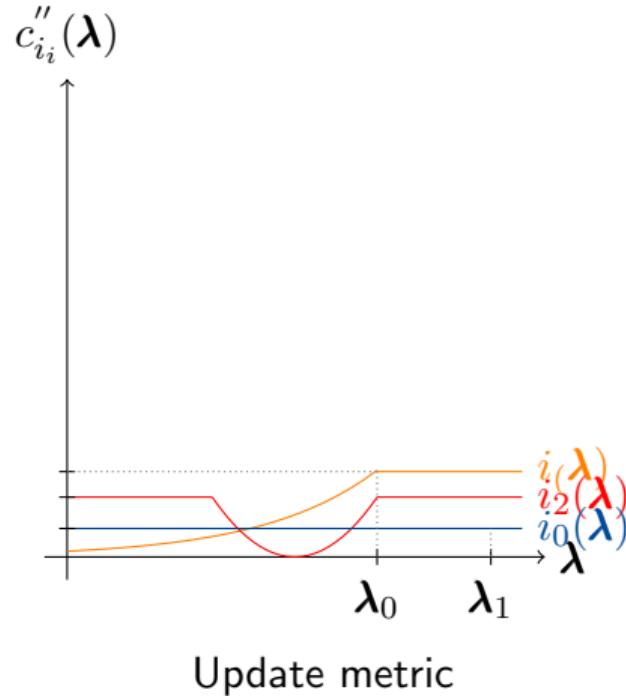


Hydra: Iteration 2



Search well performing configuration complementary to \mathbf{P} .
Add λ_1 to \mathbf{P} .

Hydra: Iteration 2



Idea

- Optimize a schedule of configurations with algorithm configuration

Idea

- Optimize a schedule of configurations with algorithm configuration

Approach

- Iteratively add a configuration with a time slot t to a schedule $\mathcal{S} \oplus \langle \lambda, t \rangle$

Idea

- Optimize a schedule of configurations with algorithm configuration

Approach

- Iteratively add a configuration with a time slot t to a schedule $\mathcal{S} \oplus \langle \lambda, t \rangle$
- In each iteration, only optimize on instances not solved so far
- The time slot is a further parameter in the configuration space

Idea

- Optimize a schedule of configurations with algorithm configuration

Approach

- Iteratively add a configuration with a time slot t to a schedule $\mathcal{S} \oplus \langle \boldsymbol{\lambda}, t \rangle$
- In each iteration, only optimize on instances not solved so far
- The time slot is a further parameter in the configuration space
- Optimize marginal contribution per time spent:

$$\frac{c(\mathcal{S}) - c(\mathcal{S} \oplus \langle \boldsymbol{\lambda}, t \rangle)}{t}$$

Submodularity

Observation

- Performance metrics of Hydra and Cedalion are submodular
 - ▶ Family of functions
 - ▶ Adding an element to a set reduces the function value
 - ▶ Diminishing returns: decrease of the value reduction over time

Submodularity

Observation

- Performance metrics of Hydra and Cedalion are submodular
 - ▶ Family of functions
 - ▶ Adding an element to a set reduces the function value
 - ▶ Diminishing returns: decrease of the value reduction over time

Definition (Submodularity of f)

For every $X, Y \subseteq Z$ with $X \subseteq Y$ and every $x \in Z - Y$ we have that
$$f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$$

Submodularity

Observation

- Performance metrics of Hydra and Cedalion are submodular
 - ▶ Family of functions
 - ▶ Adding an element to a set reduces the function value
 - ▶ Diminishing returns: decrease of the value reduction over time

Definition (Submodularity of f)

For every $X, Y \subseteq Z$ with $X \subseteq Y$ and every $x \in Z - Y$ we have that
$$f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$$

Advantage

We can bound the error of the portfolio/schedule:

At most away from optimum by factor of 0.63 (see [Streeter and Golovin. 2008])

Dynamic Instance Grouping [Liu et al. 2018]

Idea

- Similar to ISAC: group instances into clusters
- Similar to Hydra: refine clusters and configurations over several iterations

Dynamic Instance Grouping [Liu et al. 2018]

Idea

- Similar to ISAC: group instances into clusters
- Similar to Hydra: refine clusters and configurations over several iterations

Main Idea

- ① Group instances randomly into clusters
- ② run AC on each cluster
- ③ Update clusters based on performance (estimates)
- ④ Go to 2. if budget is not empty
- ⑤ Consider all configurations ever found to create final portfolio

Dynamic Instance Grouping [Liu et al. 2018]

Idea

- Similar to ISAC: group instances into clusters
- Similar to Hydra: refine clusters and configurations over several iterations

Main Idea

- ① Group instances randomly into clusters
 - ② run AC on each cluster
 - ③ Update clusters based on performance (estimates)
 - ④ Go to 2. if budget is not empty
 - ⑤ Consider all configurations ever found to create final portfolio
- increase the configuration budget in each iteration
 - ▶ first clusterings will have a poor quality → small configuration time
 - ▶ later clusterings will be better → more configuration time

