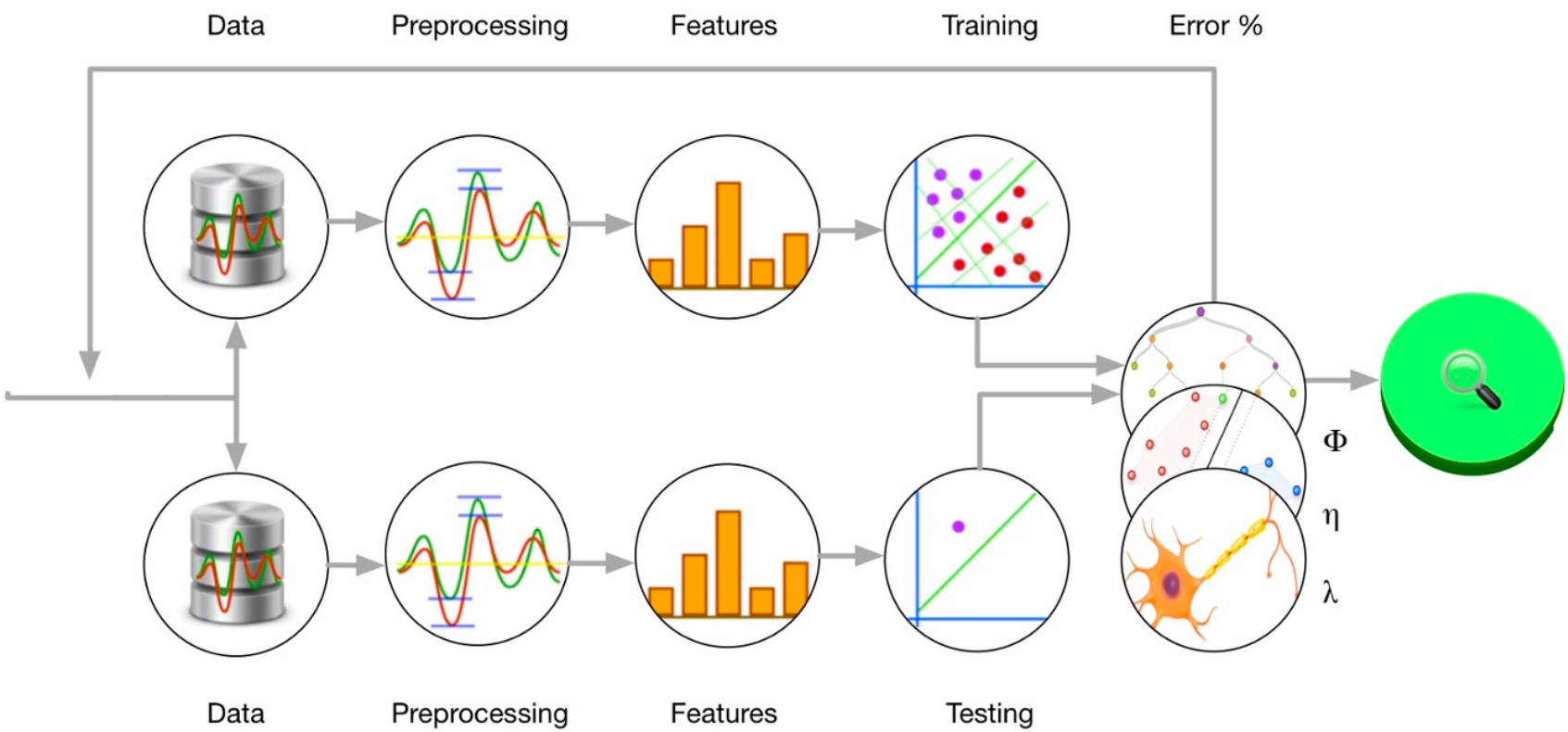
Introduction Automated Machine Learning

Aaron Klein

University of Freiburg

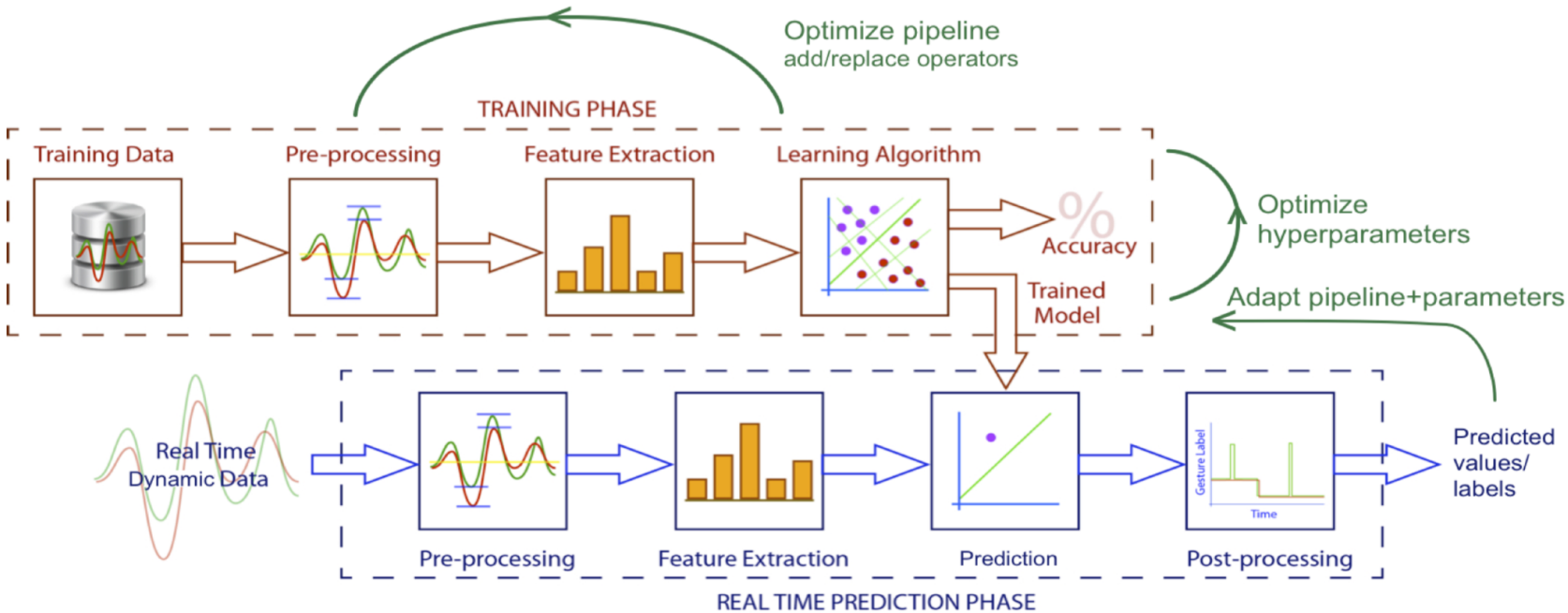
December 4, 2018

Machine Learning Pipeline



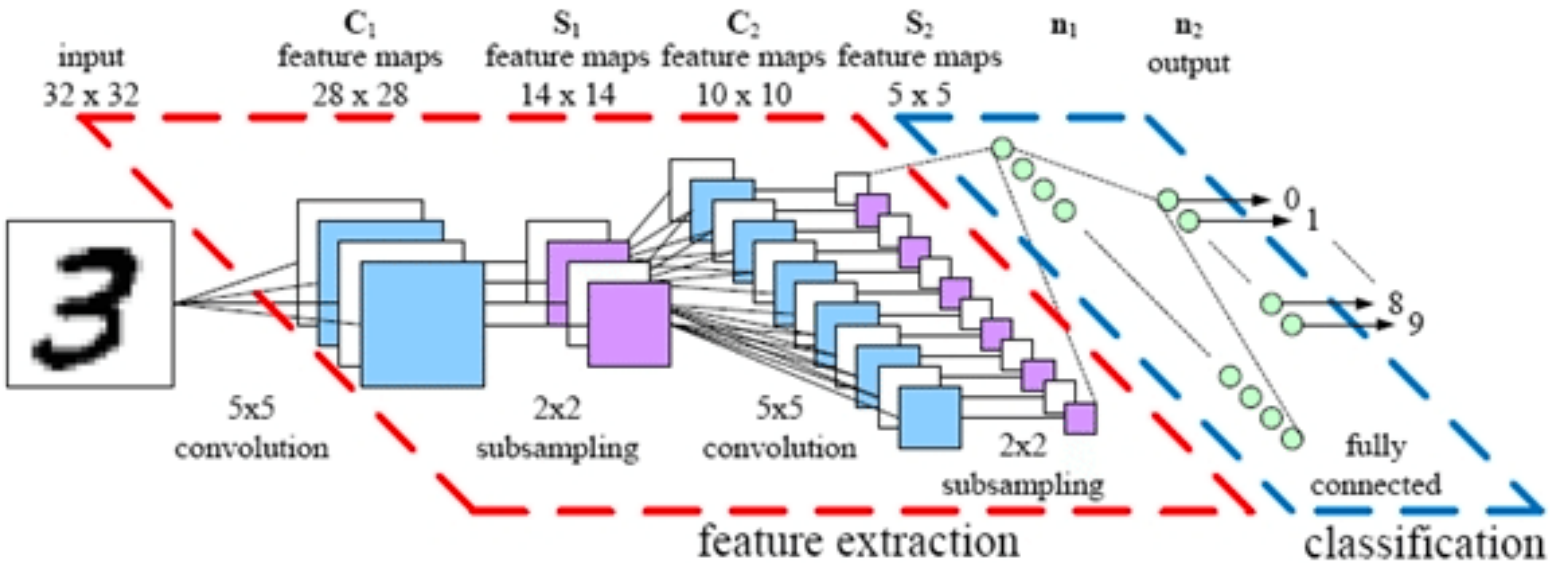
(Credit to Joaquin Vanschoren)

Automated Machine Learning



(Credit to Joaquin Vanschoren)

Automated Machine Learning



(Credit to Joaquin Vanschoren)

