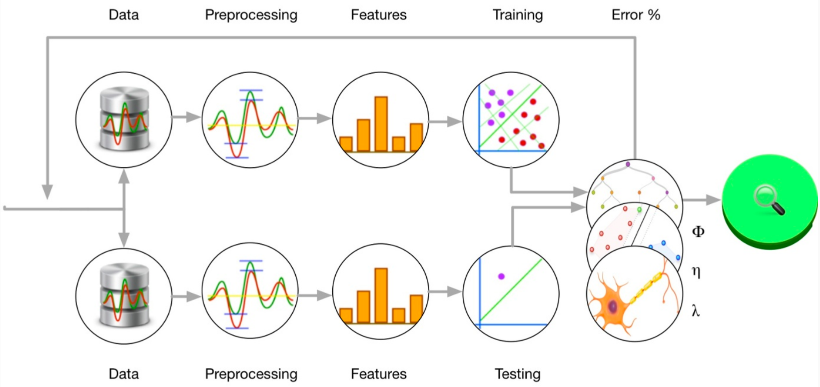
Introduction Automated Machine Learning

Aaron Klein

May 8, 2019

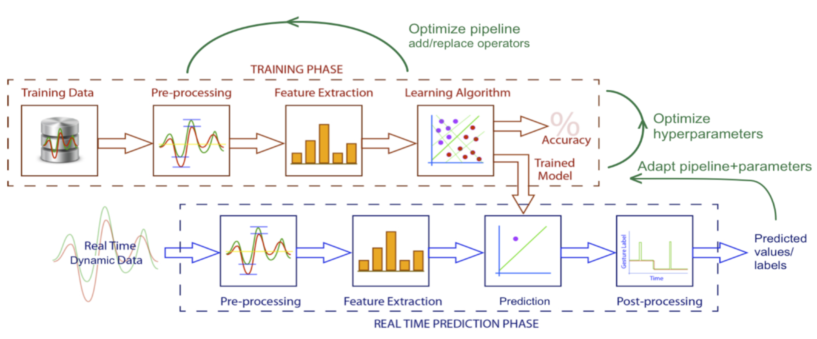
University of Freiburg

Machine Learning Pipeline

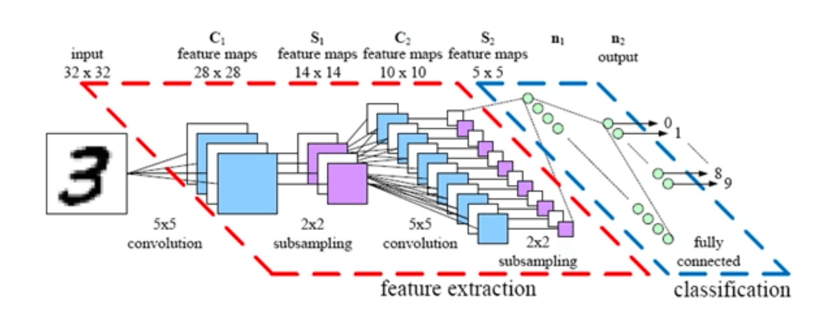


(Image Courtesy Joaquin Vanschoren)

Automated Machine Learning



(Image Courtesy Joaquin Vanschoren)



(Image Courtesy Joaquin Vanschoren)

Neural Architecture Search

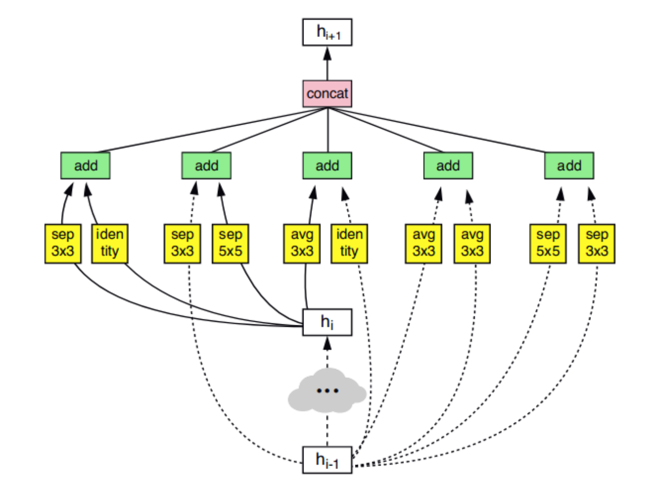


Image from [Zoph et al., 2018]

Hyperparameter Optimization

Finding the right hyperparameters for a machine learning algorithm A can be defined as an optimization problem:

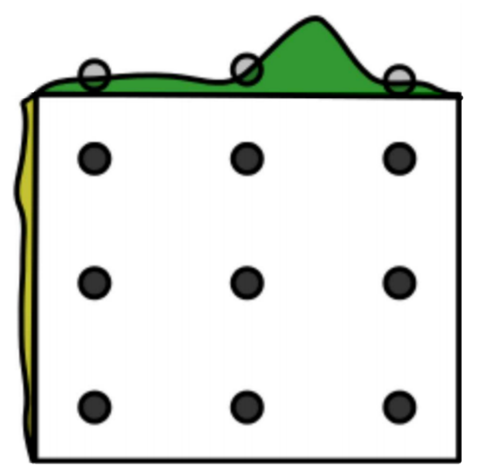
$$\boldsymbol{x}_\star \in \arg\min_{\boldsymbol{x} \in \mathbb{X}} f(\boldsymbol{x})$$

- $\boldsymbol{x}$ denotes all hyperparameters that should be optimized
- $\mathbb{X}$ is the configuration space which specifies the domain for each hyperparameter
- f measures the error of training A with hyperparameters $\boldsymbol{x}$, e. g. validation error
- we assume f to be noisy, i. e. we only observe $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \varepsilon$ where $\varepsilon \sim \mathcal{N}(0, \sigma_{noise})$