AutoML: Hyperparameter Optimization

Wrap Up

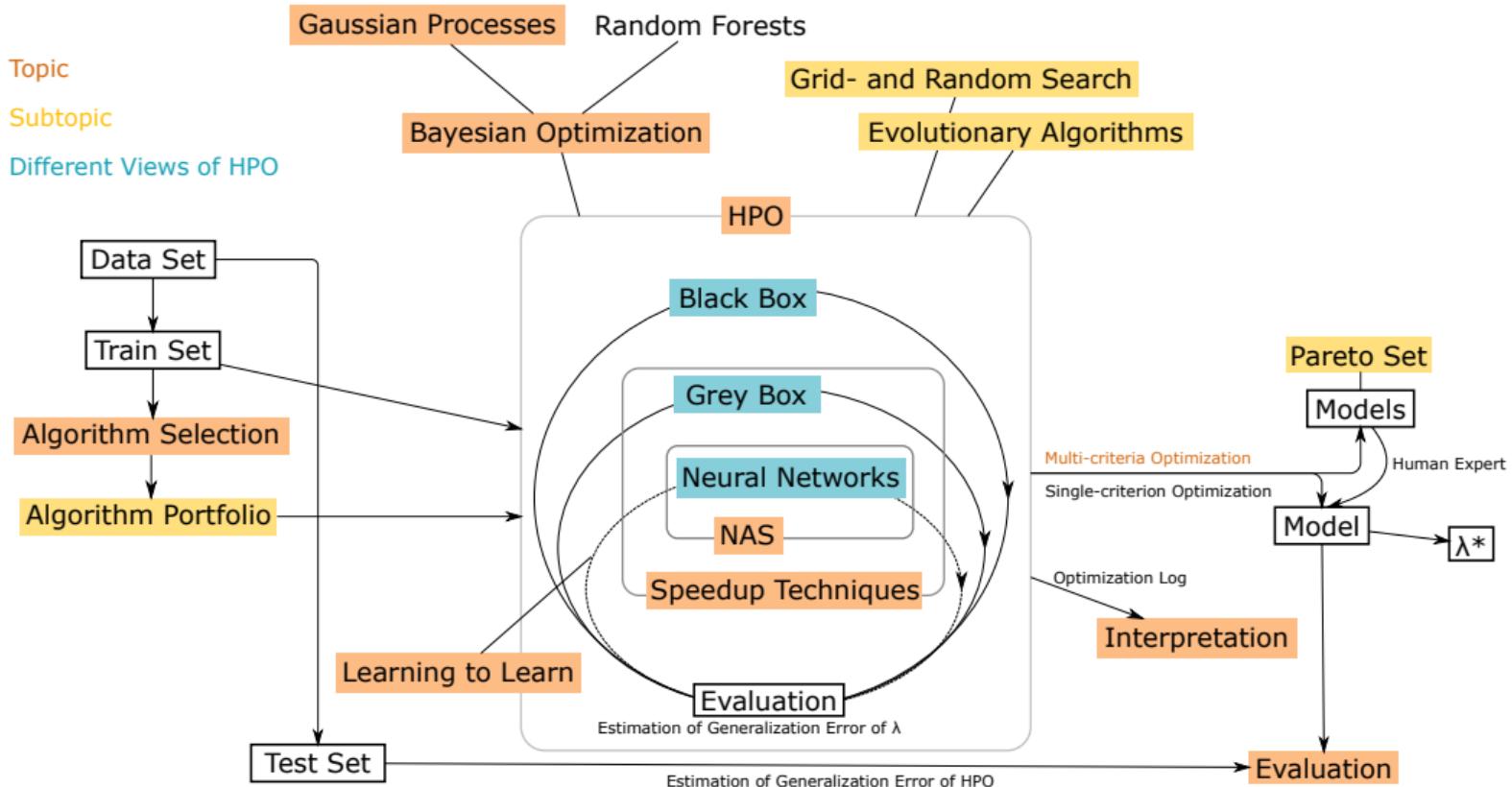
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Jakob Richter

From HPO to AutoML

So far we covered

- Mechanisms to select ML algorithms for a data set (algorithm selection)
- HPO as black-box optimization
 - ▶ Grid- and random search, EAs, BO
- HPO as a grey box problem
 - ▶ Hyperband, BOHB
- Neural Architecture Search (NAS)
 - ▶ One-Shot approaches, DART
- Dynamic algorithm configuration (learning to learn)
 - ▶ Adapt configuration during training

From HPO to AutoML



Lecture Overview

- 1 The Missing Building Blocks
- 2 Common Preprocessing Steps
- 3 Practical Problems
- 4 Combined Preprocessing and Model Building: Pipelining

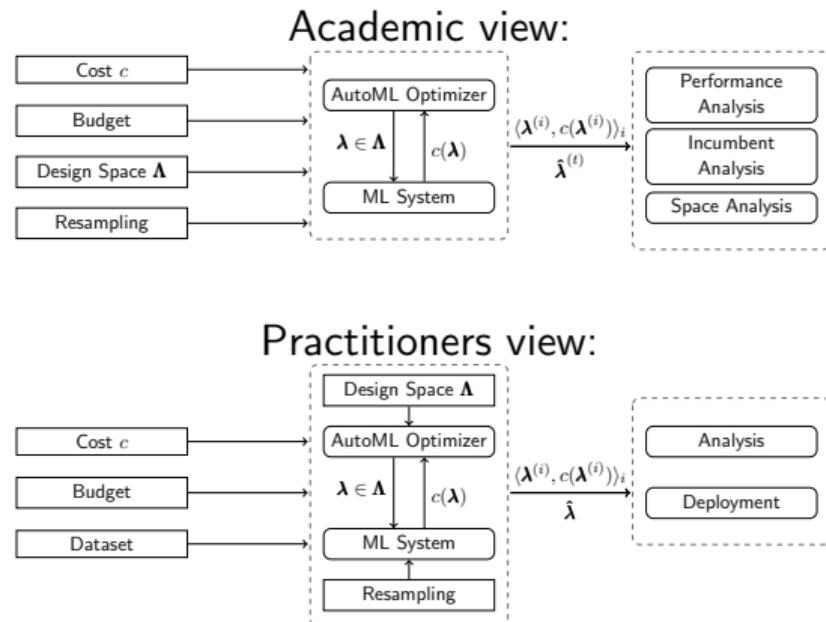
What is missing?

What do I need to know as an AutoML user?

- ~~Nothing, because it is automatic.~~
- Understand limitations of AutoML and framework.
- Know how to interpret the results.
- Maybe: Preprocessing and feature extraction.

Ingredients to implement an AutoML?