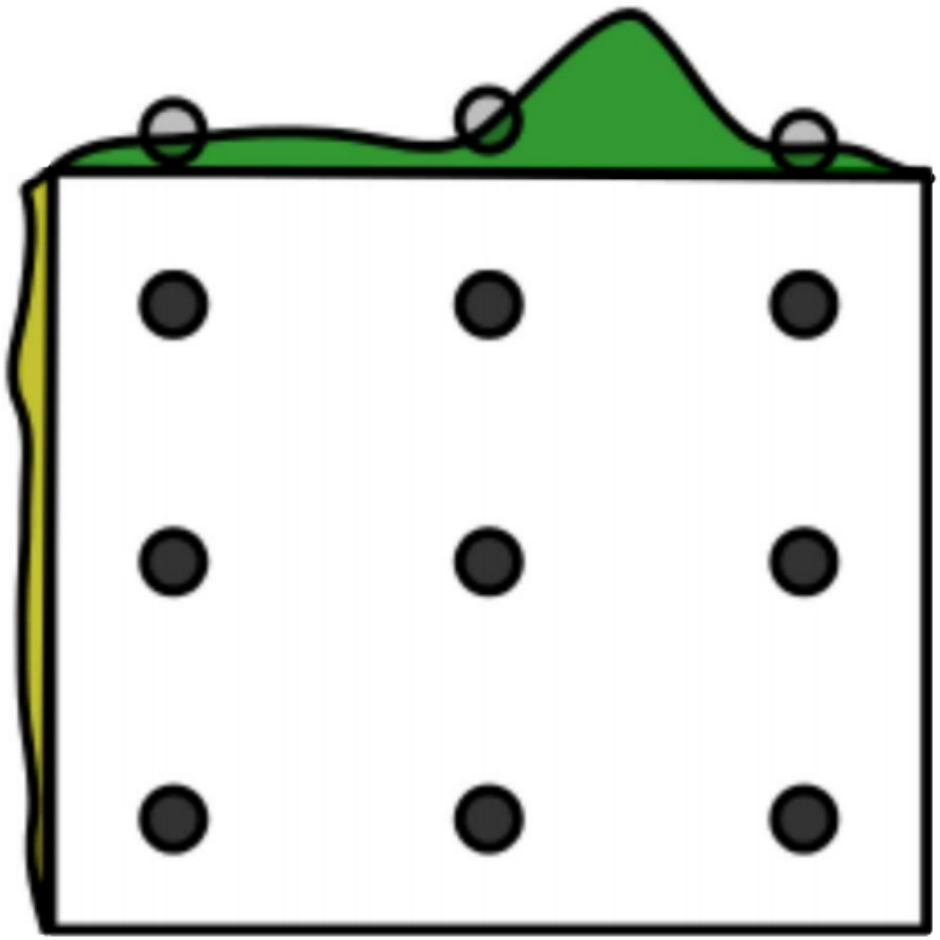
Hyperparameter Optimization

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

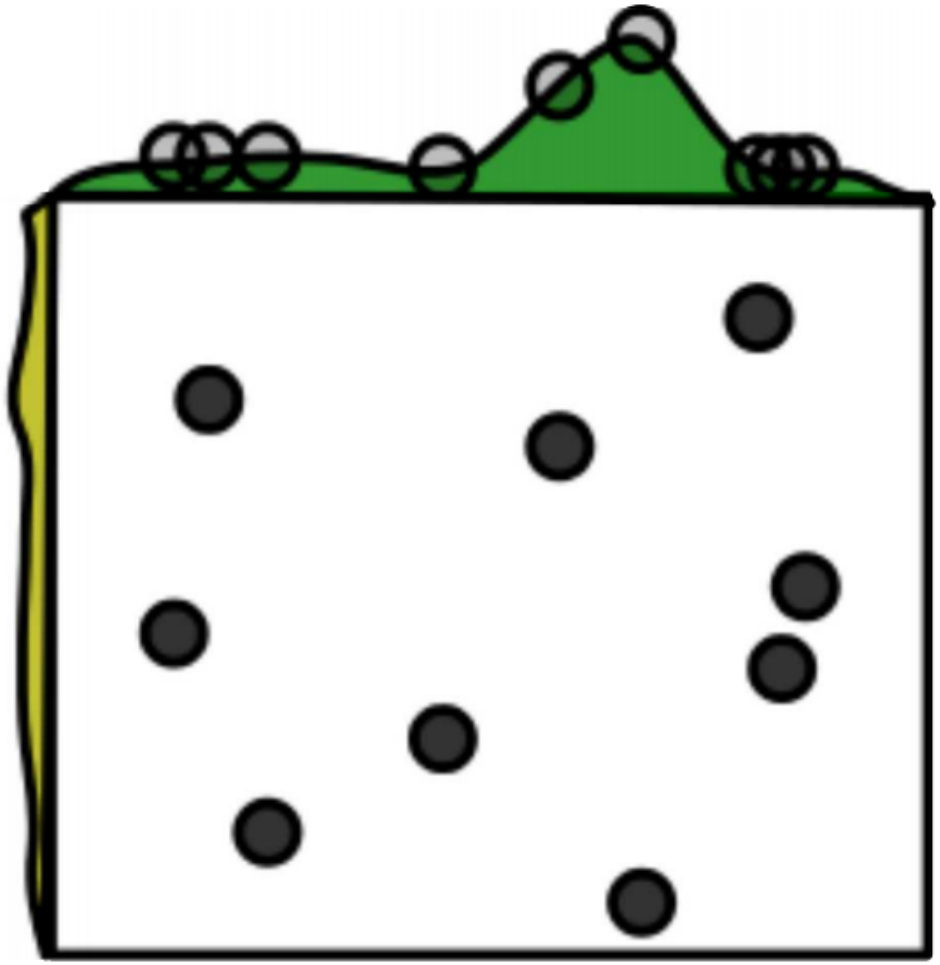
$$\boldsymbol{x}_\star \in \arg\min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{x})$$