Configspace

	Name	Range	Default	log scale	Type	Conditional
Network hyperparameters	batch size	$[32, 4096]$	32	✓	float	-
	number of updates	$[50, 2500]$	200	✓	int	-
	number of layers	$[1, 6]$	1	-	int	-
	learning rate	$[10^{-6}, 1.0]$	10^{-2}	✓	float	-
	L_2 regularization	$[10^{-7}, 10^{-2}]$	10^{-4}	✓	float	-
	dropout output layer	$[0.0, 0.99]$	0.5	✓	float	-
	solver type	{SGD, Momentum, Adam, Adadelta, Adagrad, smorm, Nesterov }	smorm3s	-	cat	-
	lr-policy	{Fixed, Inv, Exp, Step}	fixed	-	cat	-
Conditioned on solver type	β_1	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	β_2	$[10^{-4}, 10^{-1}]$	10^{-1}	✓	float	✓
	ρ	$[0.05, 0.99]$	0.95	✓	float	✓
	momentum	$[0.3, 0.999]$	0.9	✓	float	✓
Conditioned on lr-policy	γ	$[10^{-3}, 10^{-1}]$	10^{-2}	✓	float	✓
	k	$[0.0, 1.0]$	0.5	-	float	✓
	s	$[2, 20]$	2	-	int	✓
Per-layer hyperparameters	activation-type	{Sigmoid, TanH, ScaledTanH, ELU, ReLU, Leaky, Linear}	ReLU	-	cat	✓
	number of units	$[64, 4096]$	128	✓	int	✓
	dropout in layer	$[0.0, 0.99]$	0.5	-	float	✓
	weight initialization	{Constant, Normal, Uniform, Glorot-Uniform, Glorot-Normal, He-Normal, He-Uniform, Orthogonal, Sparse}	He-Normal	-	cat	✓
	std. normal init.	$[10^{-7}, 0.1]$	0.0005	-	float	✓

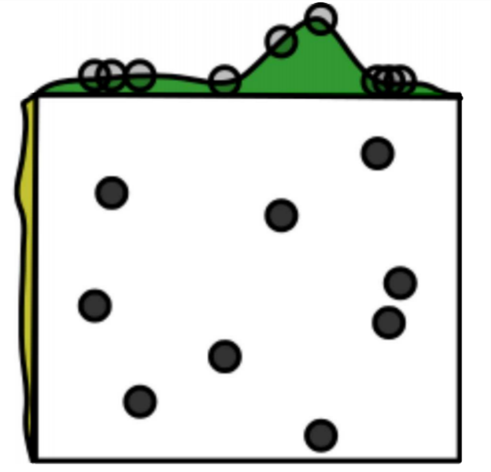
Grid Search



Grid Search

- easy to implement and to parallelize
- in continuous spaces unlikely to find the global optimum

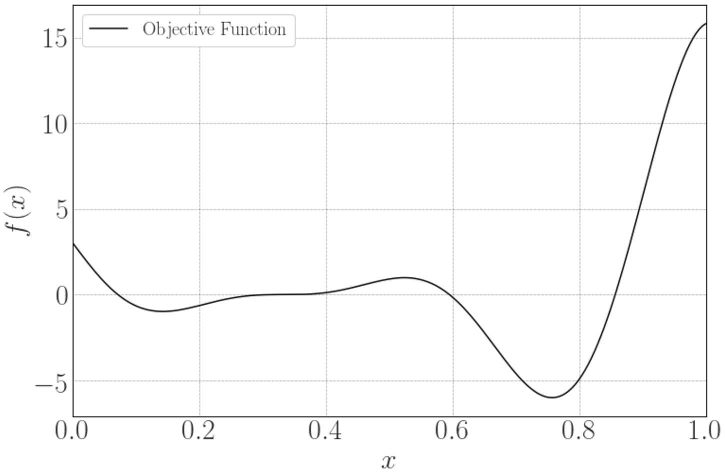
Random Search



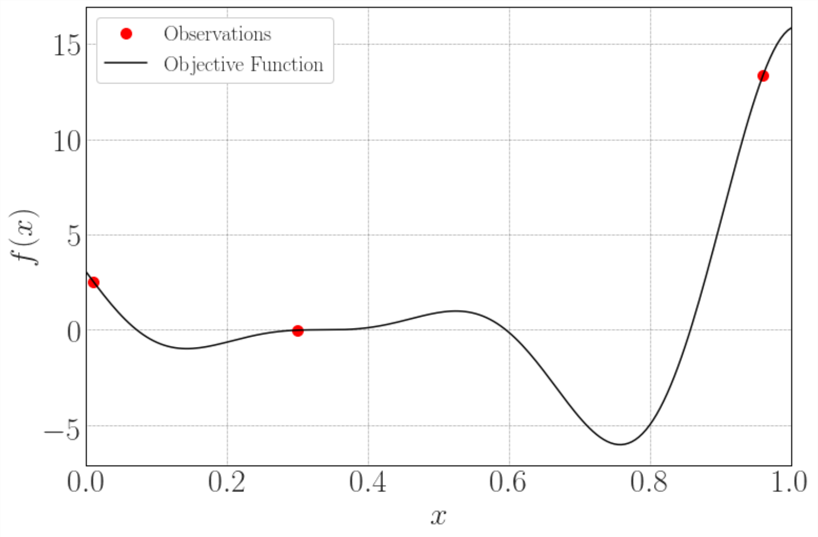
Random Search

- also easy to implement and to parallelize
- if all hyperparameters have non-zero probability, random search is guaranteed to converge to the global optimum
- cannot exploit knowledge obtain from previous function evaluations