- HPO algorithm
- ML / Pipeline framework
- Parallelization / Multifidelity
- Process encapsulation and time capping



Choice of Learning Algorithm

- A plethora of learners exists, for different data sets different models are likely needed.
- Studies [Fernandez-Delgado et al. 2014] and experience show:
One these is often good – on tabular data:
 - ▶ penalized regression
 - ▶ SVM
 - ▶ gradient boosting
 - ▶ random forests
 - ▶ (neural networks)
- Example: Auto-Sklearn 2.0 [Feurer et al. 2020] uses:
 - ▶ extra trees
 - ▶ gradient boosting
 - ▶ passive aggressive
 - ▶ random forest
 - ▶ linear model

Choice of Search Space for a Learning Algorithm

Ranges often selected based on experience

- See other AutoML frameworks: e.g. Auto-Sklearn 2.0 [Feurer et al. 2020]
- Sensitivity analysis often does not exist for ML algorithms
- Check literature on specific ML algorithm

Options for automation:

- Use huge search space to cover all possibilities
(combine with meta-learning for good initial design for Bayesian optimization)
- Use results of meta-experiments to obtain smaller search space that is estimated to work well.

| Algorithm | Hyperparameter | Type | Lower | Upper | Trafo |
|---------------------------------|---------------------------|----------|-----------|-----------|-------------|
| glmnet
(Elastic net) | alpha | numeric | 0 | 1 | - |
| | lambda | numeric | -10 | 10 | 2^x |
| rpart
(Decision tree) | cp | numeric | 0 | 1 | - |
| | maxdepth | integer | 1 | 30 | - |
| | minbucket | integer | 1 | 60 | - |
| | minsplit | integer | 1 | 60 | - |
| knn
(k-nearest neighbor) | k | integer | 1 | 30 | - |
| | metric | string | euclidean | manhattan | - |
| svm
(Support vector machine) | kernel | discrete | - | - | - |
| | cost | numeric | -10 | 10 | 2^x |
| | gamma | numeric | -10 | 10 | 2^x |
| | degree | integer | 2 | 5 | - |
| ranger
(Random forest) | num.trees | integer | 1 | 2000 | - |
| | replace | logical | - | - | - |
| | sample.fraction | numeric | 0.1 | 1 | - |
| | mtry | numeric | 0 | 1 | $x \cdot p$ |
| | respect.unordered.factors | logical | - | - | - |
| | min.node.size | numeric | 0 | 1 | n^x |
| xgboost
(Gradient boosting) | nrounds | integer | 1 | 5000 | - |
| | eta | numeric | -10 | 0 | 2^x |
| | subsample | numeric | 0.1 | 1 | - |
| | booster | discrete | - | - | - |
| | max_depth | integer | 1 | 15 | - |
| | min_child_weight | numeric | 0 | 7 | 2^x |
| | colsample_bytree | numeric | 0 | 1 | - |
| | colsample_bylevel | numeric | 0 | 1 | - |
| | lambda | numeric | -10 | 10 | 2^x |
| | alpha | numeric | -10 | 10 | 2^x |

Taken from [Probst et al. 2019].

Choice of Search Space for a Learning Algorithm

Ranges often selected based on experience

- See other AutoML frameworks: e.g. Auto-Sklearn 2.0 [Feurer et al. 2020]
- Sensitivity analysis often does not exist for ML algorithms
- Check literature on specific ML algorithm

Options for automation:

- Use huge search space to cover all possibilities (combine with meta-learning for good initial design for Bayesian optimization)
- Use results of meta-experiments to obtain smaller search space that is estimated to work well.

| Parameter | Def.P | Def.O | Tun.P | Tun.O | $q_{0.05}$ | $q_{0.95}$ |
|---------------------------|------------|-----------------|-------|-------|------------|------------|
| glmnet | | | 0.069 | 0.024 | | |
| alpha | 1 | 0.403 | 0.038 | 0.006 | 0.009 | 0.981 |
| lambda | 0 | 0.004 | 0.034 | 0.021 | 0.001 | 0.147 |
| rpart | | | 0.038 | 0.012 | | |
| cp | 0.01 | 0 | 0.025 | 0.002 | 0 | 0.008 |
| maxdepth | 30 | 21 | 0.004 | 0.002 | 12.1 | 27 |
| minbucket | 7 | 12 | 0.005 | 0.006 | 3.85 | 41.6 |
| minsplit | 20 | 24 | 0.004 | 0.004 | 5 | 49.15 |
| knn | | | 0.031 | 0.006 | | |
| svm | | | 0.056 | 0.042 | | |
| k | 7 | 30 | 0.031 | 0.006 | 9.95 | 30 |
| kernel | radial | radial | 0.030 | 0.024 | | |
| cost | 1 | 682.478 | 0.016 | 0.006 | 0.002 | 920.582 |
| gamma | $1/p$ | 0.005 | 0.030 | 0.022 | 0.003 | 18.195 |
| degree | 3 | 3 | 0.008 | 0.014 | 2 | 4 |
| ranger | | | 0.010 | 0.006 | | |
| num.trees | 500 | 983 | 0.001 | 0.001 | 206.35 | 1740.15 |
| replace | TRUE | FALSE | 0.002 | 0.001 | | |
| sample.fraction | 1 | 0.703 | 0.004 | 0.002 | 0.323 | 0.974 |
| mtry | \sqrt{p} | $p \cdot 0.257$ | 0.006 | 0.003 | 0.035 | 0.692 |
| respect.unordered.factors | TRUE | FALSE | 0.000 | 0.000 | | |
| min.node.size | 1 | 1 | 0.001 | 0.001 | 0.007 | 0.513 |
| xgboost | | | 0.043 | 0.014 | | |
| nrounds | 500 | 4168 | 0.004 | 0.002 | 920.7 | 4550.95 |
| eta | 0.3 | 0.018 | 0.006 | 0.005 | 0.002 | 0.355 |
| subsample | 1 | 0.839 | 0.004 | 0.002 | 0.545 | 0.958 |
| booster | gbtree | gbtree | 0.015 | 0.008 | | |
| max_depth | 6 | 13 | 0.001 | 0.001 | 5.6 | 14 |
| min_child_weight | 1 | 2.06 | 0.008 | 0.002 | 1.295 | 6.984 |
| colsample_bytree | 1 | 0.752 | 0.006 | 0.001 | 0.419 | 0.864 |
| colsample_bylevel | 1 | 0.585 | 0.008 | 0.001 | 0.335 | 0.886 |
| lambda | 1 | 0.982 | 0.003 | 0.002 | 0.008 | 29.755 |
| alpha | 1 | 1.113 | 0.003 | 0.002 | 0.002 | 6.105 |

Table 3: Defaults (package defaults (Def.P) and optimal defaults (Def.O)), tunability of the hyperparameters with the package defaults (Tun.P) and our optimal defaults (Tun.O) as reference and tuning space quantiles ($q_{0.05}$ and $q_{0.95}$) for different parameters of the algorithms.