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

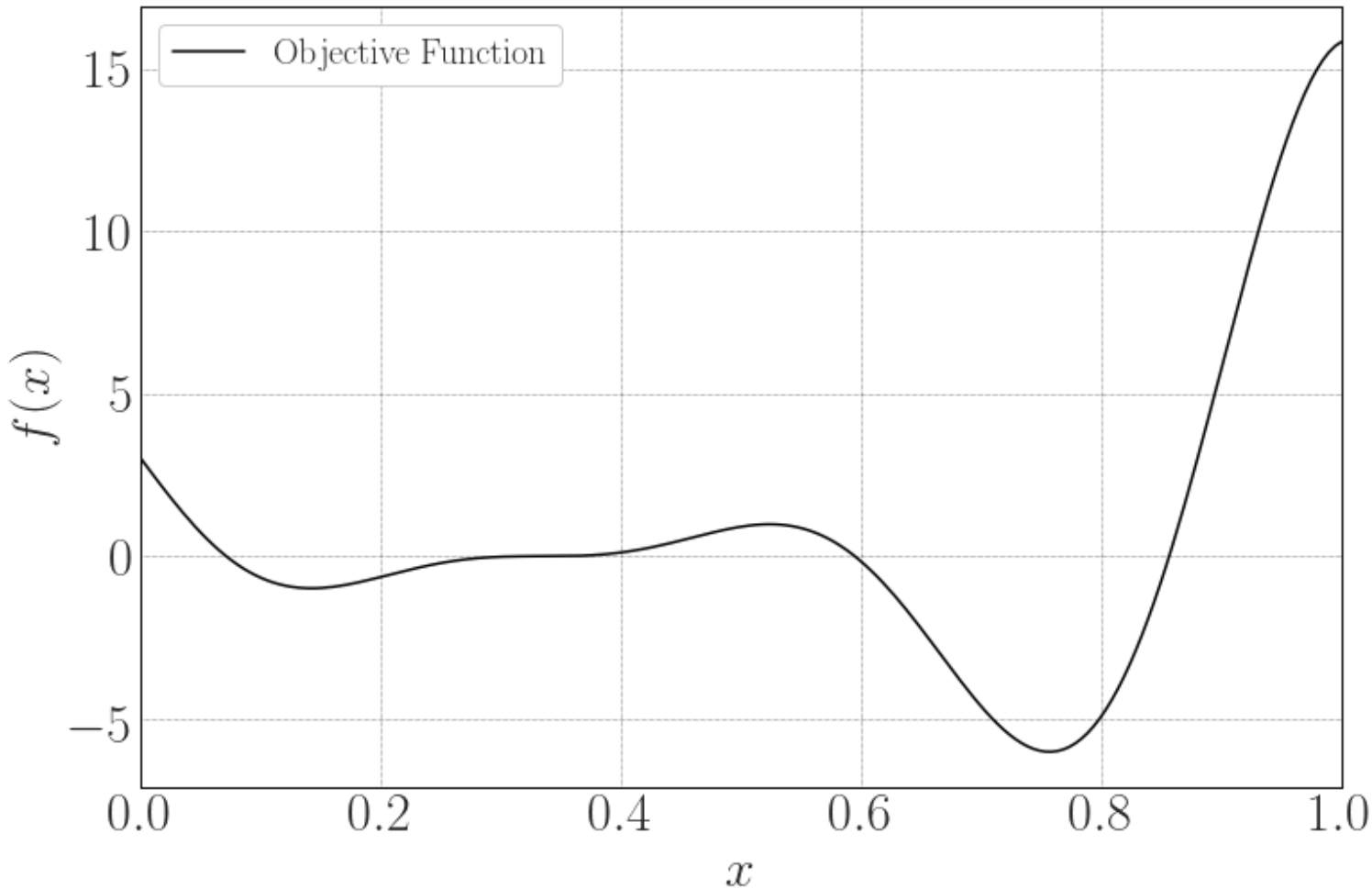
Recap Grid Search



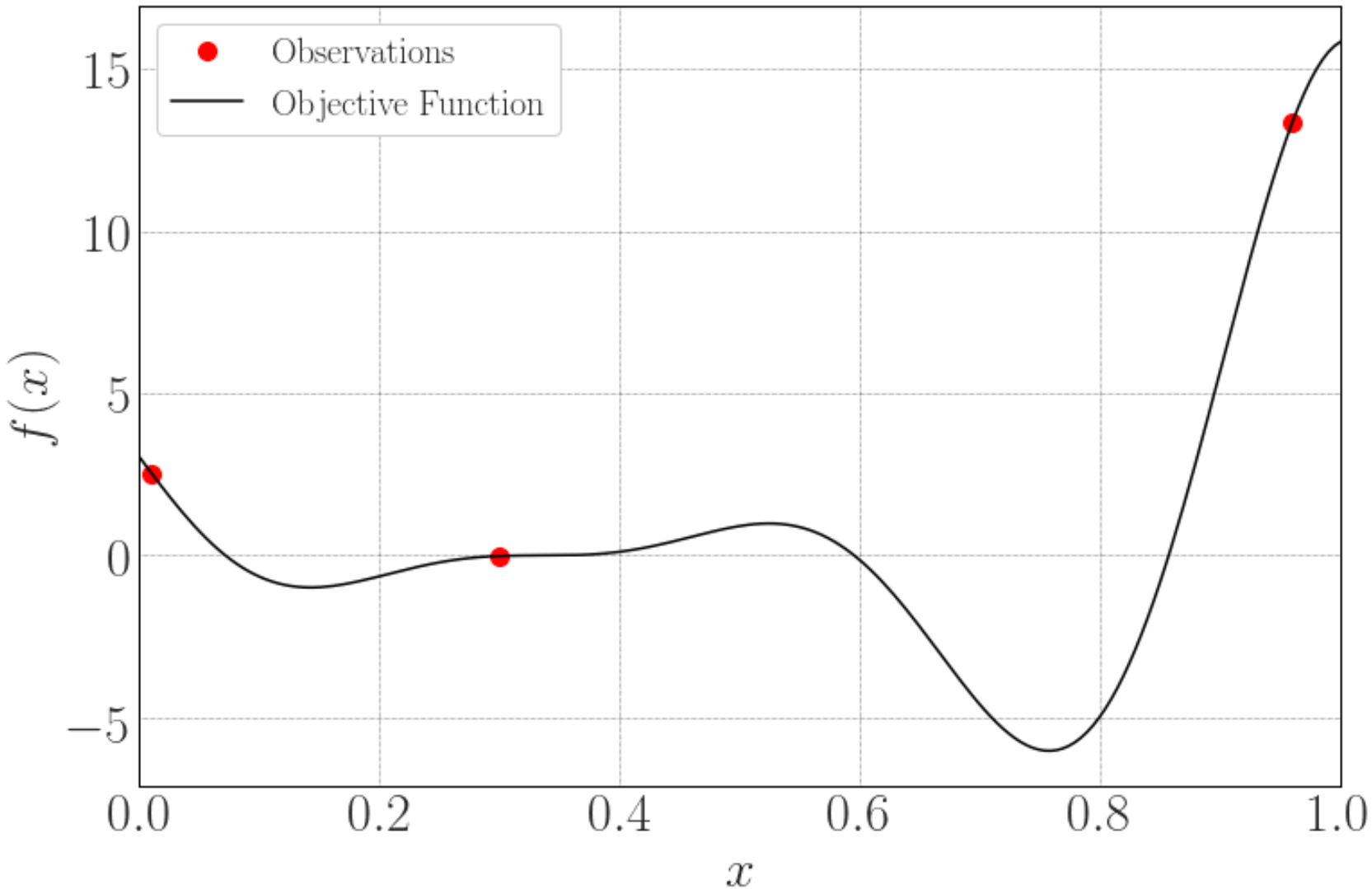
Recap Random Search



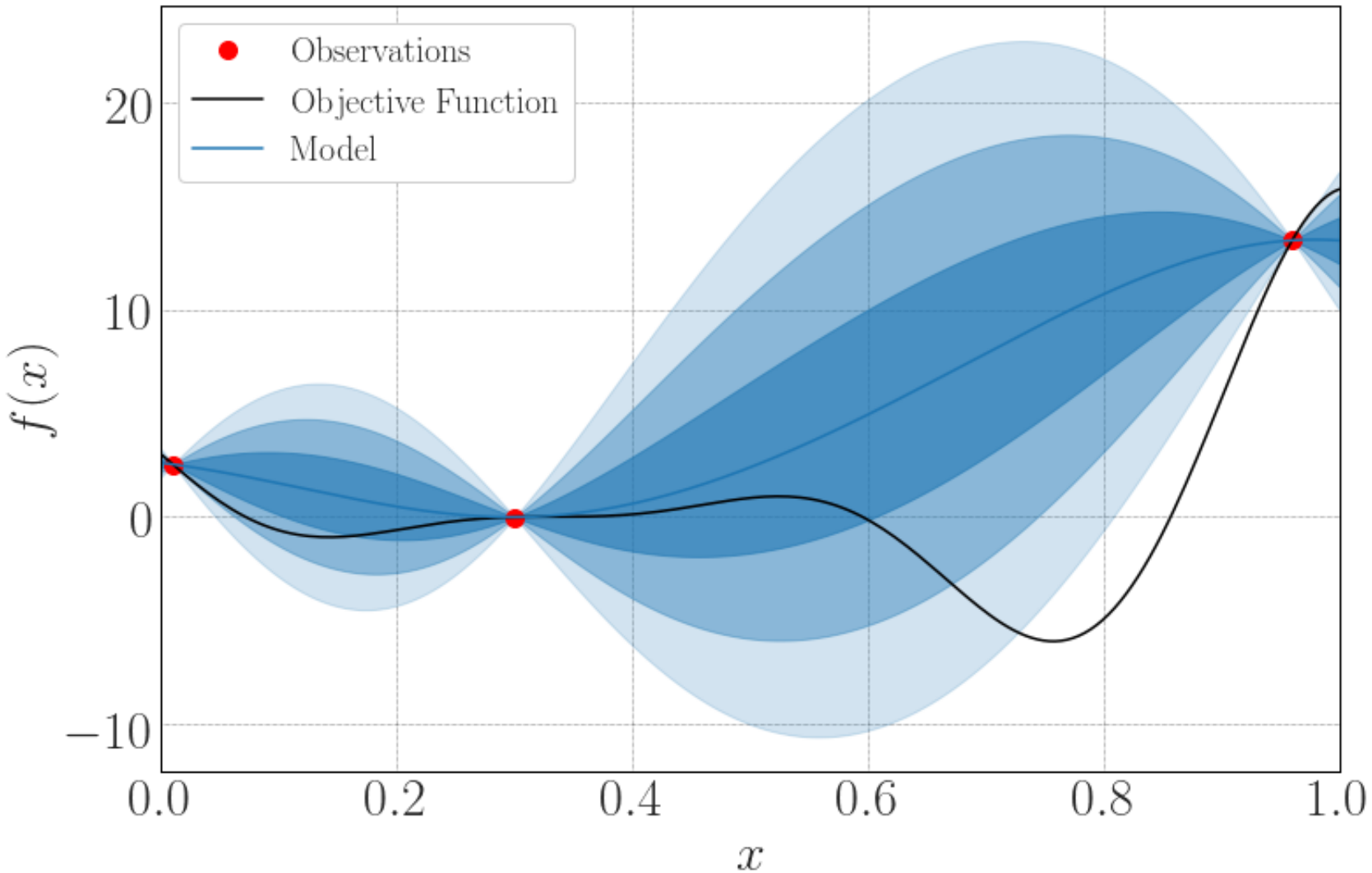
Model-Based optimization



Model-Based optimization



Model-Based optimization



Gaussian Process

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

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

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.

Random Forest

Use mean and variance of the individual tree predictions to approximate $p(f|D)$ (see [Hutter et al., 2011])
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

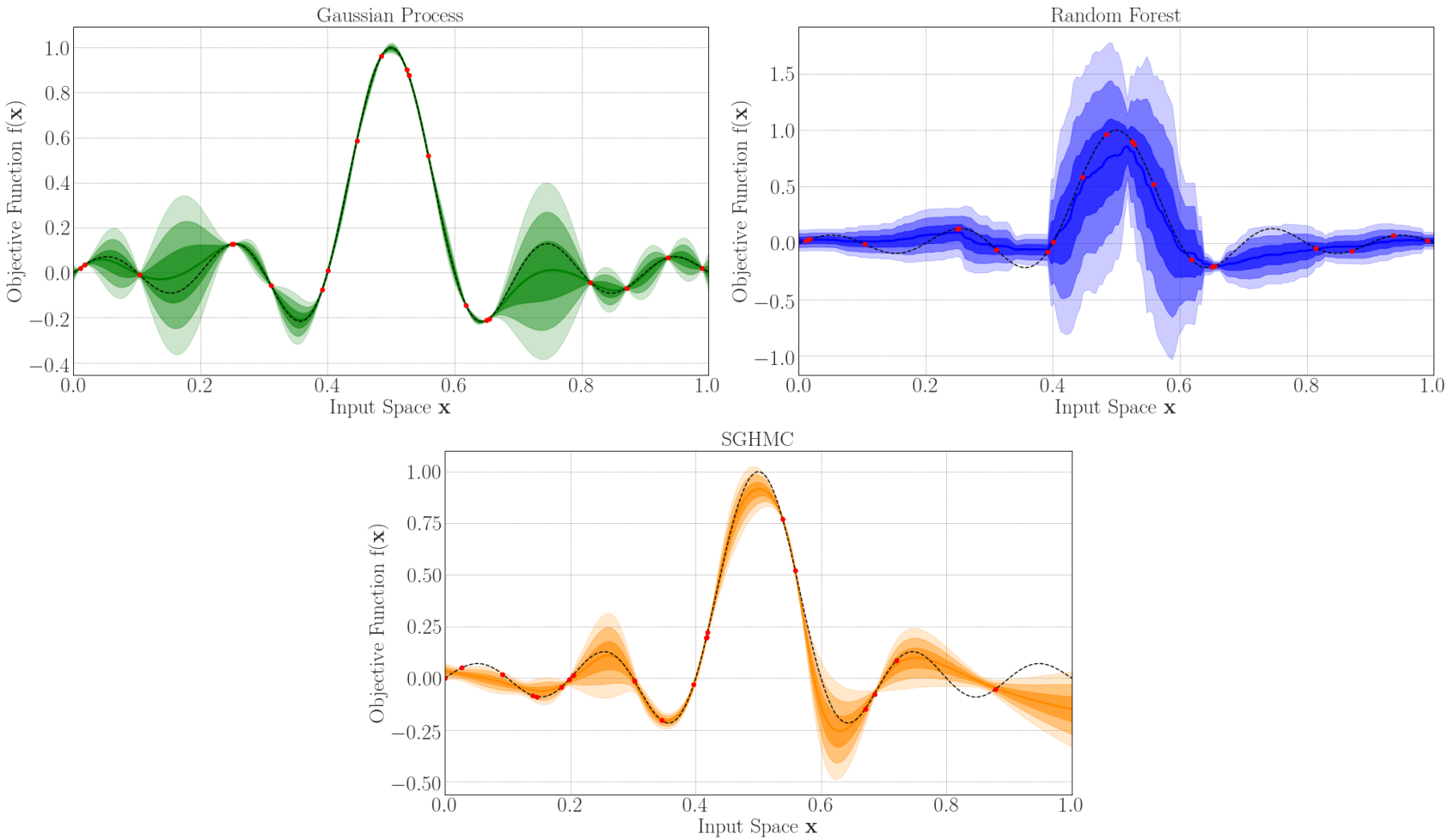
Instead of using one networks we generate multiple neural networks and use the mean and variance of the individual network predictions to approximate $p(f|D)$ (see [Springenberg et al., 2016, Snoek et al., 2015])
Pros:

- 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 process
- brittle against its own hyperparameters.

