

Lecture 9: Algorithm Independent Principles - III

Hyperparameter Optimization, AutoML

Machine Learning, Summer Term 2019

Michael Tangermann Frank Hutter Marius Lindauer

University of Freiburg

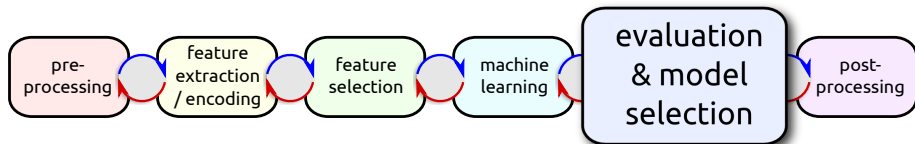


Lecture Overview

- 1 Hyperparameter Optimization
- 2 Automated Machine Learning (AutoML)
- 3 Grey-Box AutoML

- 1 Hyperparameter Optimization
- 2 Automated Machine Learning (AutoML)
- 3 Grey-Box AutoML

Hyperparameter Optimization in the ML Design Cycle



Part of model selection: how to set free algorithm hyperparameters?

- **Hyperparameter optimization** is essentially the same problem as model selection, except:
- Many more possible settings to choose from (infinite/exponential)

Parameters vs. Hyperparameters

- Most machine learning algorithms **internally optimize parameters**
 - E.g., weights in linear regression
 - E.g., split points and leaf predictions in decision trees
 - E.g., deep learning: millions of network weights

Parameters vs. Hyperparameters

- Most machine learning algorithms **internally optimize parameters**
 - E.g., weights in linear regression
 - E.g., split points and leaf predictions in decision trees
 - E.g., deep learning: millions of network weights
- Standard ML approach: **min. training loss + $\lambda \times$ regularization loss**
 - Using standard gradient-based optimizers

Parameters vs. Hyperparameters

- Most machine learning algorithms **internally optimize parameters**
 - E.g., weights in linear regression
 - E.g., split points and leaf predictions in decision trees
 - E.g., deep learning: millions of network weights
- Standard ML approach: **min. training loss + λ \times regularization loss**
 - Using standard gradient-based optimizers
- **Hyperparameters**: decisions left to the algorithm designer
 - How to set λ ?
 - How complex a model to use?
 - How many layers/which structure of deep networks to use?
 - How to set the options of the gradient-based optimizer?



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 scales at least quadratically with the number of samples and may be impractical beyond tens of thousands of samples. For large datasets consider using `sklearn.linear_model.LinearSVC` or `sklearn.linear_model.SGDClassifier` instead, possibly after a `sklearn.kernel_approximation.Nystroem` transformer.

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')

Hyperparameter Optimization Method 1: Random Search

- Select configurations uniformly at random (completely uninformed)
- Global search, won't get stuck in a local region
- Parallelizes nicely and is at least better than grid search:

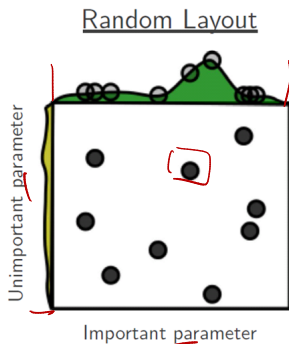
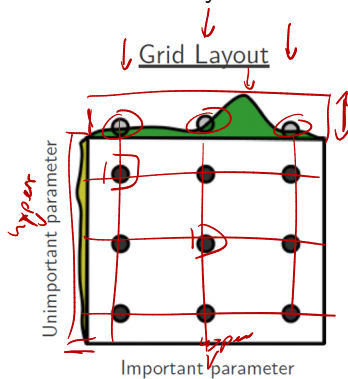


Image source: [Bergstra et al, JMLR 2012]

Hyperparameter Optimization Method 2: Local Search

(also sometimes jokingly called “Graduate Student Descent”)