Taken from [Probst et al. 2019].

Choice of Resampling Strategy

For computation of generalization error / cost:

$$c(\boldsymbol{\lambda}) = \frac{1}{k} \sum_{i=1}^k \widehat{GE}_{\mathcal{D}_{\text{val}}^i} (\mathcal{I}(\mathcal{D}_{\text{train}}^i, \boldsymbol{\lambda}))$$

Rules of thumb:

- Default: 10-fold CV ($k = 10$)
- Huge datasets: holdout
- Tiny datasets: 10x10 repeated CV
- Stratification for imbalanced classes

Watch out for this:

- Small sample size situation because of imbalances
- Leave-one-object out
- Time dependencies
- A good AutoML system should let you customize resampling
- Meta-learn good resampling strategy [Feurer et al. 2020]

Choice of Optimization Algorithm

Choose optimization algorithm based on ...

- complexity of search space / budget
- time-costs of evaluations

Complex search space

→ EA with exploratory character, TPE, BO with RF

Numerical (lower-dim) search space and tight budget

→ BO with GP

Expensive evals

→ Hyperband, BOHB

Deep learning

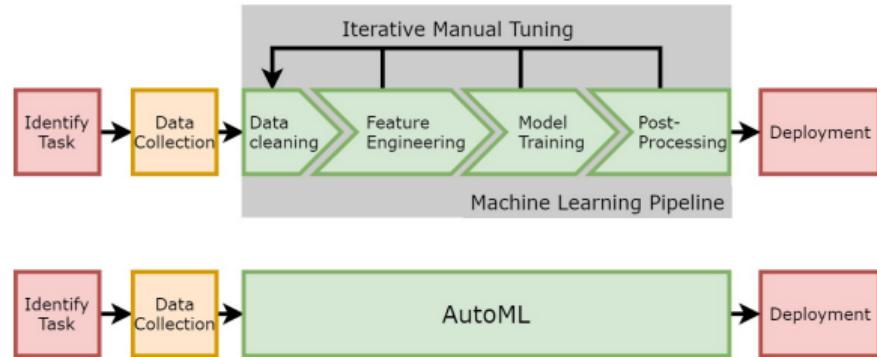
→ Parametrize architectures, then HPO, see above

→ NAS

Preprocessing

Ideal AutoML systems should also optimize:

- ✗ Data preprocessing
- ✗ Feature engineering
- ✗ Feature selection
- ✓ Model training



Lecture Overview

- 1 The Missing Building Blocks
- 2 Common Preprocessing Steps
- 3 Practical Problems
- 4 Combined Preprocessing and Model Building: Pipelining

Preprocessing capabilities differ heavily

| Tool | Platform | Input data sources | Data pre-processing | Data types detected | Feature engineering | ML Tasks | Model selection and Hyperparameter optimization | Quick start / early stop | Model evaluation / Result analysis/ Visualization |
|-----------------------|-----------------|---|---------------------|--|---|--|---|--|--|
| | | Sparsified/flat datasets
Image, text | | Numerical
Categorical
Datetime
Time-series
Other (Hierarchical types) (P') | Datetime, categorical processing
Imbalance, missing values
Feature selection, induction
Advanced feature extraction (R') | Supervised learning (B')
Unsupervised learning (T')
Ensemble | Genetic algorithm
Bayesian search
Random search | Neural architecture search
Quick finding of starting model
Allow maximum limit search time
Restricted firm consuming combination of components
Model dashboard | Feature importance
Model explainability and interpretation, and reason code (T17) |
| TransmogrifAI | Apache Spark | Y | N | Y(γ) | Y Y Y Y Y Y Y Y Y Y | | | | |
| H2O-AutoML | H2O clusters | Y | N | Y | Y Y Y Y N Y Y Y Y | | | | |
| Darwin (+) | GCP | Y | N | Y | Y Y Y Y N Y Y Y Y | | | | |
| DataRobot (+) | Datarobot & AWS | Y | Y | Y | Y Y N Y N Y Y Y Y | | | | |
| Google AutoML (+) | Google Cloud | N | Y | Y | | N Y Y Y Y Y Y Y Y | | | |
| Auto-sklearn | | Y | N | N | N N N N N N Y(β) | Y Y Y Y Y N Y N Y Y | Y Y Y Y Y Y Y Y Y Y | Y Y Y Y Y Y Y Y Y Y | |
| MLjar (+) | MLJAR Cloud | Y(β) | N | Y | Y Y N N N N Y | Y(β) N N Y(β) | N Y N Y N N N N N N | Y Y Y Y Y Y Y Y Y Y | Y Y Y Y N |
| Auto_ml | | Y | N | N | N N N N N N Y | Y Y Y Y Y N Y N Y Y | N N N N N N N N N N | Y Y Y Y Y Y Y Y Y Y | |
| TPOT | | Y | N | N | N N N N N N N Y | Y N Y Y Y N Y Y N Y | N N N N N N N N N Y | Y Y Y Y Y Y Y Y Y Y | |
| Auto-keras | | Y | Y | N | N N N N N N N Y | Y Y N Y Y Y Y N Y Y | N N N N Y Y Y Y Y Y | Y Y Y Y Y Y Y Y Y Y | N Y N Y |
| Ludwig | | Y | Y | Y(γ) | Y Y N Y Y N Y Y Y Y | Y Y Y Y Y N Y Y Y Y | Y Y Y Y Y Y Y Y Y Y | Y Y Y Y N Y N Y Y N | |
| Auto-Weka | | Y | N | N | Y Y N N N N N Y | Y Y N Y Y N Y Y N Y | Y Y Y Y Y Y Y Y Y Y | Y Y Y Y Y Y Y Y Y Y | |
| Azure ML (+) | Azure | Y | Y | Y(β) | Y Y Y Y Y N Y Y Y Y | Y N Y N Y N Y N Y N | Y Y Y Y N Y N Y N Y N | Y Y Y Y Y Y Y Y Y Y | |
| Sagemaker (+) | AWS | Y | Y | Y | Y Y Y Y Y N Y Y Y Y | Y N Y Y Y N Y N Y N | Y Y Y Y N Y N Y N Y N | Y N Y Y Y N Y N Y N | |
| H2O-Driverless AI (+) | H2O clusters | Y(β) | Y | Y | Y Y Y Y Y Y Y Y Y Y | Y Y Y Y Y Y Y Y Y Y | Y Y N N N N N N N N | Y Y Y Y Y Y N N N N | Y Y Y Y Y |