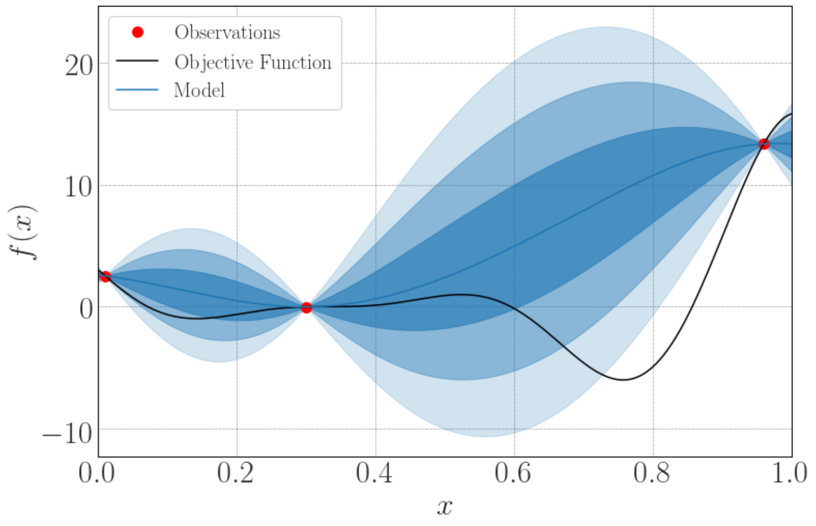
Bayesian optimization



Bayesian optimization



Bayesian optimization



Gaussian Process

We can model the objective function $f(\boldsymbol{x})$ with a Gaussian process [Rasmussen and Williams, 2006]:

$$f(\boldsymbol{x}) \sim \mathrm{GP}(\mu(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$$

A Gaussian process is fully defined by:

- a mean function $\mu(\boldsymbol{x})$ which is usually set to $\mu(\boldsymbol{x}) = 0$
- a kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$ which measures the similarity between two points $\boldsymbol{x}$ and $\boldsymbol{x}'$. For example the RBF kernel:

  $$k(x, x') = \theta_0 \cdot \exp\left(-\frac{\|x - x'\|^2}{\theta_1}\right)$$

  where θ_0 and θ_1 are hyperparameters.

Given new observed data D we can compute the posterior mean $\mu(\boldsymbol{x}|\theta, D)$ and variance $\sigma^2(\boldsymbol{x}|\theta, D)$ analytically.

Gaussian Process

Pros:

- smooth and reliable uncertainty estimates
- priors can easily be incorporated

Cons:

- not easily applicable in discrete or conditional spaces
- scales cubically with the number of data points
- sensitive to its own hyperparameters

Random Forest

Consists of T regression trees where each tree splits the input space into disjoint regions $S_0, \ldots S_{L-1}$ where L is the number of leafs. Tree prediciation for unseen points:

$$\tilde{\mu}(\boldsymbol{x}_\star) = \sum_{l=1}^{L} c_l \cdot \mathbb{I}(\boldsymbol{x}_\star \in S_i) \tag{1}$$

with $\mathbb{I}$ as the indicator function that returns 1 if $\boldsymbol{x}_\star \in S_i$ and 0 otherwise.

For unseen test points we can compute the predictive distribution by:

$$\mu(\boldsymbol{x}_\star) = \frac{1}{T} \sum_{t=1}^{T} \tilde{\mu}_t(\boldsymbol{x}_\star) \tag{2}$$

$$\sigma^2(\boldsymbol{x}_\star) = \frac{1}{T} \sum_{t=1}^{T} (\tilde{\mu}_t(\boldsymbol{x}_\star) - \mu(\boldsymbol{x}_\star))^2 \tag{3}$$

Random Forest

Pros:

- scales much better with data
- can easily handle categorical, continuous and discrete spaces
- fairly robust against its own hyperparameters

Cons:

- the uncertainty estimates are often poor
- do not extrapolate well
- priors cannot easily be incorporated

Bayesian Neural Networks

Bayesian neural networks use a Bayesian treatment of neural network weight to obtain uncertainty estimates (see [Springenberg et al., 2016, Snoek et al., 2015])

Pros:

- scales much better with data
- can easily handle categorical, continuous and discrete spaces
- given enough network samples obtain nice and smooth uncertainty estimates

Cons:

- need usually more data than Gaussian processes
- brittle against its own hyperparameters.

DNGO [Snoek et al., 2015]

- fit simple 3-layer feed forward neural network with linear output layer on $\boldsymbol{X}, \boldsymbol{y}$
- after training, remove output layer
- use features of last layer as basis functions for Bayesian linear regression to get uncertainty estimates

Recap Bayesian Linear Regression

Given some data points $\boldsymbol{X} \in \mathbb{R}^{N\times D}$ with targets $\boldsymbol{y} \in \mathbb{R}^{N}$ we model:

$$y_i = x_i \boldsymbol{w} + \varepsilon_i \tag{4}$$

where we assume that $\varepsilon_i \sim \mathcal{N}(0, \frac{1}{\beta})$

Recap Bayesian Linear Regression

By assuming a Gaussian prior $p(\boldsymbol{w} \mid \alpha) = \mathcal{N}(\boldsymbol{w} \mid 0, \alpha^{-1}\mathbb{I})$ we can compute the posterior in closed form $p(\boldsymbol{w} \mid \mathbf{m}, \mathbf{K})$ after observing some data $\boldsymbol{X}$, $\boldsymbol{y}$, where:

$$\mathbf{m} = \beta \mathbf{K}^{-1} \boldsymbol{X} \boldsymbol{y} \tag{5}$$

$$\mathbf{K} = \beta \boldsymbol{X}^T \boldsymbol{X} + \alpha \mathbb{I} \tag{6}$$

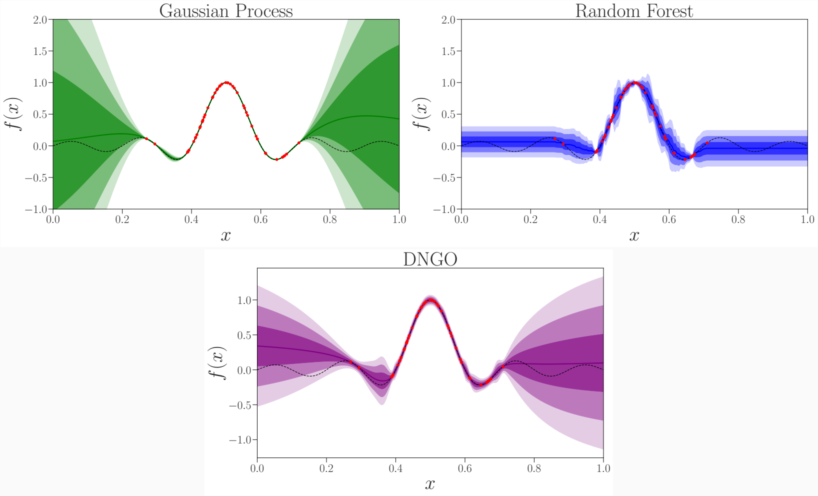
Recap Bayesian Linear Regression

For unseen test points $\boldsymbol{x}_\star$ the predictive distribution is a Gaussian $p(y_\star \mid \boldsymbol{x}_\star, \boldsymbol{X}, \boldsymbol{y}, \alpha, \beta) = \mathcal{N}(y_\star \mid m_\star, \sigma_\star^2)$:

$$m_\star = \mathbf{m}^T \boldsymbol{x}_\star \tag{7}$$

$$\sigma_\star = \frac{1}{\beta} + \boldsymbol{X}^T \mathbf{K} \boldsymbol{X} \tag{8}$$

Model Comparison



Exploitation vs Exploration

Given our model m and some data $D = \{(\boldsymbol{x}_0, y_0), \ldots (\boldsymbol{x}_n, y_n)\}$ how do we **decide** which hyperparameter configuration $\boldsymbol{x}_{n+1}$ we shall evaluate next?

Exploitation vs Exploration

Given our model m and some data $D = \{(\boldsymbol{x}_0, y_0), \ldots (\boldsymbol{x}_n, y_n)\}$ how do we **decide** which hyperparameter configuration $\boldsymbol{x}_{n+1}$ we shall evaluate next?

Naive solution: simply optimize $\mu(\boldsymbol{x})$, however, that would only pick points around the best observed point.

Exploitation vs Exploration

Given our model m and some data $D = \{(\boldsymbol{x}_0, y_0), \ldots (\boldsymbol{x}_n, y_n)\}$ how do we **decide** which hyperparameter configuration $\boldsymbol{x}_{n+1}$ we shall evaluate next?

Naive solution: simply optimize $\mu(\boldsymbol{x})$, however, that would only pick points around the best observed point.

We have to trade off between:

- **exploring** in regions of the configuration space where our model is uncertain
- however, since our ultimate goal is to locate the global optimum $\boldsymbol{x}_\star$, we also want to **exploit** in the good regions of the configuration space

Acquisition Functions

We use an acquisition function $a(\boldsymbol{x})$ that automatically trades off exploration and exploitation.

To find the next point $\boldsymbol{x}_{n+1}$ we numerically optimize $a(\boldsymbol{x})$:

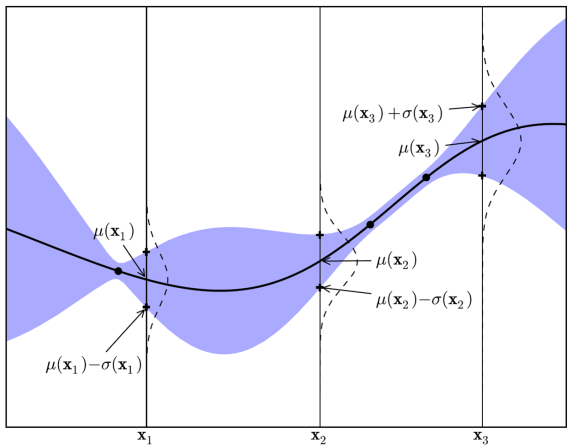
$$\boldsymbol{x}_{n+1} \in \arg\max_{\boldsymbol{x} \in \mathbb{X}} a(\boldsymbol{x})$$

Since the acquisition function only depends on our model, it is cheap to evaluate and often provides gradient information.

Common ways to optimize the acquisition function:

- Gradient Ascent
- Evolutionary Algorithms
- Local Search
- Random Search

Upper Confidence Bound



Upper Confidence Bound [Srinivas et al., 2010]

Computes the acquisition function by:

$$a(\boldsymbol{x}) = \mu(\boldsymbol{x}) + \beta\sigma(\boldsymbol{x})$$

- β is a hyperparameter that controls exploration and exploitation
- under some assumptions, you can proof that UCB converges to the global optimum

Expected Improvement [Jones et al., 1998]

Probably the most often used acquisition function is expected improvement, which computes:

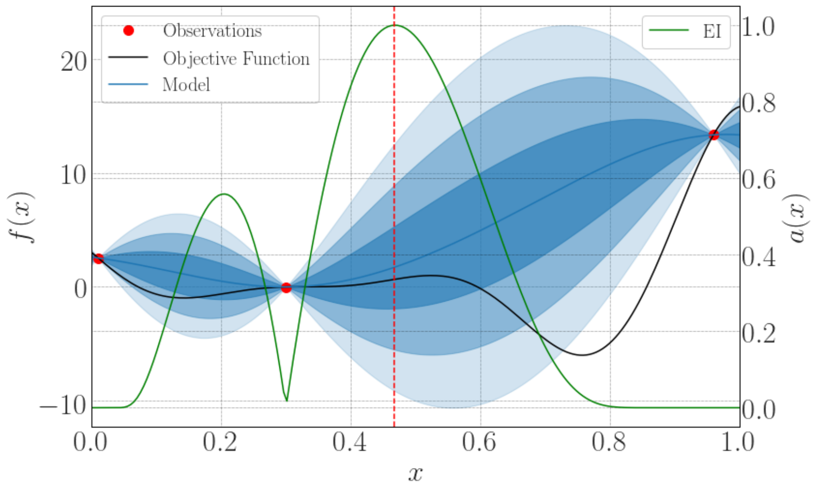
$$a(\boldsymbol{x}) = E_{p(f|D)}[\max(y_\star - f(\boldsymbol{x}), 0)].$$

where $y_\star \in \arg\min\{y_0, \ldots, y_n\}$. Assuming $p(f|D)$ to be a Gaussian, we can compute EI in closed form by:

$$a(\boldsymbol{x}) = \sigma(\boldsymbol{x})(\gamma(\boldsymbol{x})\Phi(\gamma(\boldsymbol{x})) + \phi(\gamma(\boldsymbol{x})))$$

here $\gamma(\boldsymbol{x}) = \frac{y_\star - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}$ and Φ is the CDF and ϕ is the PDF of a standard normal distribution.