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 $D = \{(\boldsymbol{x}_0, y_0), \dots (\boldsymbol{x}_n, y_n)\}$ 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

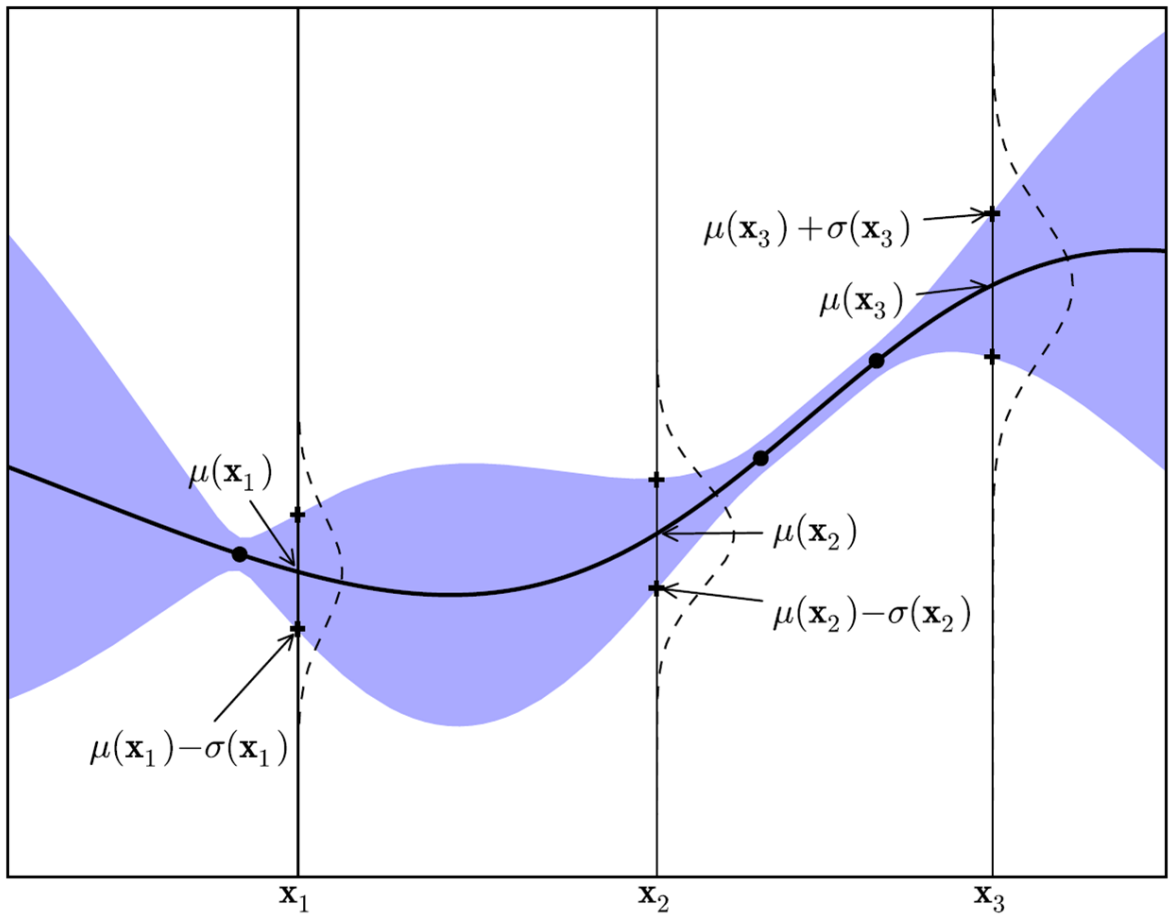
We use an acquisition function $a(\boldsymbol{x})$ that automatically trades off exploration and exploitation.
To find the next point $\boldsymbol{x}_{n+1}$ we numerically optimize $a(\boldsymbol{x})$:

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

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:

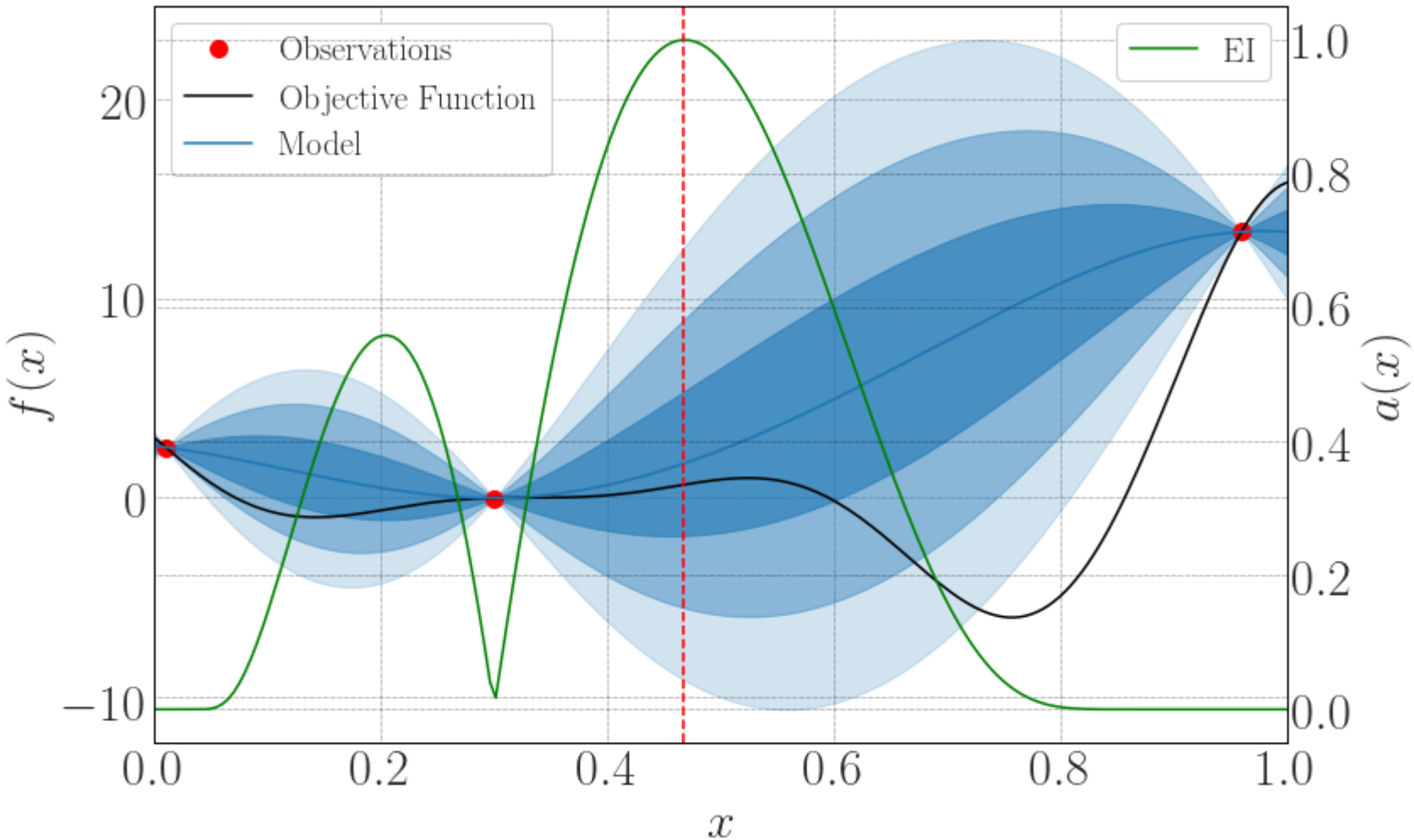
$$E_{p(f|D)}[\max(y_\star - f(x), 0)].$$

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:

$$\sigma(x)(\gamma(x)\Phi(\gamma(x))) + \phi(\gamma(x))$$

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

Acquisition Functions

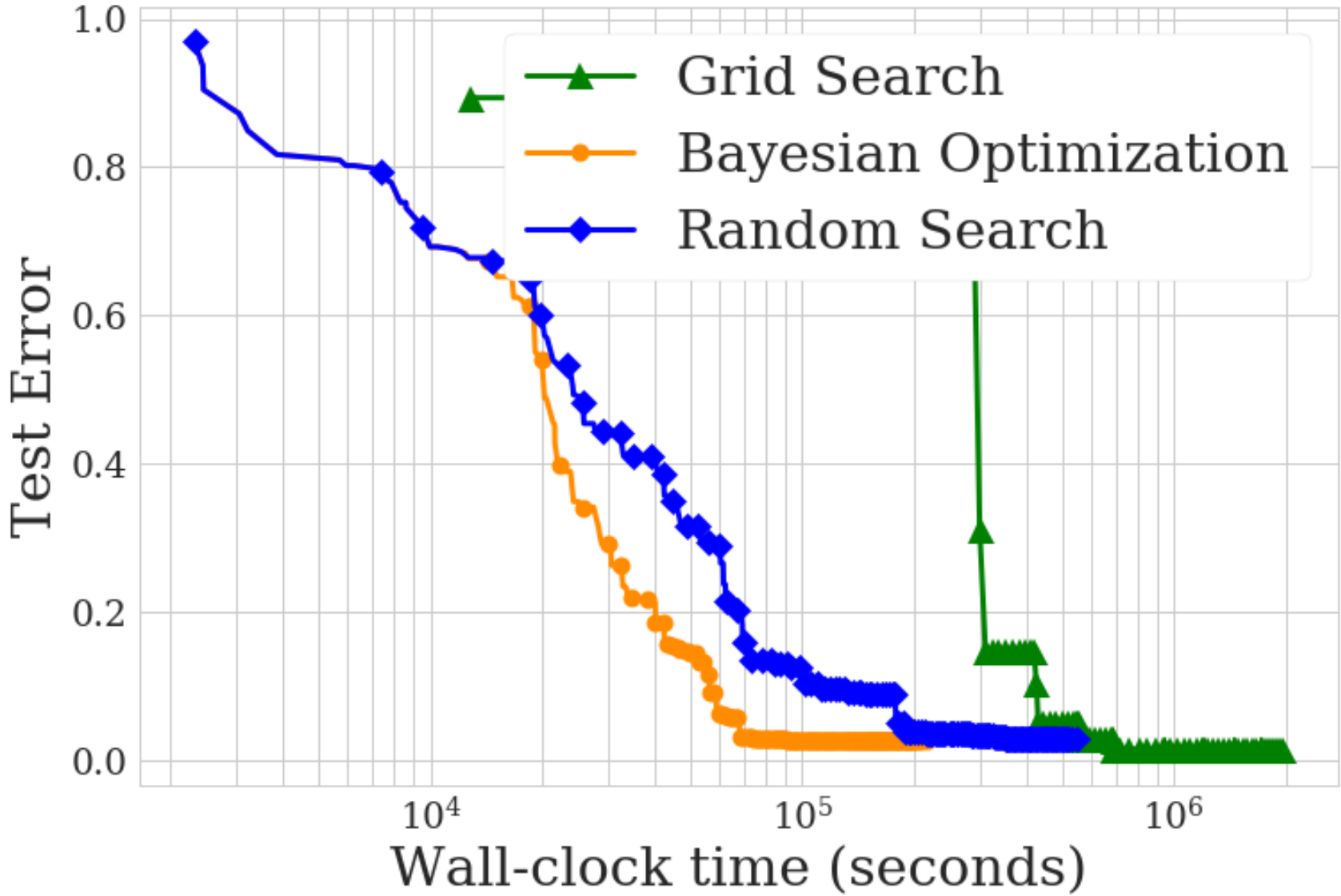


Bayesian Optimization [Jones et al., 1998]