Taken from [Truong et al. 2019].

Highlighted: Non-commercial AutoML frameworks

- Auto-detection of feature types: some
 - Preprocess categoricals: some
 - Imputation: all
 - Class imbalance handling: all

Fig. 2. Comparison table of functionality for AutoML tools. (+): commercialized tools; (*): the function is not very stable, it fails for some datasets; (2*): categorical input must be converted into integers; (3*): datasets have to include headers; (4*): missing values must be represented as NA; (5*): multiclass classification not provided; (6*): need some users' input for dataset description such as column types; (7*): ability to detect primitive data types and rich data types such as: text (id, url, phone), numerical (integer, real); (8*): advanced feature processing: bucketing of values, removing features with zero variance or features with drift over time; (9+): supervised learning includes binary classification, multiclass classification, regression; (10+): unsupervised learning includes clustering and anomaly detection; (11+): model interpretation and explainability refers to techniques such as LIME, Shapley, Decision Tree Surrogate, Partial Dependence, Individual Conditional Expectation, Lift chart, feature fit, prediction distribution plot; accuracy over time, hot spot and reason codes; In a few empty cells, is not clear that the functionality is provided from documentation of the tools, to the best of our knowledge.

Cleaning

Data cleaning is hard to fully automate, but a few heuristics exist:

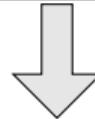
- Remove ID columns, columns with mostly unique values.
- Outlier detection
- Detect time series or spatial data → at the very least, evaluation and resampling needs to be adapted, but feature extraction likely as well

It is highly unclear how much of this is required input by the user (last point is more or less task specification), and what should be automated by the system.

Categorical Features: Dummy Encoding

- Most simple encoding
- Works well with low cardinality categoricals

| SalePrice | Central.Air | Bldg.Type |
|-----------|-------------|-----------|
| 189900 | Y | 1Fam |
| 195500 | Y | 1Fam |
| 213500 | Y | TwnhsE |
| 191500 | Y | TwnhsE |
| 236500 | Y | TwnhsE |



| SalePrice | Y | Bldg.Type.2fmCon | Bldg.Type.Duplex | Bldg.Type.Twnhs | Bldg.Type.TwnhsE |
|-----------|---|------------------|------------------|-----------------|------------------|
| 189900 | 1 | 0 | 0 | 0 | 0 |
| 195500 | 1 | 0 | 0 | 0 | 0 |
| 213500 | 1 | 0 | 0 | 0 | 1 |
| 191500 | 1 | 0 | 0 | 0 | 1 |
| 236500 | 1 | 0 | 0 | 0 | 1 |

Categorical Features: Target / Impact Encoding

- Well known from CART
- Handles high cardinality categoricals, too

Goal: Encodes each categorical x as a single numeric \tilde{x}

Regression: $\text{Impact}(x) = \mathbb{E}(y|x) - \mathbb{E}(y)$

Classification: $\text{Impact}(x) = \text{logit}(P(y = \text{target}|x)) - \text{logit}(P(y = \text{target}))$

- Needs regularization (through CV) to prevent target leakage
[Zumel et al. 2019]
- Advantage: Handles unknown categorical levels on test data

Alternatives:

- factorization machines
- clustering feature levels
- feature hashing

| $\mathcal{D}_{\text{train}}$ | |
|------------------------------|-----|
| x | y |
| A | 10 |
| A | 20 |
| B | 30 |

Encoding

| x | \tilde{x} |
|-----|-------------|
| A | -5 |
| B | 10 |

| $\mathcal{D}_{\text{test}}$ | |
|-----------------------------|-------------|
| x | \tilde{x} |
| A | -5 |
| B | 10 |
| C | 0 |

Common Preprocessing Steps: Missing Values

- Replace missings with imputed values, try not to disturb distribution too much
- Examples: mean, median, mode, sample from histogram, predict value based on other features
- Additional factor column indicating missingness can help if NA-state carries information for target
- *Out-Of-Range* works often well for tree-based techniques (RF and boosting!), saves extra missingness col
- For inference, simple imputation is often shunned, and multiple, model-based imputation recommended; for prediction naive imputation often works remarkably well