Acquisition Functions



Bayesian Optimization [Jones et al., 1998]

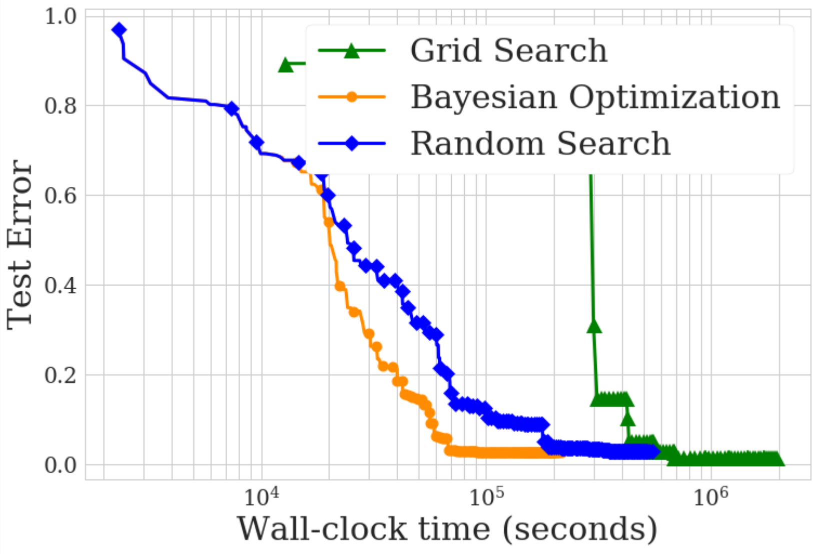
Algorithm 1 Bayesian Optimization

```
Initialize data $D_0$ using an initial design.
for $t = 1, 2, \ldots$ do
    Fit probabilistic model for $f(\boldsymbol{x})$ on data $D_{t-1}$
    Choose $\boldsymbol{x}_t$ by maximizing the acquisition function $a(\boldsymbol{x})$
    Evaluate $y_t \sim f(\boldsymbol{x}_t) + \mathcal{N}(0, \sigma^2)$, and augment the data: $D_t = D_{t-1} \cup \{(\boldsymbol{x}_t, y_t)\}$
    Choose incumbent $\hat{\boldsymbol{x}}_t \leftarrow \arg\min\{y_1, \ldots y_t\}$
```

Bayesian optimization

- uses a probabilistic model to guide the search towards the global optimum
- under some assumptions converges to the global optimum
- more tricky to implement, especially in a parallel setting

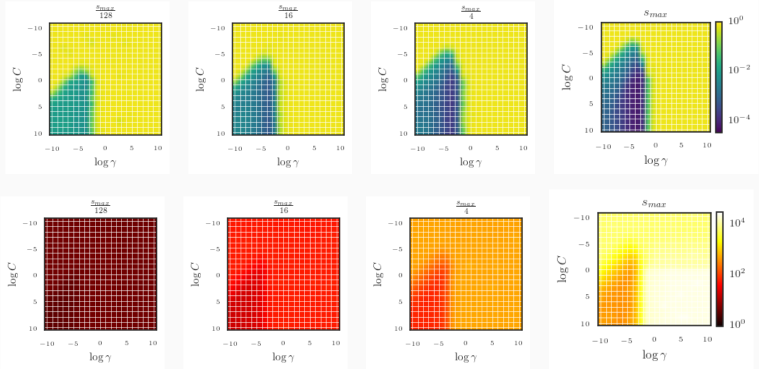
Bayesian Optimization



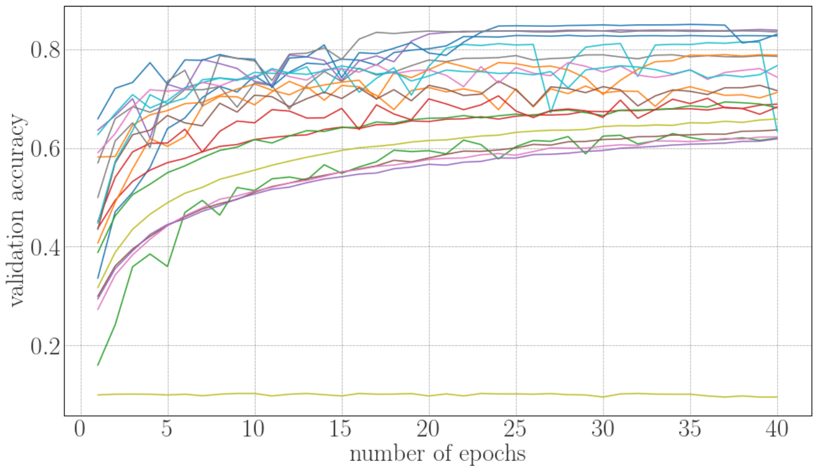
Multi-fidelity Optimization

- Even though Bayesian optimization is *sample efficient*, it still requires tens to hundreds of function evaluations.
- We often have access to *cheap-to-evaluate* approximations $\tilde{f}(\cdot, b)$ of of the true objective function $f(\cdot)$, so called **fidelities**.
- Each fidelity is parameterized by a so-called *budget* $b \in [b_{min}, b_{max}]$.
  - if $b = b_{max}$: then $\tilde{f}(\cdot, b_{max}) = f(\cdot)$
  - if $b < b_{max}$: then $\tilde{f}(\cdot, b)$ is only an approximation of $f(\cdot)$ whose quality typically increases with b.