Start with some configuration θ

repeat

 Modify a single hyperparameter

if *results on benchmark set improve* **then**

 keep new configuration

until *no more improvement possible (or “good enough”)*

⇒ Manually-executed first-improvement local search

Hyperparameter Opt. Method 3: Bayesian Optimization

General approach

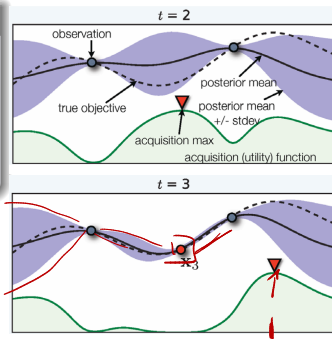
- Fit a probabilistic model to the collected function samples $\langle \theta, f(\theta) \rangle$
- Use the model to guide optimization, trading off exploration vs exploitation

validation loss

Hyperparameter Opt. Method 3: Bayesian Optimization

General approach

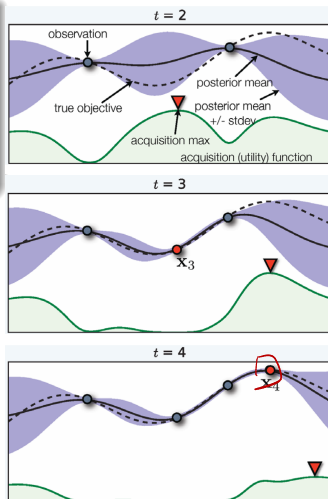
- Fit a probabilistic model to the collected function samples $\langle \theta, f(\theta) \rangle$
- Use the model to guide optimization, trading off exploration vs exploitation



Hyperparameter Opt. Method 3: Bayesian Optimization

General approach

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



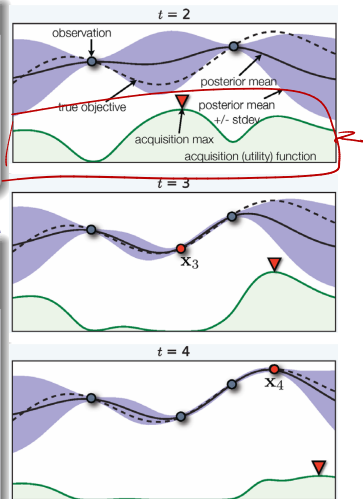
Hyperparameter Opt. Method 3: Bayesian Optimization

General approach

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

Popular approach in the statistics literature since [Mockus, 1978]

- Efficient in # function evaluations
- Works when objective is nonconvex, noisy, has unknown derivatives, etc
- Recent convergence results
[Srinivas et al, 2010; Bull 2011; de Freitas et al, 2012; Kawaguchi et al, 2015]



Bayesian Optimization Algorithm

Algorithm 1: Bayesian Optimization (BO)

Input : Search Space \mathcal{X} , black box function f , acquisition function α ,
maximal number of function evaluations m

1 $\mathcal{D}_0 \leftarrow$ initial_design(\mathcal{X});

Bayesian Optimization Algorithm

Algorithm 2: Bayesian Optimization (BO)

Input : Search Space \mathcal{X} , black box function f , acquisition function α ,
maximal number of function evaluations m

- 1 $\mathcal{D}_0 \leftarrow \text{initial_design}(\mathcal{X});$
- 2 **for** $n = 1, 2, \dots, \underline{m} - |D_0|$ **do**

Bayesian Optimization Algorithm

Algorithm 3: Bayesian Optimization (BO)

Input : Search Space \mathcal{X} , black box function f , acquisition function α , maximal number of function evaluations m

```
1  $\mathcal{D}_0 \leftarrow \text{initial\_design}(\mathcal{X});$   
2 for  $n = 1, 2, \dots, m - |\mathcal{D}_0|$  do  
3    $(\hat{m} : \underline{\theta} \mapsto \underline{y}) \leftarrow \text{fit predictive model on } \mathcal{D}_{n-1};$   
4   select  $\underline{x}_n$  by optimizing  $\underline{x}_n \in \arg \max_{x \in \mathcal{X}} \alpha(x; \mathcal{D}_{n-1}, \hat{m});$ 
```

Bayesian Optimization Algorithm

Algorithm 4: Bayesian Optimization (BO)