The diagram illustrates the process of generating a modified training set $\tilde{\mathcal{D}}_{\text{train}}$ from the original training set $\mathcal{D}_{\text{train}}$. The original data $\mathcal{D}_{\text{train}}$ is shown as:

| | x_1 | x_2 |
|----|-------|-------|
| 10 | 3 | |
| NA | | 2 |
| 5 | | NA |

Two arrows point from the missing values (NA) in the $\mathcal{D}_{\text{train}}$ table to the corresponding rows in the $\tilde{\mathcal{D}}_{\text{train}}$ table. The resulting modified dataset $\tilde{\mathcal{D}}_{\text{train}}$ is:

| | x_1 | x_2 | Feature | NA |
|----|-------|-------|---------|-----|
| 10 | 3 | | x_1 | 100 |
| NA | | 2 | x_2 | 30 |

Below this, the modified dataset $\tilde{\mathcal{D}}_{\text{train}}$ is shown again with its own header:

| | x_1 | x_2 |
|-----|-------|-------|
| 10 | 3 | |
| 100 | | 2 |
| 5 | | 30 |

Out of range imputation

Feature Selection

- Filter; for ultra-higdim tasks
- Stepwise / wrapper methods; seldom increases performance when well-regularized learners are used, but quite expensive
- Embedded: CART, lasso, ...
- Selection interesting when predictive performance vs. sparseness should be optimized → multi-criteria optimization
- Combined feature selection and HPO: [Binder et al. 2020]

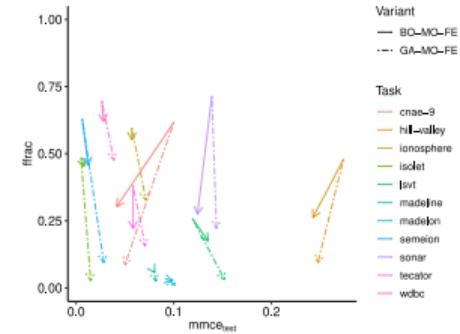
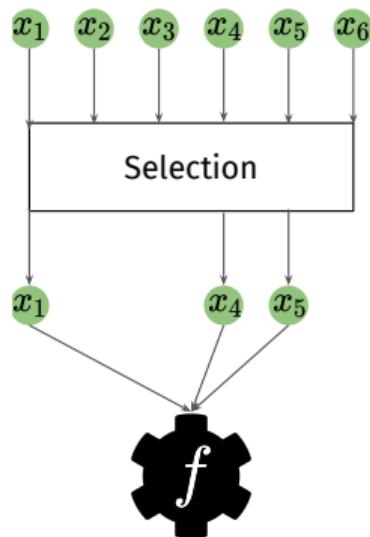


Figure 3: Comparison of multi-objective and single-objective methods applied to the SVM learning algorithm: Performance mmcetest and the fraction of included features ffrac found after 2000 evaluations by baseline BO-SO (tail end of arrows), BO-MO-FE (head of solid arrows) and GA-MO-FE (head of dashed arrows). Each dataset (Table 1) is shown, values are averaged over 10 outer CV runs. Choice of individuals for MO methods described in Section 6.2.

Taken from [Binder et al. 2020].

Common Feature Construction Methods

Reduction:

- PCA, ICA, autoencoder

Feature extraction:

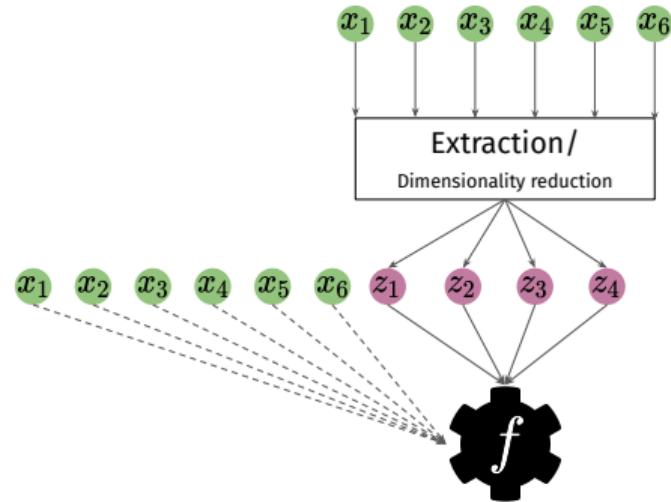
- Polynomial features: $x_j \rightarrow x_j, x_j^2, x_j^3, \dots$
- Interactions: $x_j, x_k \rightarrow x_j, x_k, x_j \cdot x_k$

Feature generation:

- Transform to “circular” features (month, day)
e.g. $\tilde{x}_1 = \sin(2\pi \cdot x/24)$ and $\tilde{x}_2 = \cos(2\pi \cdot x/24)$

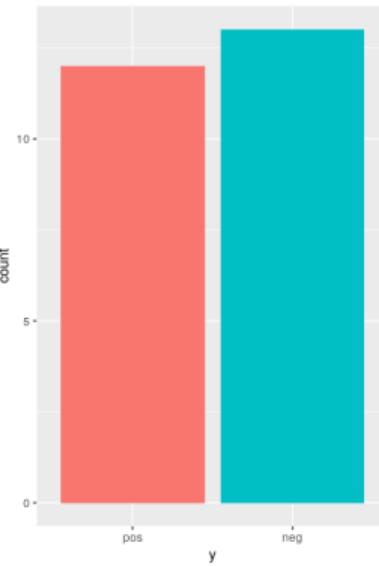
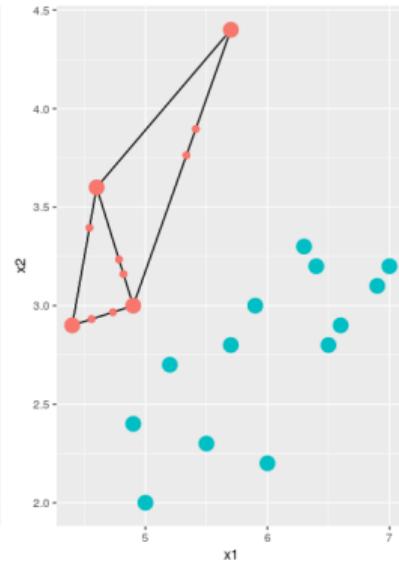
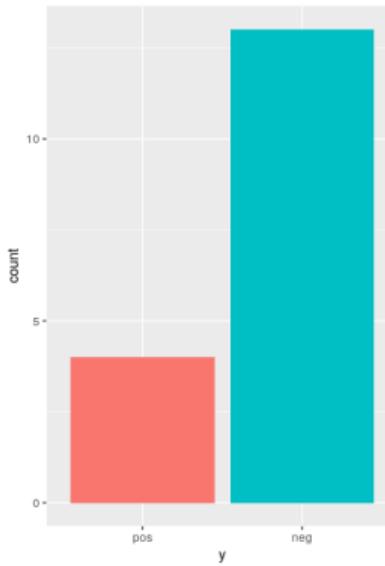
Combine with external data:

- names \rightarrow gender, ethnicity, age
- home address \rightarrow household income
- location + date \rightarrow weather



Imbalanced Classes

- Oversampling of minority class
- Seldom: undersampling of majority class
- Intelligent oversampling strategies which create synthetic observations (SMOTE)



Lecture Overview

- 1 The Missing Building Blocks
- 2 Common Preprocessing Steps
- 3 Practical Problems
- 4 Combined Preprocessing and Model Building: Pipelining

Practical Problems: Stability

AutoML system should:

- never fail to return a result
- terminate within a given time
- save intermediate results and allow to continue

Failure points:

- optimizer can crash (e.g. GP in BO)
- training can crash (e.g. segfault)
- training can run "forever"

Ways out

- Encapsulate train/predict in separate process from HPO
- Ressource limit time and memory of that process by OS
- If learner crashes, run robust fallback (constant predictor)
- Use "robust" HPO, run random config as last resort if proposal fails (exploration)

Practical Problems: Parallelization

Parallelization should allow:

- multiple cores
- multiple nodes

Possible parallelization levels:

- training of learner (threading / GPU)
- resampling
- eval configurations (batch proposal of HPO)

Possible problems:

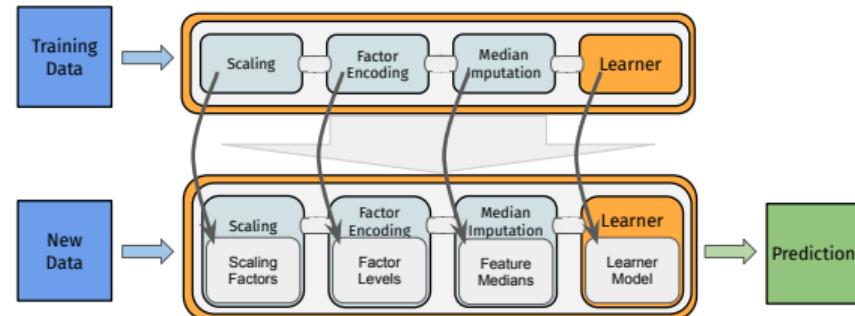
- Sequential nature of HPO algorithms (e.g. BO)
- Heterogeneous runtimes cause idling → asynchronous HPO attractive, but more complex
- Main memory or CPU-cache becomes bottleneck
- Communication between workers

Lecture Overview

- 1 The Missing Building Blocks
- 2 Common Preprocessing Steps
- 3 Practical Problems
- 4 Combined Preprocessing and Model Building: Pipelining

Linear Pipelining

- Applying preprocessing to the whole dataset leads to data leakage
- Preprocessing should have train and predict steps, too
- Can add it to learner, and embed it in CV
- Surprise: Preproc has hyperparameters
- Optimize pipeline jointly:
 $\Lambda = \Lambda_{\text{preproc}} \times \Lambda_{\mathcal{I}}$
- Still HPO, not much different than for single learner
- Λ "simply" of higher dimension



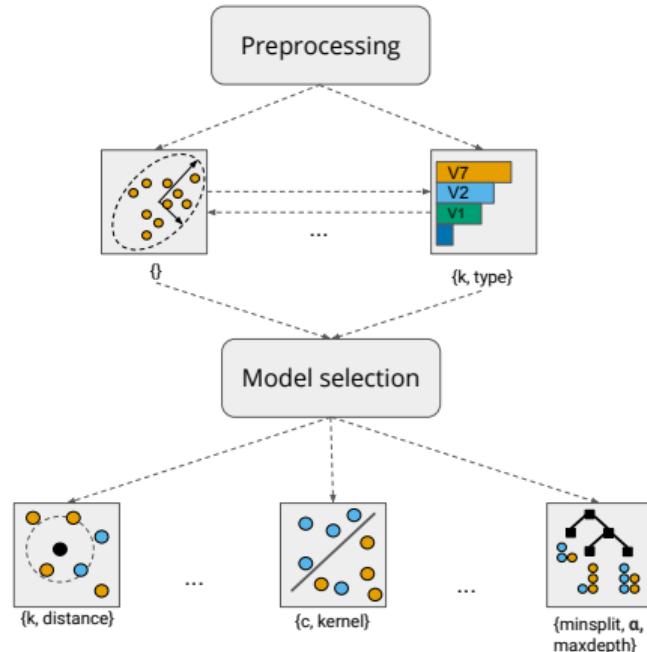
Nonlinear Pipelines

Ideal to let HPO choose automatically:

- preprocessing
 - feature extraction
 - learner
- Λ becomes hierarchical search space!