Algorithm 1 Bayesian Optimization

1: Initialize data $\mathcal{D}_0$ using an initial design.
2: **for** $t = 1, 2, \ldots$ **do**
3: Fit probabilistic model for $f(\boldsymbol{x})$ on data $\mathcal{D}_{t-1}$
4: Choose $\boldsymbol{x}_t$ by maximizing the acquisition function $a_p(\boldsymbol{x})$
5: Evaluate $y_t \sim f(\boldsymbol{x}_t) + \mathcal{N}(0, \sigma^2)$, and augment the data: $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(\boldsymbol{x}_t, y_t)\}$
6: Choose incumbent $\hat{\boldsymbol{x}}_t \leftarrow \arg\min\{y_1, \ldots y_t\}$

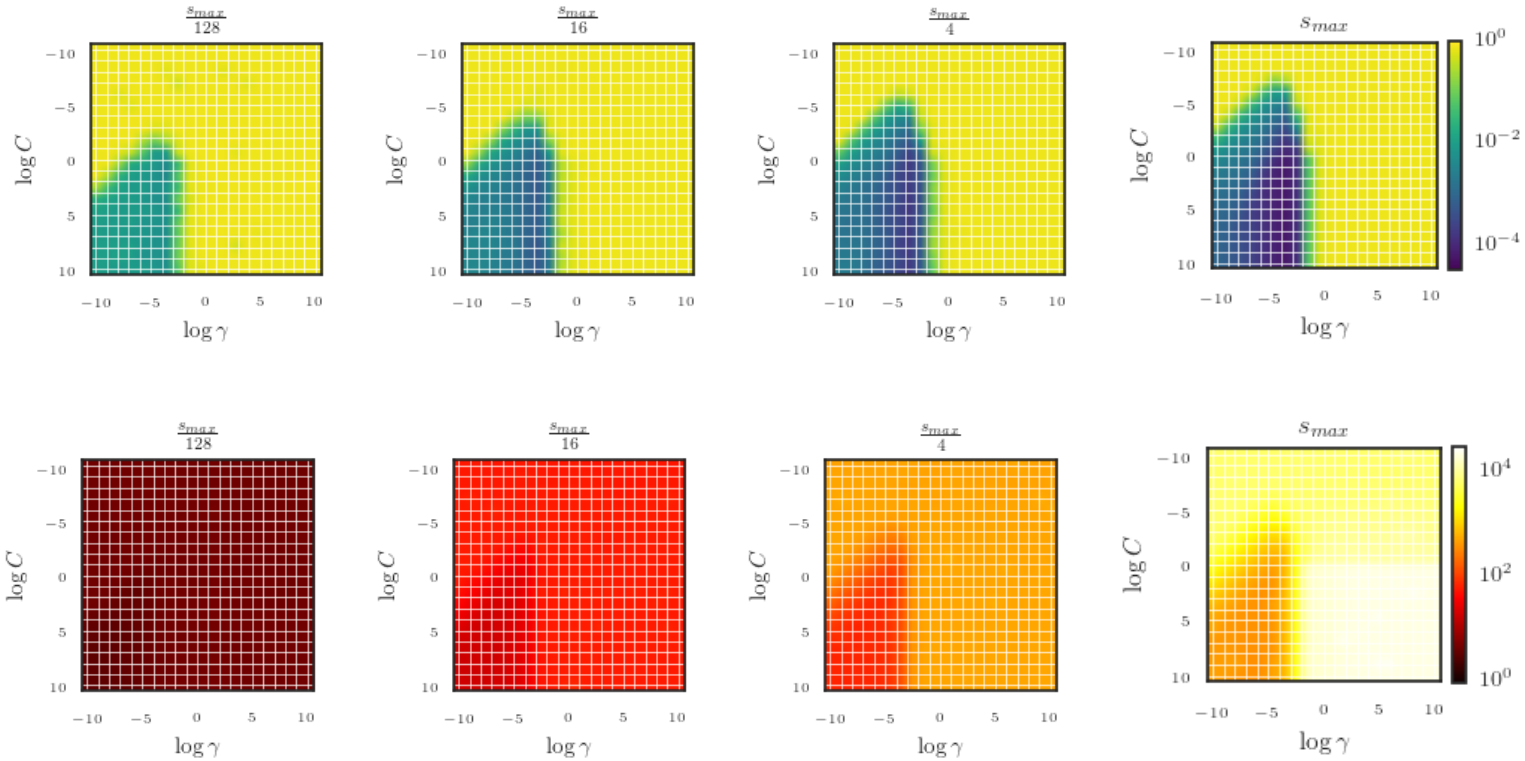
Bayesian Optimization



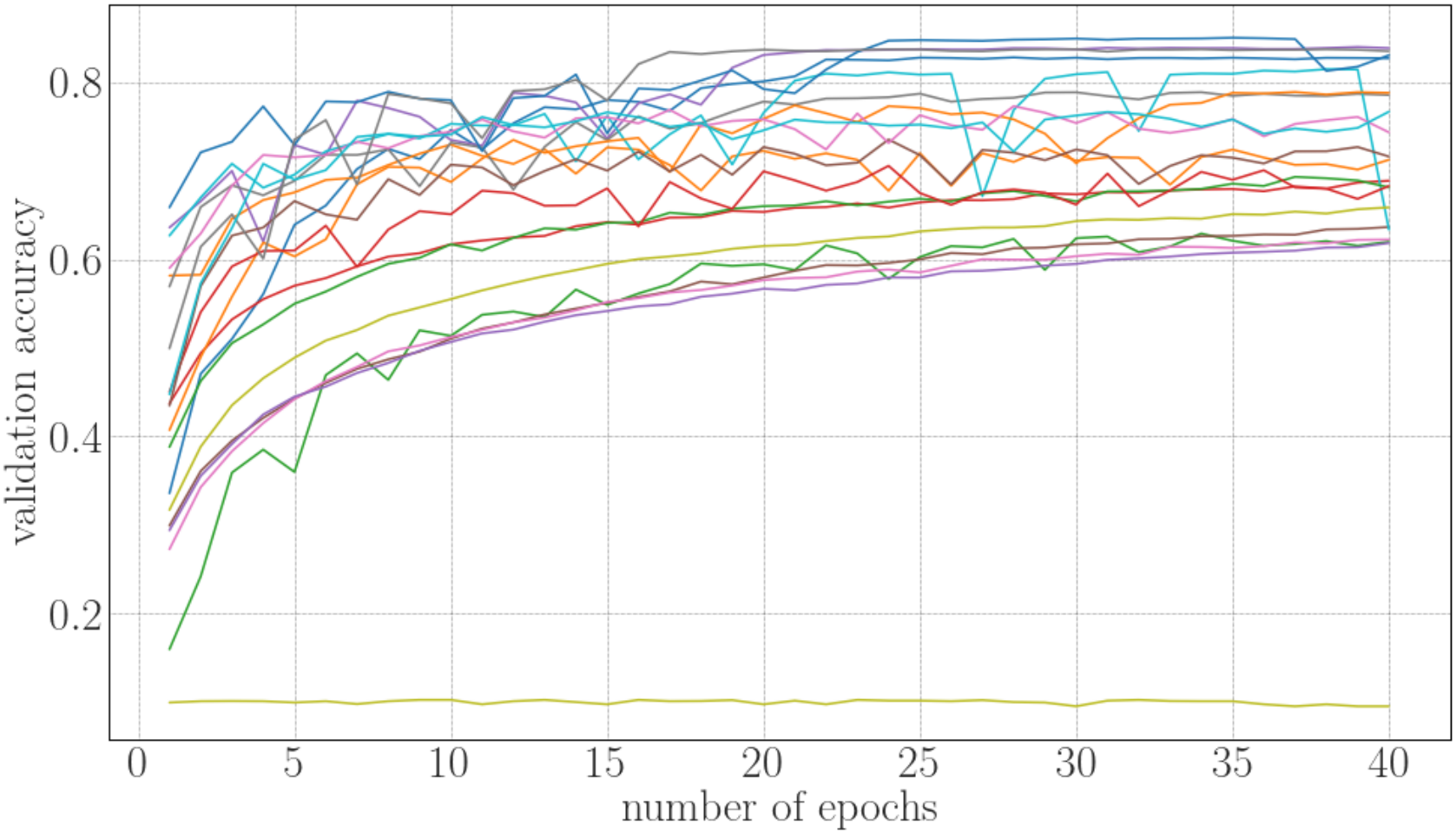
Multi-fidelity Optimization

- 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



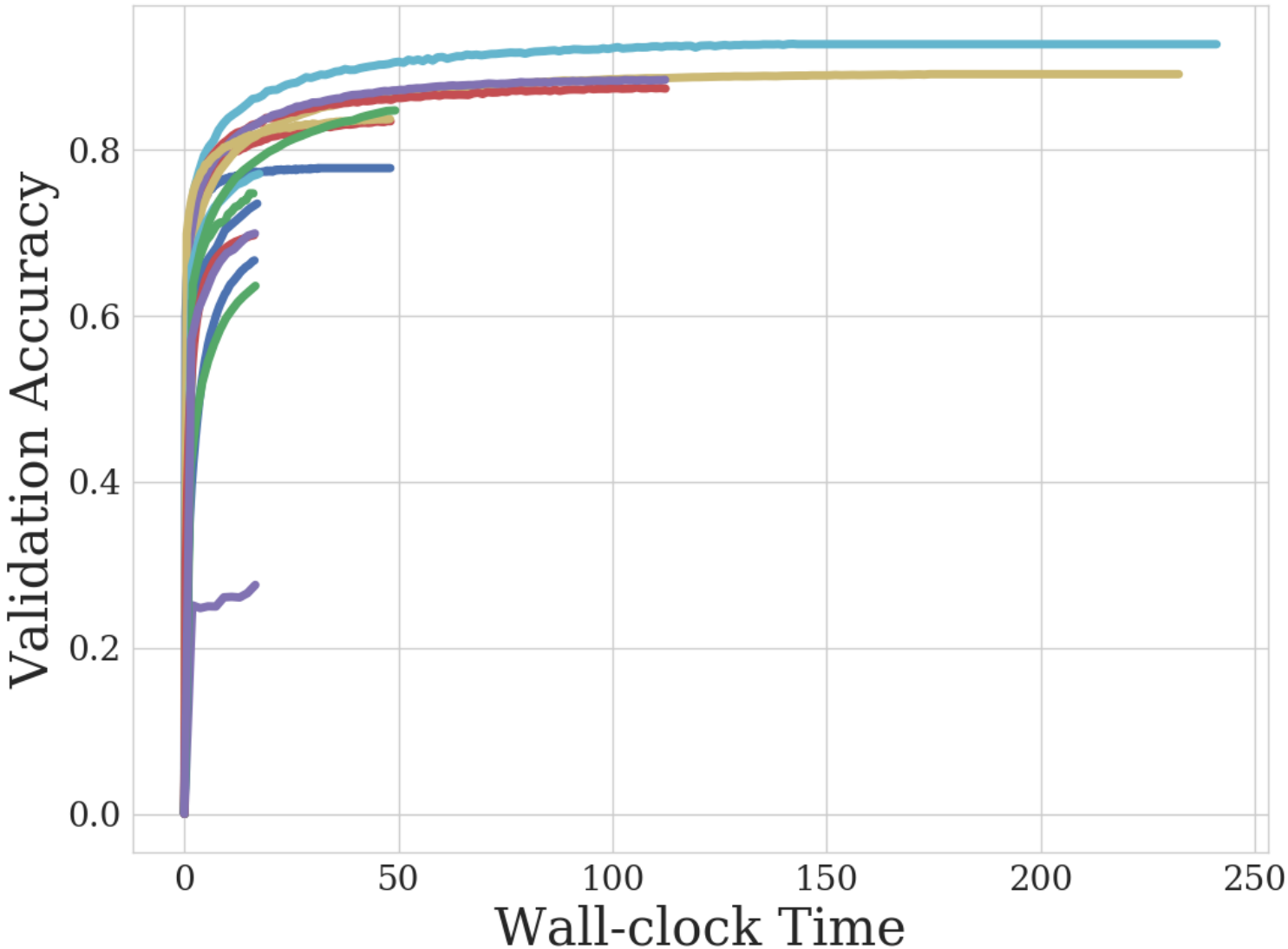
Successive Halving [Jamieson and Talwalkar, 2016]