Input : Search Space \mathcal{X} , black box function f , acquisition function α , maximal number of function evaluations m

```
1  $\mathcal{D}_0 \leftarrow \text{initial\_design}(\mathcal{X});$   
2 for  $n = 1, 2, \dots, m - |\mathcal{D}_0|$  do  
3    $\hat{m} : \theta \mapsto y \leftarrow \text{fit predictive model on } \mathcal{D}_{n-1};$   
4   select  $x_n$  by optimizing  $x_n \in \arg \max_{x \in \mathcal{X}} \alpha(x; \mathcal{D}_{n-1}, \hat{m});$   
5   Query  $y_n := \underline{f(x_n)};$ 
```

Bayesian Optimization Algorithm

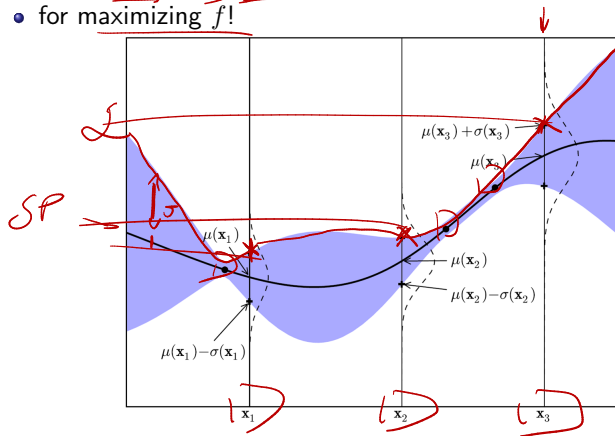
Algorithm 5: Bayesian Optimization (BO)

Input : Search Space \mathcal{X} , black box function f , acquisition function α , maximal number of function evaluations m

1 $\mathcal{D}_0 \leftarrow \text{initial_design}(\mathcal{X});$
2 **for** $n = 1, 2, \dots, m - |D_0|$ **do** δP
3 $\hat{m} : \theta \mapsto y \leftarrow \text{fit } \underline{\text{predictive model}}$ on $\mathcal{D}_{n-1};$
4 select x_n by optimizing $x_n \in \arg \max_{x \in \mathcal{X}} \alpha(x; \mathcal{D}_{n-1}, \hat{m});$
5 Query $y_n := f(x_n);$ \leftarrow
6 Add observation to data $\mathcal{D}_n := \mathcal{D}_{n-1} \cup \{ \langle x_n, y_n \rangle \};$
7 **return** Best x according to \mathcal{D}_m

Acquisition Function: Upper Confidence Bound

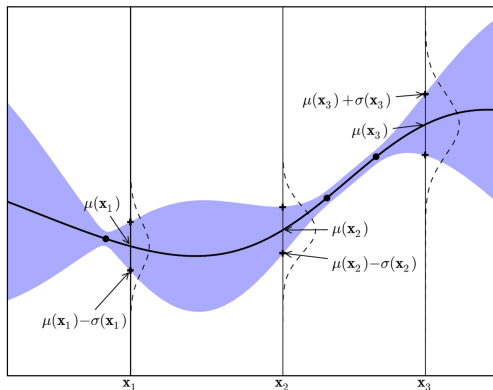
- $UCB(\theta) = \mu(\theta) + \kappa_t \sigma(\theta)$, with exploration parameter κ
 - for maximizing f !



- Which point would we pick next with UCB and $\kappa = 1$? \leftarrow

Acquisition Function: Upper Confidence Bound

- $UCB(\theta) = \mu(\theta) + \kappa_t \sigma(\theta)$, with exploration parameter κ
 - for maximizing f !



- Which point would we pick next with UCB and $\kappa = 1$?
- κ_t increases over time

Range Transformations of Hyperparameters

- Several hyperparameters naturally lay on a logarithmic scale
 - E.g., learning rate between 10^{-5} and 10^0

Range Transformations of Hyperparameters

- Several hyperparameters naturally lay on a **logarithmic scale**
 - E.g., learning rate between 10^{-5} and 10^0
- Transform these to a **linear scale**
 - E.g., instead work with $\log_{10}(\text{lr})$, with values between -5 and 0