Suitable optimizers:

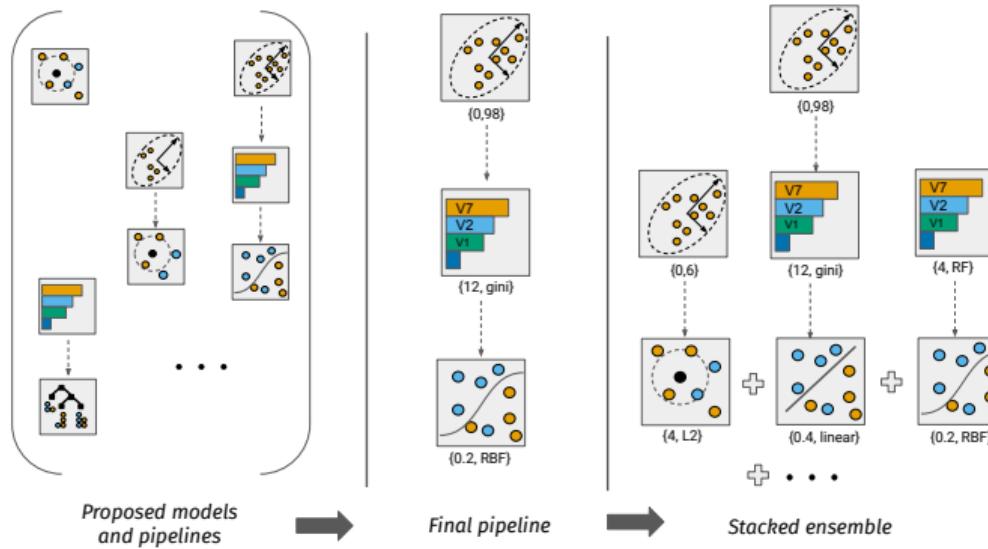
- TPE
- Random search, hyperband with sampler that follows the hierarchy
- BO with RF (imputation or hierarchical trees)
- Evolutionary approaches (similar to NAS)



Obtaining Final Model

Options:

- Choose the optimal path as linear pipeline.
- Build ensemble of best configurations
(e.g. [Feurer et al. 2015], [LeDell and Poirier. 2020]).



Current Benchmark on Tabular Data

| Framework:
Binary tasks: | auto-sklearn | Auto-WEKA | H2O | AutoML | RandomForest | TPOT |
|-----------------------------|--------------|-----------|-------|--------|--------------|------|
| adult | 1.045 | 1.000 | 1.049 | 1.000 | 1.048 | |
| airlines | 1.403 | 1.016 | 1.435 | 0.997 | 1.343 | |
| albert | 1.009 | | 1.115 | 1.001 | 0.981 | |
| amazon_employee... | 0.972* | 0.886 | 1.048 | 1.003 | 1.012 | |
| aps_failure | 1.000 | 0.985 | 1.001 | 1.000 | 1.001 | |
| australian | 1.010 | 1.015 | 0.909 | 1.010 | 1.011 | |
| bank_marketing | 1.012 | 0.950 | 1.015 | 1.000 | 1.008 | |
| blood_transfusion | 1.495 | 1.379 | 1.532 | 0.985 | 1.149 | |
| christine | 1.072 | 0.998 | 1.048 | 0.988 | 1.029 | |
| credit-g | 0.970* | 0.829 | 0.991 | 1.004 | 0.924 | |
| guillermo | 1.004 | 0.934 | 1.024 | 0.999 | 0.878 | |
| higgs | 1.018* | 0.845 | 1.041 | 0.999 | 1.005 | |
| jasmine | 0.987 | 0.939 | 1.001 | 0.998 | 1.004 | |
| kcl | 0.999* | 0.934 | 0.992 | 0.987 | 1.013 | |
| kddcup09_appetency | 1.181* | 1.043 | 1.176 | 1.016 | 1.134 | |
| kr-vs-kp | 1.000* | 0.959 | 1.000 | 0.999 | 0.999 | |
| miniBoone | 1.008 | 0.957 | 1.010 | 0.999 | 1.001 | |
| nomao | 1.002 | 0.973 | 1.002 | 1.000 | 1.001 | |
| numerai28.6 | 1.679 | 1.544 | 1.730 | 1.042 | 1.428 | |
| phoneme | 0.993* | 0.998 | 1.005 | 1.000 | 1.015 | |
| riccardo | 1.000 | 0.996 | 1.000 | 0.999 | 0.992 | |
| sylvine | 1.013 | 0.985 | 1.011 | 0.999 | 1.023 | |
| Multiclass tasks: | | | | | | |
| car | 1.030 | 0.906 | 1.060 | 0.878 | 1.060 | |
| cnae-9 | 1.069 | 0.541 | 1.076 | 0.999 | 1.057 | |
| connect-4 | 1.184 | -1.565 | 1.409 | 0.954 | 1.276 | |
| covtype | 0.976 | -0.361 | 0.856 | 0.944 | 0.933 | |
| dlibert | 1.182 | 0.459 | 1.205 | 0.979 | 1.111 | |
| dionsis | 0.580 | 0.590 | | 1.002 | | |
| fabert | 1.026 | -5.235 | 1.049 | 1.004 | 1.005 | |
| fashion-mnist | 0.995 | 0.717 | 1.052 | 0.993 | 0.841 | |
| helena | 0.660 | -18.420 | 1.905 | 0.970 | 1.676 | |
| jannis | 1.083 | -1.989 | 1.065 | 0.973 | 0.987 | |
| jungle_chess... | 1.299 | -3.309 | 1.235 | 0.933 | 1.459 | |
| mfeat-factors | 1.059* | 0.789 | 1.053 | 0.992 | 1.018 | |
| robert | -0.001 | | 1.545 | 1.000 | 0.640 | |
| segment | 1.004 | 0.808 | 1.012 | 0.992 | 1.008 | |
| shuttle | 1.000 | 0.979 | 1.000 | 1.000 | 1.000 | |
| vehicle | 1.102 | -4.630 | 1.166 | 0.986 | 1.099 | |
| volkert | 1.002 | -5.585 | 1.111 | 0.954 | 0.945 | |

Table taken from [Gijsbers et al. 2019].

- On some datasets AutoML yields big performance improvements
- On many datasets RF is equally good
- Need more and diverse benchmarks

Table 2: Performance of AutoML frameworks, scaled between a constant class prior predictor (=0) and a tuned random forest (=1). Missing values mean that no results were returned in time. *: the task was also included in meta-learning models.