Dataset Subsets [Klein et al., 2017]



Learning Curves



Successive Halving [Jamieson and Talwalkar, 2016]

Algorithm 2 Successive Halving

Require: initial budget b_0, maximum budget b_{max}, set of n configurations $C = \{c_1, c_2, \dots c_n\}$

```
b = b_0
while b ≤ b_max do
    L = {f̃(c, b) : c ∈ C}
    C = top_k(C, L, ⌊|C|/η)⌋
    b = η · b
```

$b = b_0$

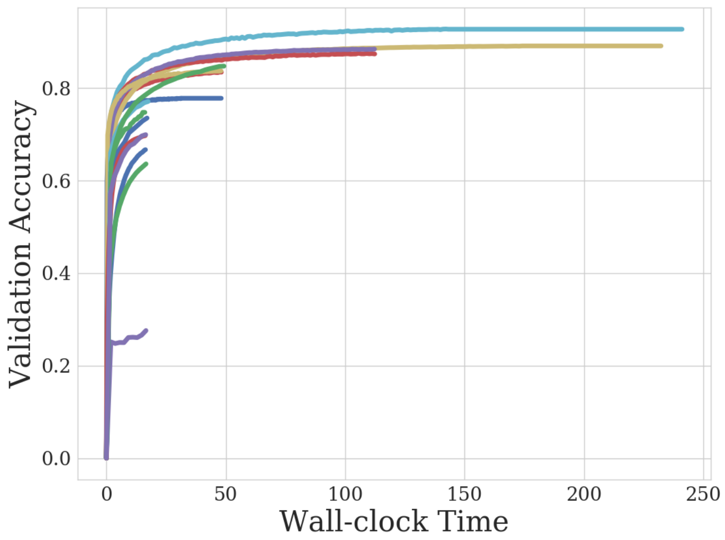
while $b \leq b_{max}$ **do**

$\quad L = \{\tilde{f}(c, b) : c \in C\}$

$\quad C = \mathrm{top_k}(C, L, \lfloor |C|/\eta)\rfloor$

$\quad b = \eta \cdot b$

Successive Halving



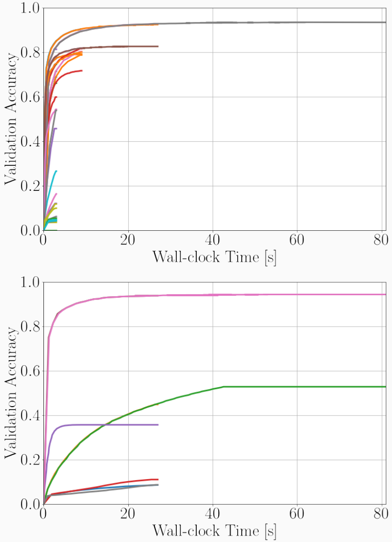
Hyperband [Li et al., 2017]

Algorithm 3 Hyperband

Require: budgets b_{min} and b_{max}, η

$s_{max} = \lfloor \log_\eta \frac{b_{max}}{b_{min}} \rfloor$
for $s \in \{s_{max}, s_{max} - 1, \ldots, 0\}$ **do**
  sample $n = \lceil \frac{s_{max}+1}{s+1} \cdot \eta^s \rceil$ configurations
  run SH on them with $\eta^s \cdot b_{max}$ as initial budget

Hyperband


Validation Accuracy
Wall-clock Time [s]
0.0
0.2
0.4
0.6
0.8
1.0
0
20
40
60
80

Hyperband vs. Bayesian Optimization [Falkner et al., 2018]

Hyperband:

- very efficient in terms of anytime performance
- due to the random sampling, cannot reuse previously gain knowledge and take a long time to converge

Bayesian optimization:

- in its standard form it cannot exploit fidelites (however, several extensions exist)
- in the most cases converges faster than random search

Can we combine both methods?

Tree of Parzen Estimators [Bergstra et al., 2011]

We fit two kernel density estimator for the good and bad configurations:

$$l(\boldsymbol{x}) = p(y < \alpha | \boldsymbol{x}, D)$$
$$g(\boldsymbol{x}) = p(y > \alpha | \boldsymbol{x}, D)$$

To select a new candidate $\boldsymbol{x}_{new}$ to evaluate, it maximizes the ratio $\frac{l(\boldsymbol{x})}{g(\boldsymbol{x})}$, which is equivalent of optimizing expected improvement.

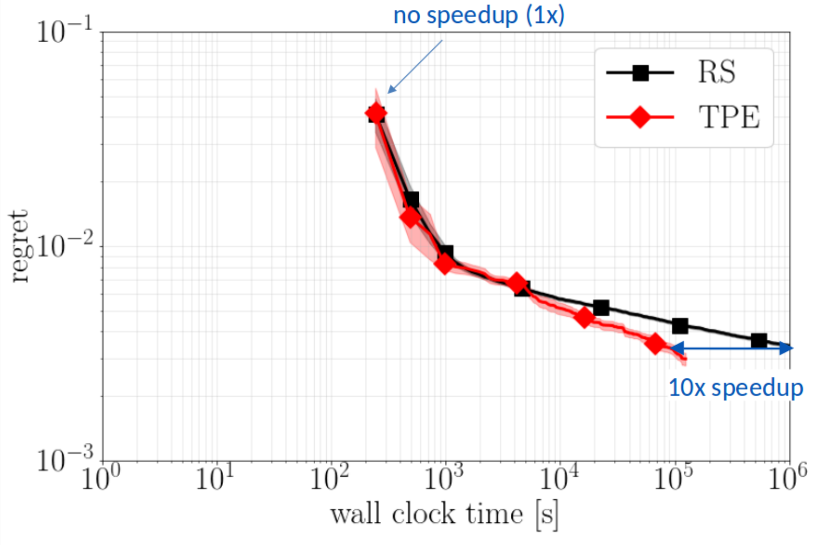
BOHB [Falkner et al., 2018]

Algorithm 4 Pseudocode for sampling in BOHB

Require: observations D, fraction of random runs ρ, percentile q, number of samples N_s, minimum number of points N_{min} to build a model, and bandwidth factor b_w