Range Transformations of Hyperparameters

- Several hyperparameters naturally lay on a **logarithmic scale**
 - E.g., learning rate between 10^{-5} and 10^0
- Transform these to a **linear scale**
 - E.g., instead work with $\log_{10}(\text{lr})$, with values between -5 and 0
- Good rule of thumb: transform the space to the one you would want to sample from uniformly
 - E.g., uniform sampling from $\underbrace{[10^{-5}, 10^0]}$ would have 90% samples greater than $\underbrace{10^{-1}}$

$[-5, 0]$

Conditional Hyperparameters

- Some hyperparameters h are only active if other hyperparameters p take certain values
 - E.g., weight initialization for layer k is only active if the hyperparameter number of layers is at least k
- We call these hyperparameters **conditional** with **parents** p
- You can always ignore such conditionality, but exploiting it can dramatically simplify the problem

Lecture Overview

- 1 Hyperparameter Optimization
- 2 Automated Machine Learning (AutoML)
- 3 Grey-Box AutoML

Machine Learning is very successful in many applications.

- But it still requires human machine learning experts to
 - Preprocess the data
 - Select / engineer features
 - Select a model family
 - Optimize hyperparameters
 - Construct ensembles
 - ...

Machine Learning is very successful in many applications.

- But it still requires human machine learning experts to
 - Preprocess the data
 - Select / engineer features
 - Select a model family
 - Optimize hyperparameters
 - Construct ensembles
 - ...
- **AutoML**: taking the human expert out of the loop

Machine Learning is very successful in many applications.

- But it still requires human machine learning experts to
 - Preprocess the data
 - Select / engineer features
 - Select a model family
 - Optimize hyperparameters
 - Construct ensembles
 - ...
- **AutoML**: taking the human expert out of the loop
- **Deep learning** helps to automatically learn features
 - But it is even more sensitive to hyperparameters

The AutoML approach introduced by Auto-WEKA [Thornton et al, 2013]

- Expose the choices in a machine learning framework
 - Algorithms, hyperparameters, preprocessors, ...
 - Highly conditional hyperparameter space
 - Combined algorithm selection and hyper-parameter optimization
→ defined in detail on the next slide
- Optimize CV performance using Bayesian optimization
- ⇒ Obtain a true push-button solution for machine learning

The AutoML approach introduced by Auto-WEKA [Thornton et al, 2013]

- Expose the choices in a machine learning framework
 - Algorithms, hyperparameters, preprocessors, ...
 - Highly conditional hyperparameter space
 - Combined algorithm selection and hyper-parameter optimization
→ defined in detail on the next slide
- Optimize CV performance using Bayesian optimization
- Obtain a true push-button solution for machine learning

Extended in Auto-sklearn [Feurer et al, 2015]

Let

- $\mathcal{A} = \underline{A^{(1)}}, \dots, \underline{A^{(J)}}$ be a set of algorithms,
- the hyper-parameters of each algorithm $A^{(j)}$ have domain $\Lambda^{(j)}$,

Let

- $\mathcal{A} = A^{(1)}, \dots, A^{(J)}$ be a set of algorithms,
- the hyper-parameters of each algorithm $A^{(j)}$ have domain $\Lambda^{(j)}$,
- $\underline{D_{train}} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ be a training set which is split into K cross-validation folds $\{D_{\underline{valid}}^{(1)}, \dots, D_{\underline{valid}}^{(K)}\}$ and $D_{train}^{(i)} = D_{train} - D_{\underline{valid}}^{(i)}$ for $i = 1, \dots, K$,

Let

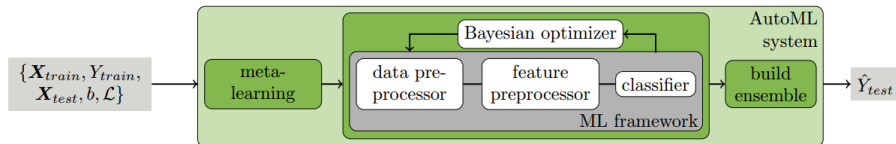
- $\mathcal{A} = A^{(1)}, \dots, A^{(J)}$ be a set of algorithms,
- the hyper-parameters of each algorithm $A^{(j)}$ have domain $\Lambda^{(j)}$,
- $D_{train} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ be a training set which is split into K cross-validation folds $\{D_{valid}^{(1)}, \dots, D_{valid}^{(K)}\}$ and $D_{train}^{(i)} = D_{train} - D_{valid}^{(i)}$ for $i = 1, \dots, K$,
- $\mathcal{L}(A_{\lambda}^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})$ denote the loss of $A_{\lambda}^{(j)}$ trained on $D_{train}^{(i)}$ and evaluated on $D_{valid}^{(i)}$.