Algorithm 2 Successive Halving

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

1: $b = b_0$
2: **while** $b \leq b_{max}$ **do**
3: $\quad L = \{\tilde{f}(c, b) : c \in C\}$
4: $\quad C = \text{top}_\text{k}(C, L, \lfloor |C|/\eta \rfloor)$
5: $\quad b = \eta \cdot b$

Successive Halving



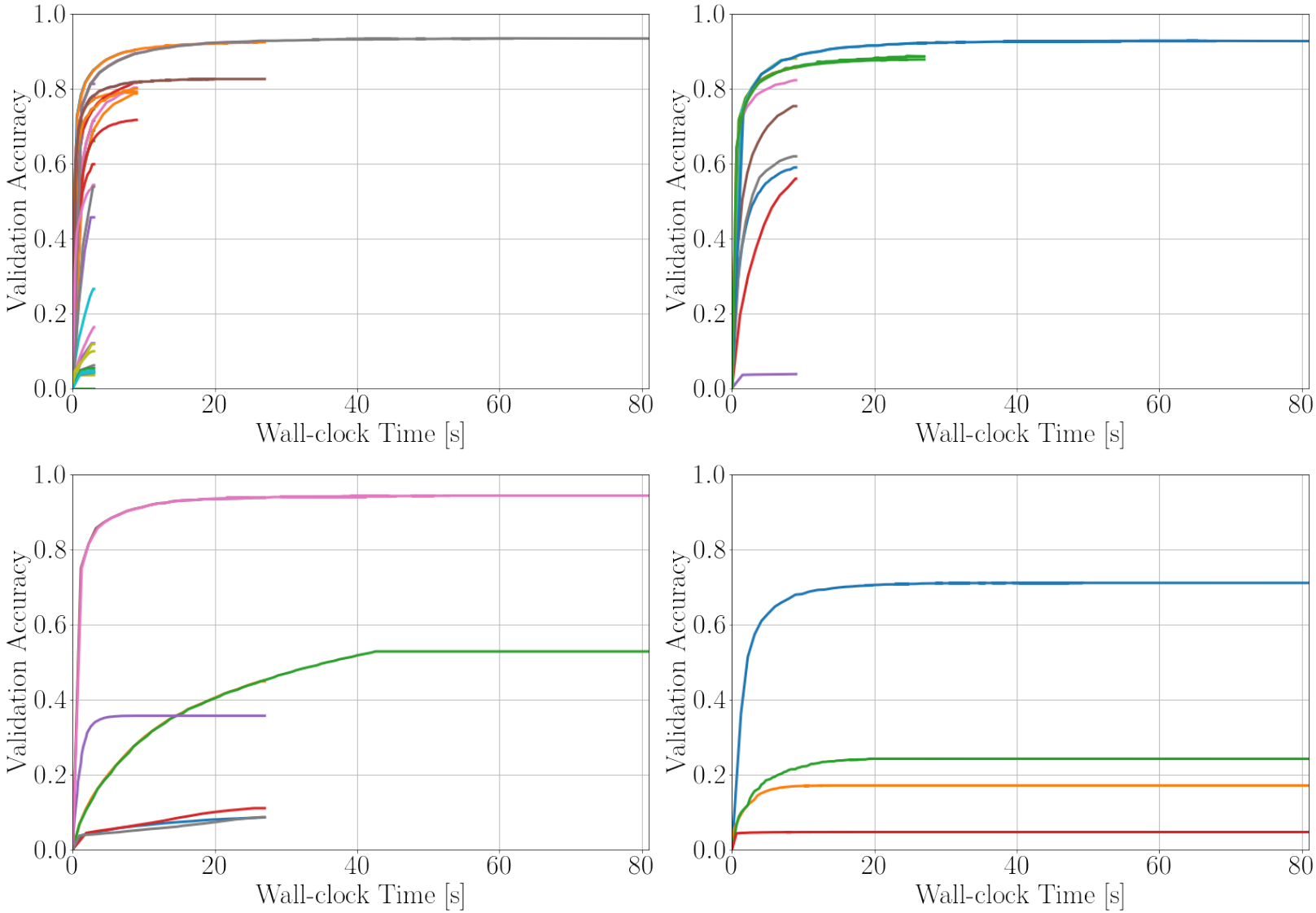
Hyperband [Li et al., 2017]

Algorithm 3 Hyperband

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

1: $s_{max} = \lfloor \log_\eta \frac{b_{max}}{b_{min}} \rfloor$
2: **for** $s \in \{s_{max}, s_{max} - 1, \ldots, 0\}$ **do**
3: sample $n = \lceil \frac{s_{max} + 1}{s + 1} \cdot \eta^s \rceil$ configurations
4: 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

Combining Hyperband with 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]

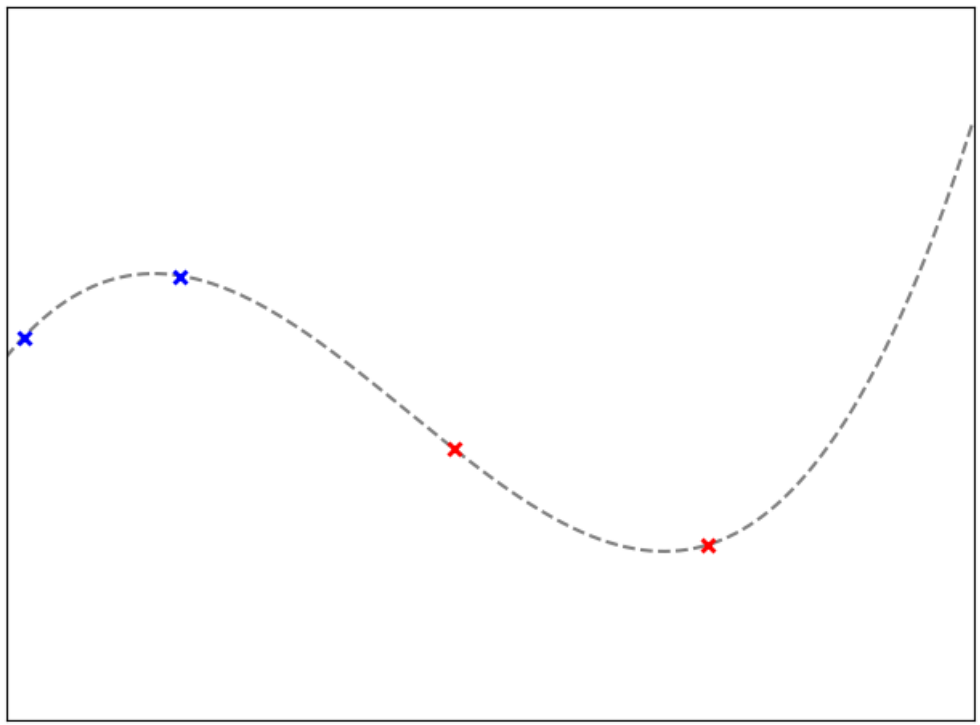
- non-parametric KDE for $p(\vec{x})$ instead of Gaussian Processes modelling $p(y|\vec{x})$
- equivalent to expected improvement

\+ efficient $\mathcal{O}(N \cdot d)$

\+ complex search spaces with priors

\+ parallelizable

\- not as sample efficient as GPs



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

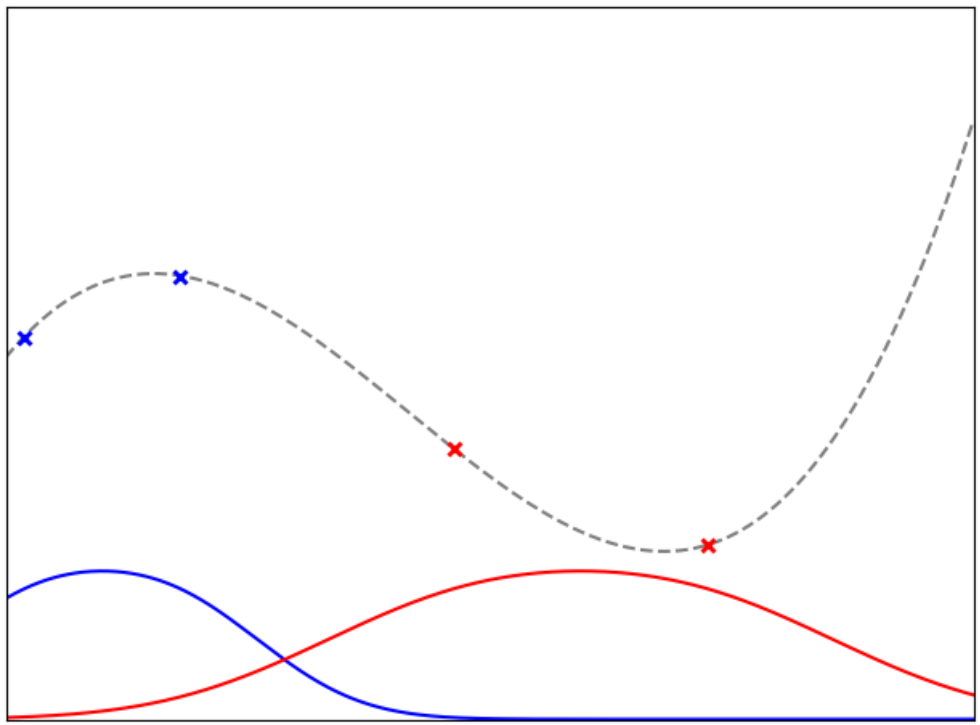
- non-parametric KDE for $p(\vec{x})$ instead of Gaussian Processes modelling $p(y|\vec{x})$
- equivalent to expected improvement