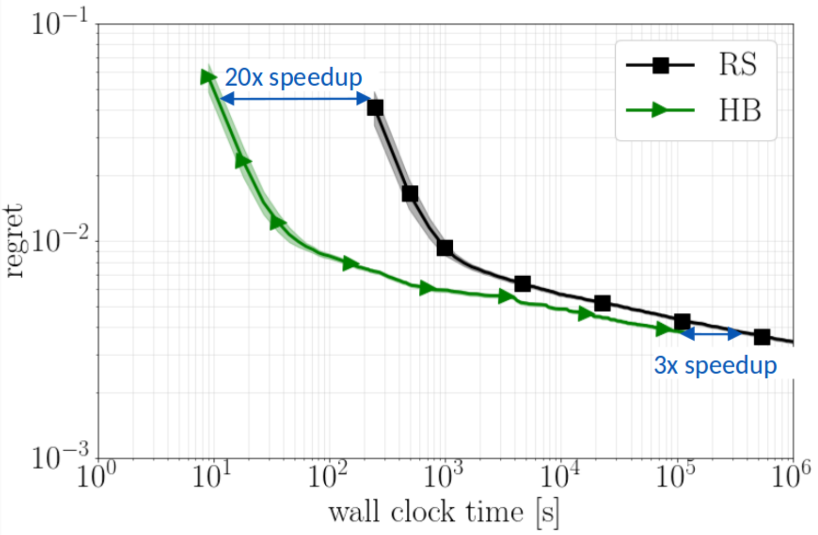
if rand() $\leq \rho$ **then**
    return random configuration
$b = \arg\max \left\{ D_b : |D_b| \geq N_{min} + 2 \right\}$
if $b = \emptyset$ **then**
    return random configuration
fit KDEs as in TPE but for each budget b
draw N_s samples according to $l'(\boldsymbol{x})$
return sample with highest ratio $l(\boldsymbol{x})/g(\boldsymbol{x})$

BOHB



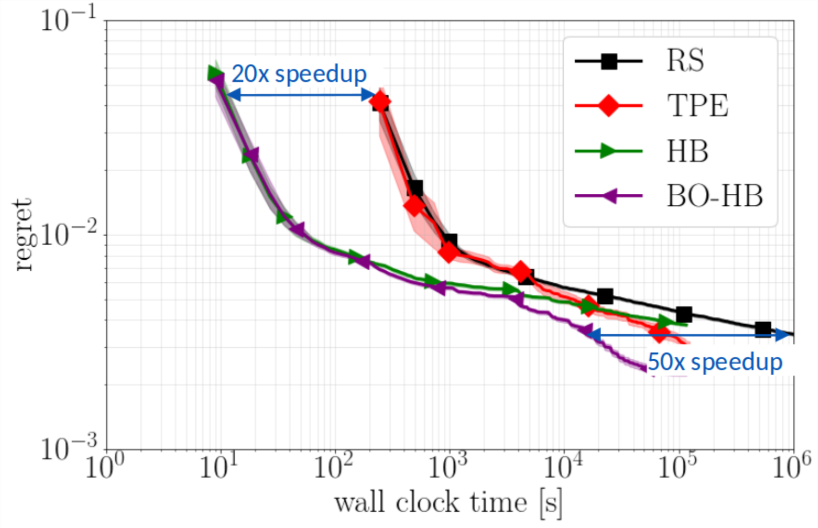
no speedup (1x)
RS
TPE
10x speedup
regret
10^{-1}
10^{-2}
10^{-3}
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
wall clock time [s]

BOHB



RS
HB
20x speedup
3x speedup
10^{-1}
10^{-2}
10^{-3}
regret
10^{0}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
wall clock time [s]

BOHB

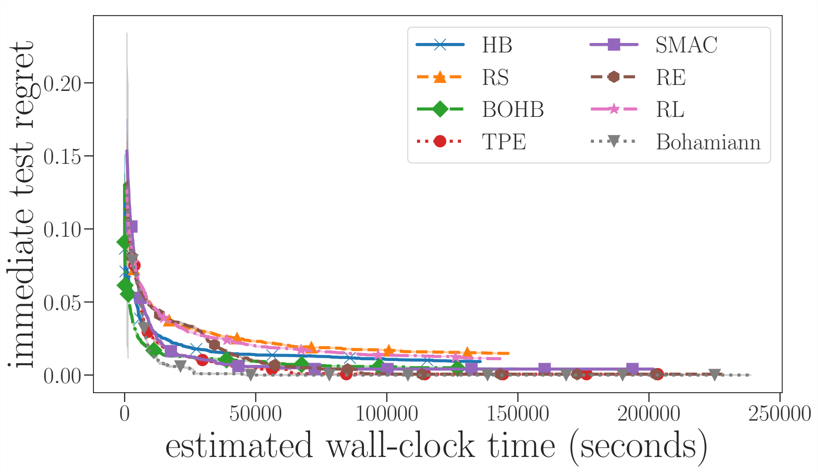


regret
10^{-1}
10^{-2}
10^{-3}
10^0
10^1
10^2
10^3
10^4
10^5
10^6
wall clock time [s]
RS
TPE
HB
BO-HB
20x speedup
50x speedup

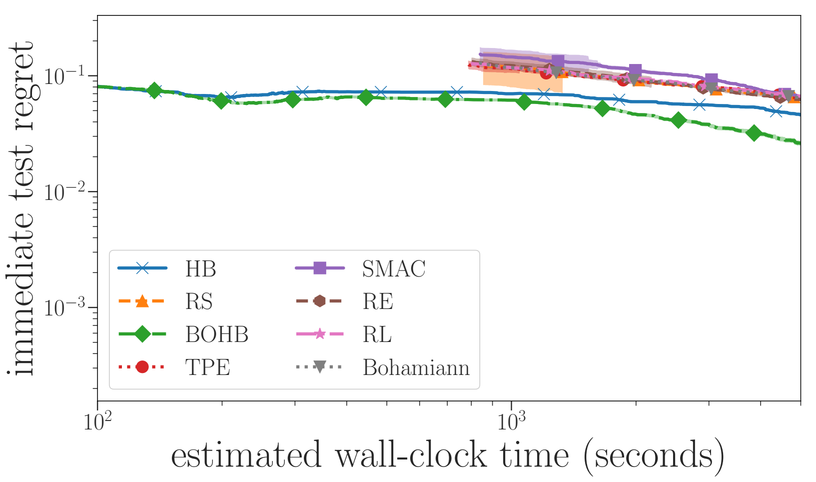
Benchmarking

- Benchmarking is important to make further progress in the field
- Large computational demands make thorough benchmarking extremely expensive
- In practice this slows down development of new methods