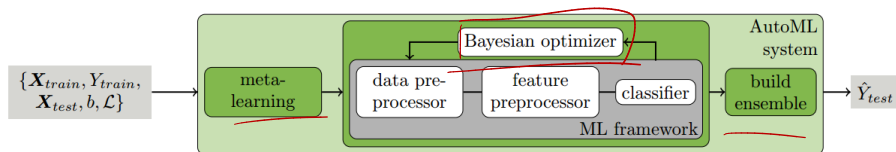
Let

- $\mathcal{A} = A^{(1)}, \dots, A^{(J)}$ be a set of algorithms,
- the hyper-parameters of each algorithm $A^{(j)}$ have domain $\Lambda^{(j)}$,
- $D_{train} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ be a training set which is split into K cross-validation folds $\{D_{valid}^{(1)}, \dots, D_{valid}^{(K)}\}$ and $D_{train}^{(i)} = D_{train} - D_{valid}^{(i)}$ for $i = 1, \dots, K$,
- $\mathcal{L}(A_{\lambda}^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})$ denote the loss of $A_{\lambda}^{(j)}$ trained on $D_{train}^{(i)}$ and evaluated on $D_{valid}^{(i)}$.

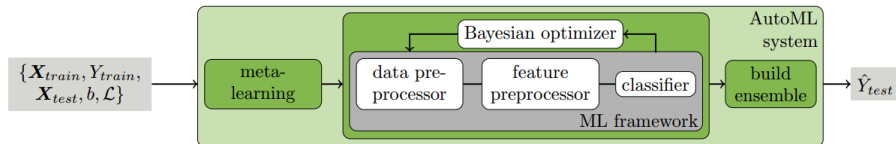
The *CASH* problem is to find the joint algorithm and hyper-parameter setting that minimizes this loss across the K folds:

$$\underline{A^*, \lambda^*} \in \underline{\arg \min_{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}} \frac{1}{K} \sum_{i=1}^K \mathcal{L}(A_{\lambda}^{(j)}, D_{train}^{(i)}, D_{valid}^{(i)})}$$





- Winner of ChaLearn Automatic Machine Learning Challenge
 - Performed better than 150 teams of human experts
 - Searches over ≈ 15 classifiers & preprocessors (110 hyperparameters)



- Winner of ChaLearn Automatic Machine Learning Challenge
 - Performed better than 150 teams of human experts
 - Searches over ≈ 15 classifiers & preprocessors (110 hyperparameters)
- Trivial to use even for novices in machine learning:

```
→ import autosklearn.classification as cls  
→ automl = cls.AutoSklearnClassifier()  
→ automl.fit(X_train, y_train)  
→ y_hat = automl.predict(X_test)
```

Available online: <https://github.com/automl/auto-sklearn>

Lecture Overview

- 1 Hyperparameter Optimization
- 2 Automated Machine Learning (AutoML)
- 3 Grey-Box AutoML

Speeding up Hyperparameter Optimization Methods

- If we use k -fold cross-validation
 - We can reject poor hyperparameter settings after few folds