\+ efficient $\mathcal{O}(N \cdot d)$

\+ complex search spaces with priors

\+ parallelizable

\- not as sample efficient as GPs



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

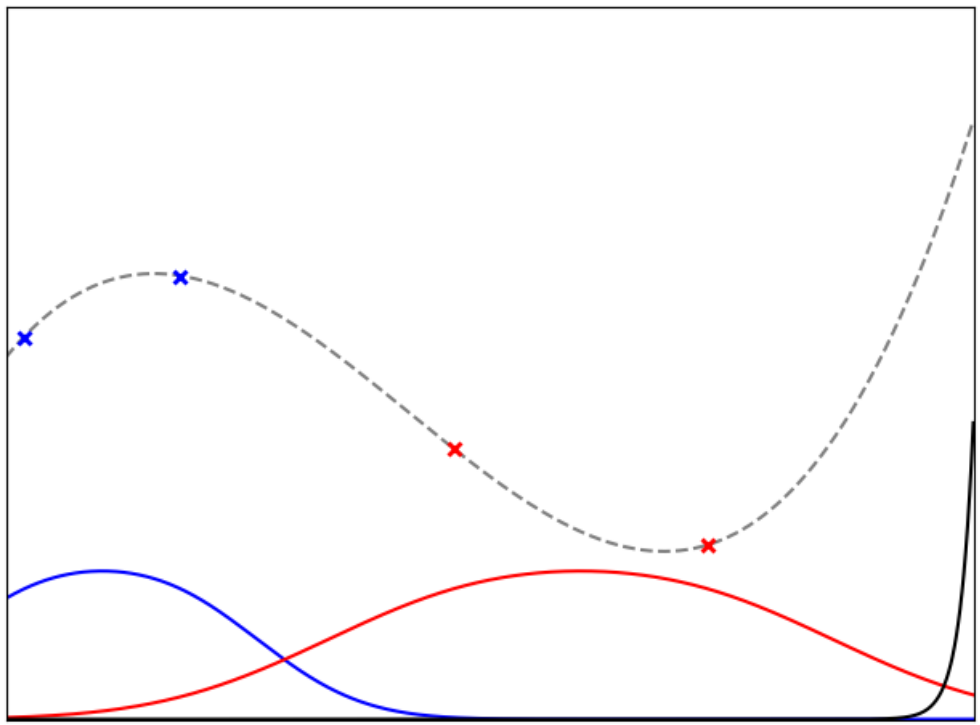
- non-parametric KDE for $p(\vec{x})$ instead of Gaussian Processes modelling $p(y|\vec{x})$
- equivalent to expected improvement

\+ efficient $\mathcal{O}(N \cdot d)$

\+ complex search spaces with priors

\+ parallelizable

\- not as sample efficient as GPs



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

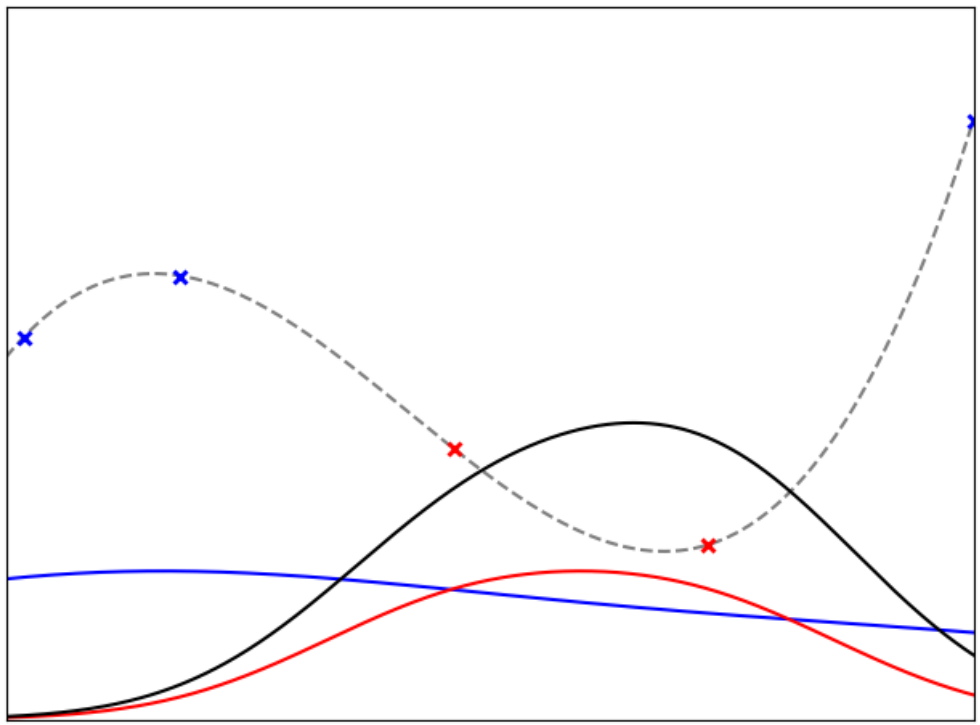
- non-parametric KDE for $p(\vec{x})$ instead of Gaussian Processes modelling $p(y|\vec{x})$
- equivalent to expected improvement

\+ efficient $\mathcal{O}(N \cdot d)$

\+ complex search spaces with priors

\+ parallelizable

\- not as sample efficient as GPs



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

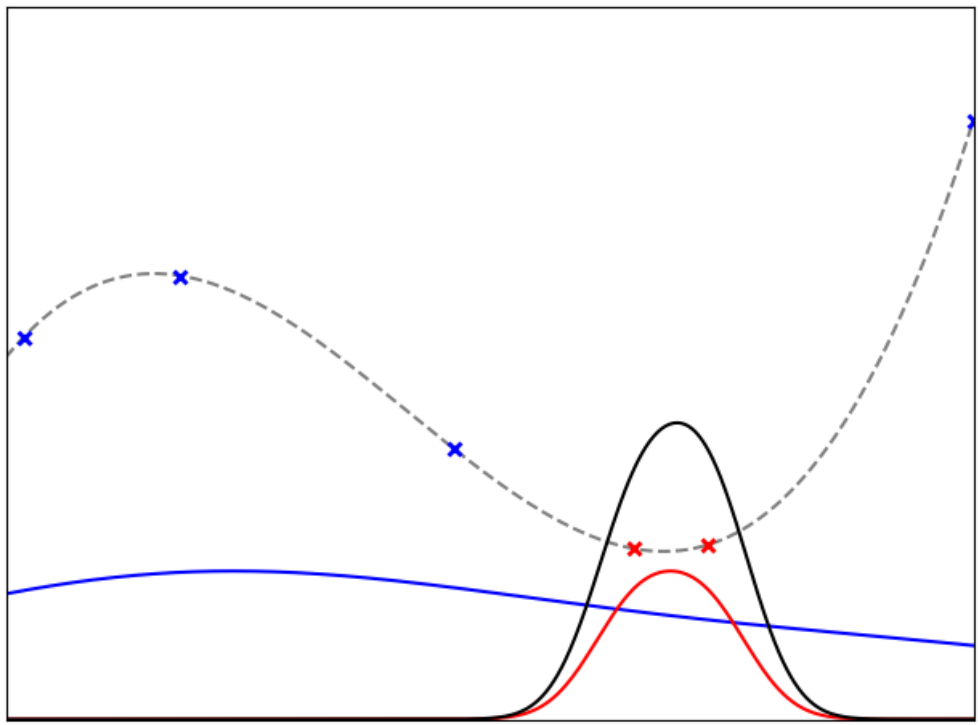
- non-parametric KDE for $p(\vec{x})$ instead of Gaussian Processes modelling $p(y|\vec{x})$
- equivalent to expected improvement

\+ efficient $\mathcal{O}(N \cdot d)$

\+ complex search spaces with priors

\+ parallelizable

\- not as sample efficient as GPs



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.

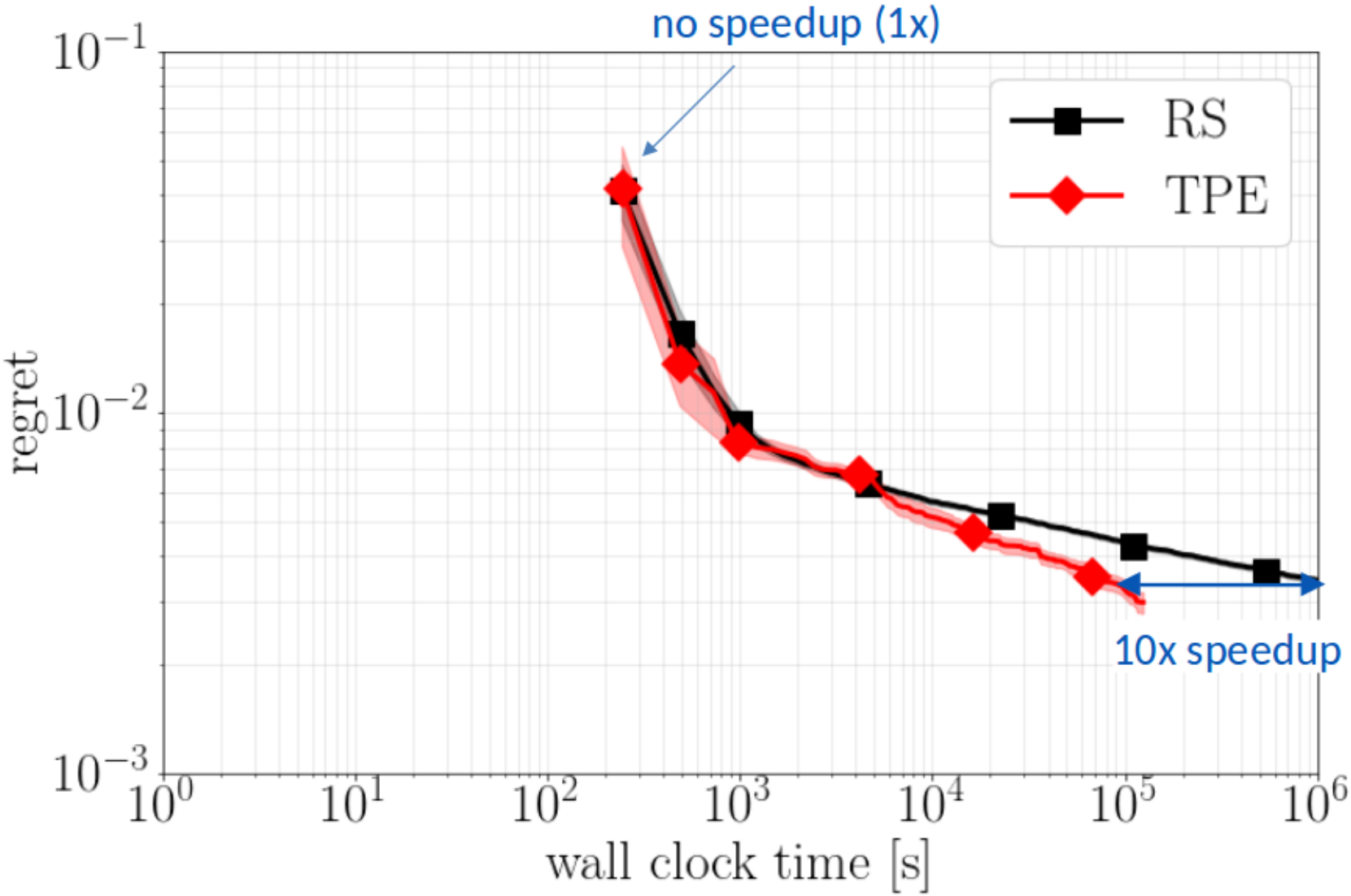
BOHB [Falkner et al., 2018]