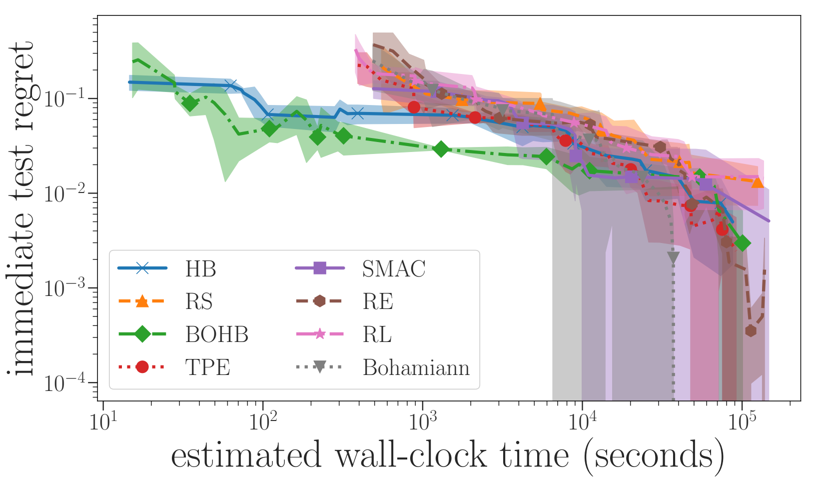
Benchmarking Pitfalls



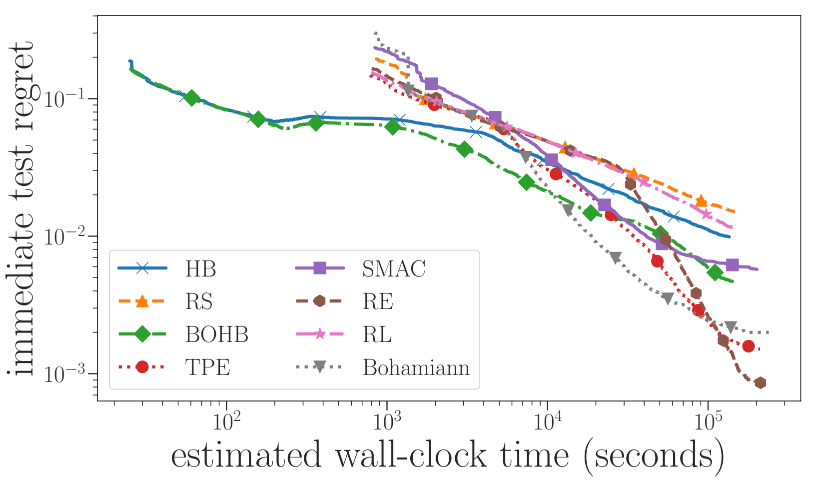
Benchmarking Pitfalls



Benchmarking Pitfalls



Benchmarking Pitfalls



Pro Tips for Benchmarking

- Almost all optimizers are **randomized**, i.e depend on the seed, thus, we need a sufficent amount of independent runs to getter a better estimate of a optimizer's performance
- Run all optimizers **sufficiently long**, since in the beginning the most methods do not work better than random
- Plot the **performance over time** rather than just the final performance
- Use **log-scales** and learn how to read them

Conclusions

- Bayesian optimization is an efficient strategy for hyperparameter optimization
- By using fidelities of the objective function we can speed up the optimization procedure
- Hyperband is an extension of random search that exploits multi-fidelity of the objective function,
- BOHB combines Hyperband with Bayesian optimization to combine the strengths of both methods
- Benchmarking plays an important role in developing new methods

References I

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011).
Algorithms for hyper-parameter optimization.
In *Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NIPS'11)*.

Eggensperger, K., Hutter, F., Hoos, H., and Leyton-Brown, K. (2015).
Efficient benchmarking of hyperparameter optimizers via surrogates.
In *Proceedings of the 29th National Conference on Artificial Intelligence (AAAI'15)*.

References II

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

Hutter, F., Hoos, H., and Leyton-Brown, K. (2011).
Sequential model-based optimization for general algorithm configuration.
In *Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11)*.

References III

Jamieson, K. and Talwalkar, A. (2016).

Non-stochastic best arm identification and hyperparameter optimization.

In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTATS'16)*.

Jones, D., Schonlau, M., and Welch, W. (1998).

Efficient global optimization of expensive black box functions.

Journal of Global Optimization.

References IV

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. (2017).

Fast Bayesian hyperparameter optimization on large datasets.

In *Electronic Journal of Statistics*.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2017).

Hyperband: Bandit-based configuration evaluation for hyperparameter optimization.

In *International Conference on Learning Representations (ICLR'17)*.

References V

Rasmussen, C. and Williams, C. (2006).
Gaussian Processes for Machine Learning.
The MIT Press.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, and Adams, R. (2015).
Scalable Bayesian optimization using deep neural networks.
In *Proceedings of the 32nd International Conference on Machine Learning (ICML'15)*.

References VI

Springenberg, J. T., Klein, A., Falkner, S., and Hutter, F. (2016).
Bayesian optimization with robust bayesian neural networks.
In *Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NIPS'16)*.

Srinivas, N., Krause, A., Kakade, S., and Seeger, M. (2010).
Gaussian process optimization in the bandit setting: No regret and experimental design.
In *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*.

References VII

Ying, C., Klein, A., Real, E., Christiansen, E., Murphy, K., and Hutter, F. (2019).
NAS-Bench-101: Towards reproducible neural architecture search.
arXiv:1902.09635 [cs.LG].

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018).
Learning transferable architectures for scalable image recognition.
In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR'18)*.