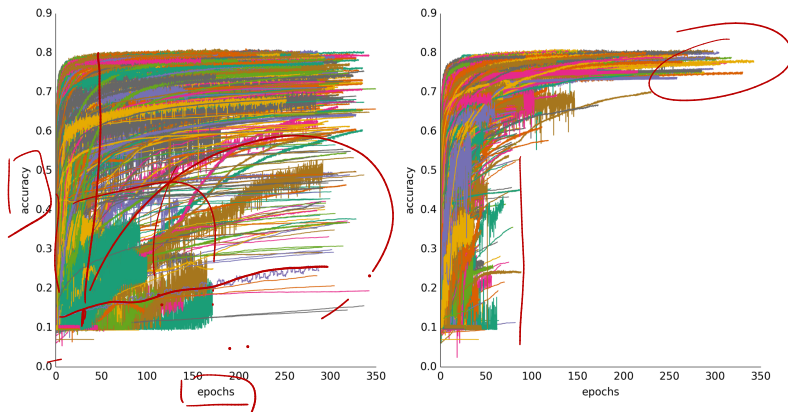
Speeding up Hyperparameter Optimization Methods

- If we use k -fold cross-validation
 - We can **reject poor hyperparameter settings** after few folds
- If we use iterative ML algorithms
 - We can **stop poor runs early**

Speeding up Hyperparameter Optimization Methods

- If we use k -fold cross-validation
 - We can **reject poor hyperparameter settings** after few folds
- If we use iterative ML algorithms
 - We can **stop poor runs early**
- If runs on smaller datasets are faster (almost always)
 - We can **quickly select decent models based on data subsets**

Learning Curve Tuning



~ We only want to train the best learning curves until the end

Successive Halving

- Ideas:
 - Invest only resources in promising configurations
 - ↪ aggressive dropping of poor configurations
 - Model-free — (more or less assumption free)

Successive Halving

- Ideas:
 - Invest only resources in promising configurations
 - ↪ aggressive dropping of poor configurations
 - Model-free — (more or less assumption free)
- Algorithm Outline:
 - Input: n (randomly sampled) configurations and budget B

Successive Halving

- Ideas:
 - Invest only resources in promising configurations
 - ↪ aggressive dropping of poor configurations
 - Model-free — (more or less assumption free)
- Algorithm Outline:
 - Input: n (randomly sampled) configurations and budget B
 - ① Run remaining configurations with some resource allocation (depending on B)

Successive Halving

- Ideas:
 - Invest only resources in promising configurations
 - ~> aggressive dropping of poor configurations
 - Model-free — (more or less assumption free)
- Algorithm Outline:
 - Input: n (randomly sampled) configurations and budget B
 - ① Run remaining configurations with some resource allocation (depending on B)
 - ② Sort configurations by cost (e. g., validation loss)
 - ③ Throw away lower half of configurations

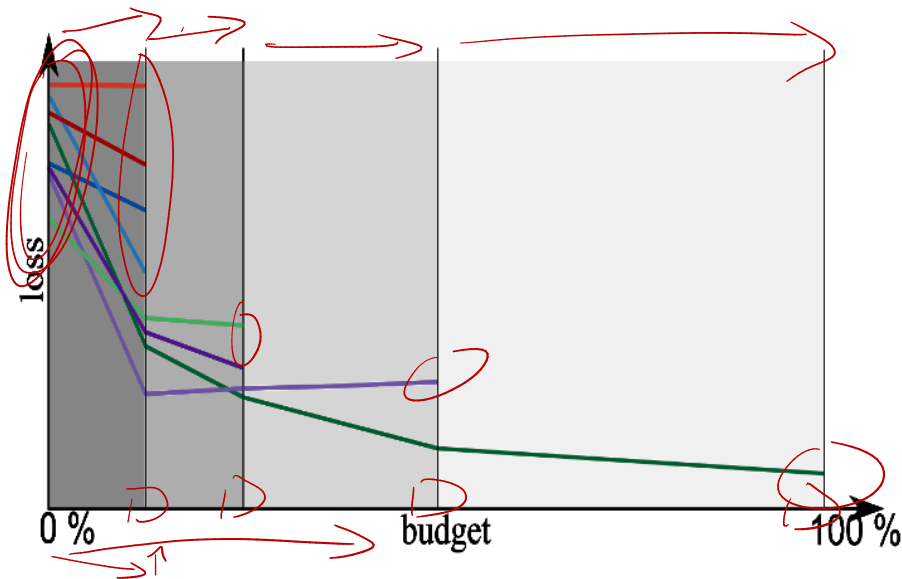
Successive Halving

- Ideas:
 - Invest only resources in promising configurations
 - ~> aggressive dropping of poor configurations
 - Model-free — (more or less assumption free)
- Algorithm Outline:
 - Input: n (randomly sampled) configurations and budget B
 - ① Run remaining configurations with some resource allocation (depending on B)
 - ② Sort configurations by cost (e. g., validation loss)
 - ③ Throw away lower half of configurations
 - ④ Repeat