Algorithm 4 Pseudocode for sampling in BOHB

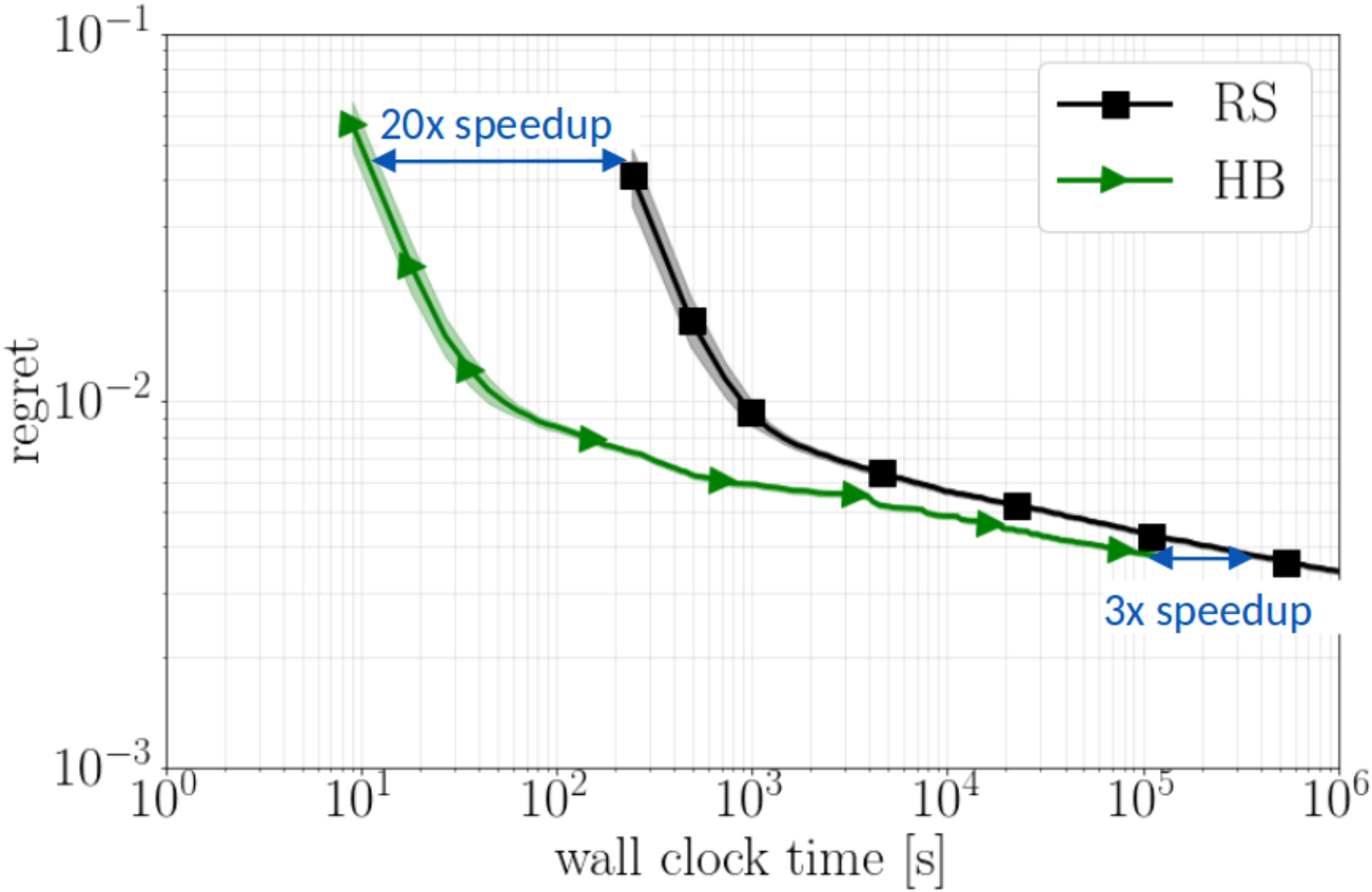
Require: observations D, fraction of random runs ρ, percentile q, number of samples N_s, minimum number of points N_{min} to build a model, and bandwidth factor b_w

if rand() $\leq \rho$ **then**
    return random configuration
$b = \arg\max \{D_b : |D_b| \geq N_{min} + 2\}$
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 $l(\boldsymbol{x})/g(\boldsymbol{x})$

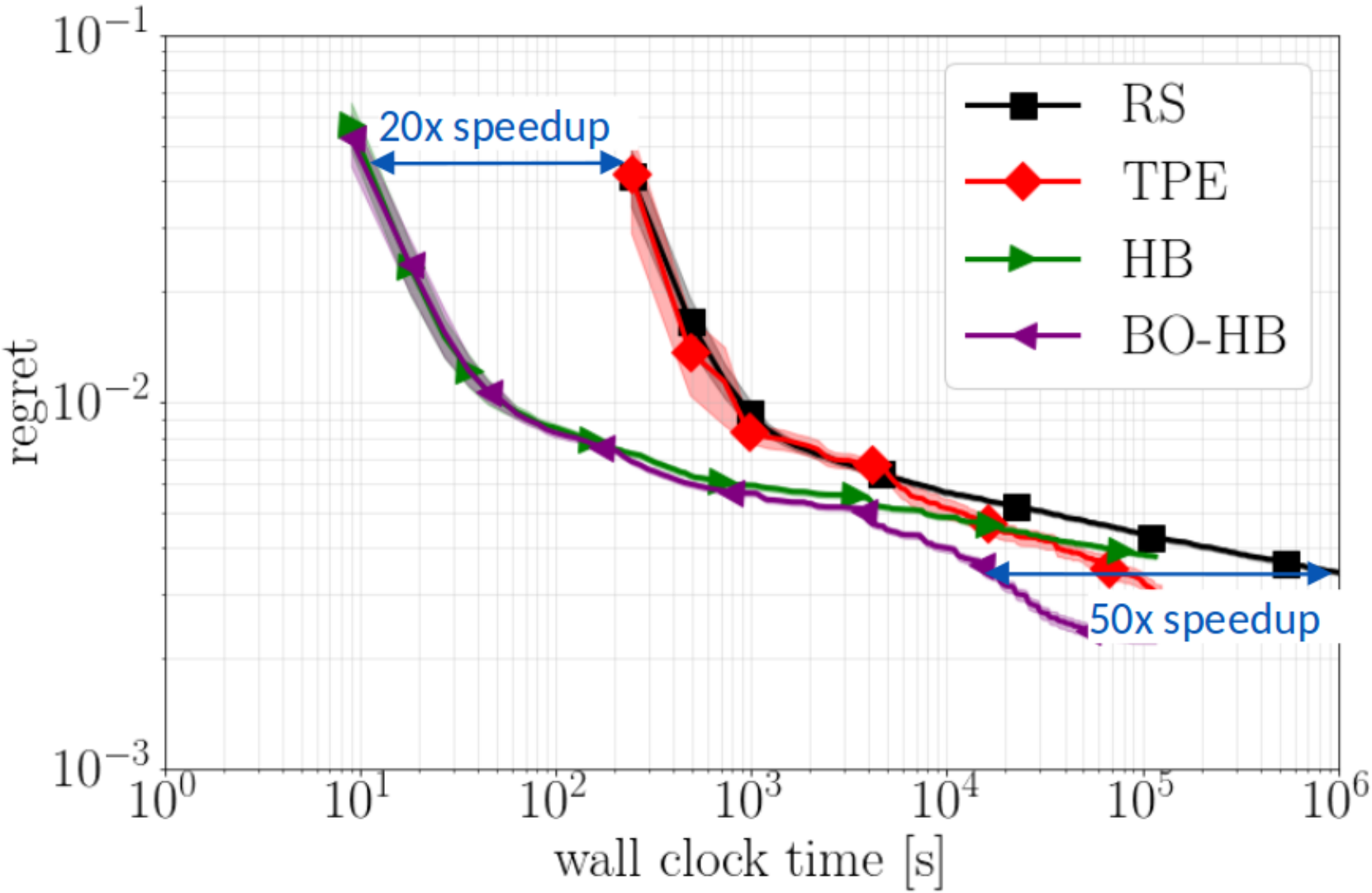
BOHB


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


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]
RS
HB
20x speedup
3x speedup

BOHB


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]
20x speedup
50x speedup
RS
TPE
HB
BO-HB

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

References I

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011).

Algorithms for hyper-parameter optimization.

In *Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NIPS'11)*, pages 2546–2554.

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)*, pages 1436–1445.

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)*, pages 507–523.

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

Non-stochastic best arm identification and hyperparameter optimization.

In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS)*.

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

Efficient global optimization of expensive black box functions.

Journal of Global Optimization, 13:455–492.

References II

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

Fast bayesian hyperparameter optimization on large datasets.

In *Electronic Journal of Statistics*, volume 11.

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

Hyperband: Bandit-based configuration evaluation for hyperparameter optimization.

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

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)*.

Springenberg, J., Klein, A., S.Falkner, and Hutter, F. (2016).

Bayesian optimization with robust bayesian neural networks.

In *Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NIPS'16)*.

References III

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)*.