Successive Halving

- Ideas:
 - Invest only resources in promising configurations
 - ↪ aggressive dropping of poor configurations
 - Model-free — (more or less assumption free)
- Algorithm Outline:
 - Input: n (randomly sampled) configurations and budget B
 - ① Run remaining configurations with some resource allocation (depending on B)
 - ② Sort configurations by cost (e. g., validation loss)
 - ③ Throw away lower half of configurations
 - ④ Repeat
- Resource allocation can correspond to
 - partial learning curves
 - subset of training data

Successive Halving: Illustration



Hyperband

- Issue of successive halving (for a fixed B):
Do you want to run many configurations with aggressive rejection?
Or: Do you want to run few configurations with non-aggressive rejection?

Hyperband

- Issue of successive halving (for a fixed B):
Do you want to run many configurations with aggressive rejection?
Or: Do you want to run few configurations with non-aggressive rejection?
- Ideas:
 - Add an outer loop to try different trade-offs between #configurations and budget
 - Add further parameter: proportion of configurations discarded in each round of successive halving

Hyperband

- Issue of successive halving (for a fixed B):
Do you want to run many configurations with aggressive rejection?
Or: Do you want to run few configurations with non-aggressive rejection?
- Ideas:
 - Add an outer loop to try different trade-offs between #configurations and budget
 - Add further parameter: proportion of configurations discarded in each round of successive halving
- Starts with many configurations that gets aggressively rejected

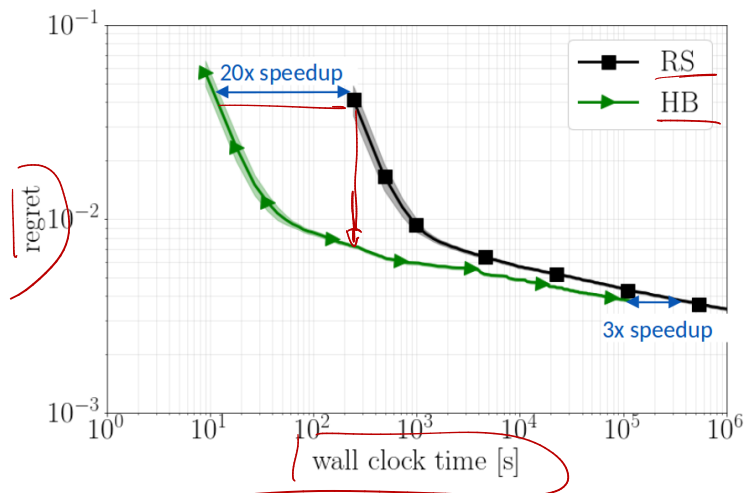
Hyperband

- Issue of successive halving (for a fixed B):
Do you want to run many configurations with aggressive rejection?
Or: Do you want to run few configurations with non-aggressive rejection?
- Ideas:
 - Add an outer loop to try different trade-offs between #configurations and budget
 - Add further parameter: proportion of configurations discarded in each round of successive halving
- Starts with many configurations that gets aggressively rejected
- In later iterations, few configurations with more budget each

Hyperband

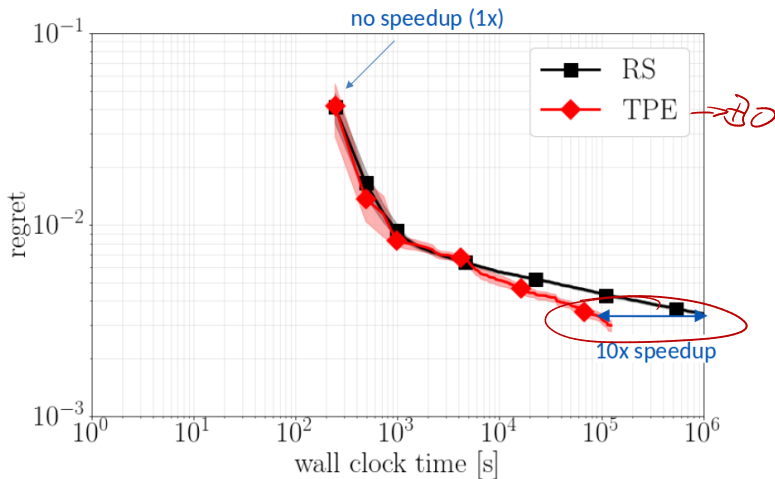
- Issue of successive halving (for a fixed B):
Do you want to run many configurations with aggressive rejection?
Or: Do you want to run few configurations with non-aggressive rejection?
- Ideas:
 - Add an outer loop to try different trade-offs between #configurations and budget
 - Add further parameter: proportion of configurations discarded in each round of successive halving
- Starts with many configurations that gets aggressively rejected
- In later iterations, few configurations with more budget each
- Returns: configuration with the smallest intermediate loss seen so far.

Random Search vs. Hyperband



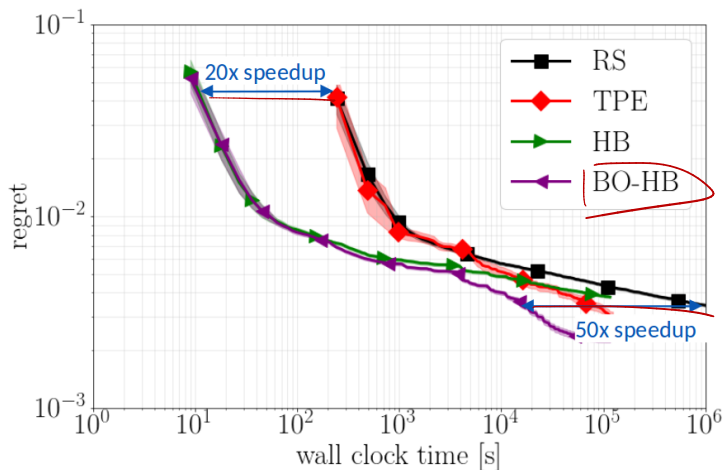
⇒ Hyperband performs well early on

Random Search vs. Bayesian Optimization



↪ Bayesian optimization performs well later on

Random Search vs. Bayesian Optimization vs. Hyperband



⇒ A combination of Bayesian Optimization and Hyperband performs well overall

BOHB: a Robust & Efficient Framework for Hyperparameter Optimization

- BOHB: Bayesian Optimization & HyperBand
 - Bayesian optimization for selecting configurations
 - Hyperband for implementing the speedup techniques, using the concept of a budget (size of data subsets, #epochs, #CV folds, etc)

BOHB: a Robust & Efficient Framework for Hyperparameter Optimization

- BOHB: Bayesian Optimization & HyperBand
 - Bayesian optimization for selecting configurations
 - Hyperband for implementing the speedup techniques, using the concept of a budget (size of data subsets, #epochs, #CV folds, etc)
 - Currently one of the best available approach for robust & efficient hyperparameter optimization

BOHB: a Robust & Efficient Framework for Hyperparameter Optimization

- BOHB: Bayesian Optimization & HyperBand
 - Bayesian optimization for selecting configurations
 - Hyperband for implementing the speedup techniques, using the concept of a budget (size of data subsets, #epochs, #CV folds, etc)
 - Currently one of the best available approach for robust & efficient hyperparameter optimization
- You can use this for optimizing your own hyperparameters
<https://github.com/automl/HpBandSter>

Summary by learning goals

Having heard this lecture, you can now . . .

- Describe the difference between **parameters** and **hyperparameters**
- Give some **examples of hyperparameters** of various algorithms
- Describe the ideas behind **random search** and **Bayesian optimization**
- Explain methods to speedup hyperparameter optimization by using **grey-box approaches**

- Random search: [Bergstra et al, 2012]
- Bayesian optimization: [Mockus, 1978]; [Brochu et al, 2010]
- BOHB: [Falkner et al, 2019]