# 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:

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

# Global Blackbox Optimization

- Consider the global optimization problem of finding:

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

- 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.

# Global Blackbox Optimization

- Consider the global optimization problem of finding:

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

- 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

# Global Blackbox Optimization

- Consider the global optimization problem of finding:

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

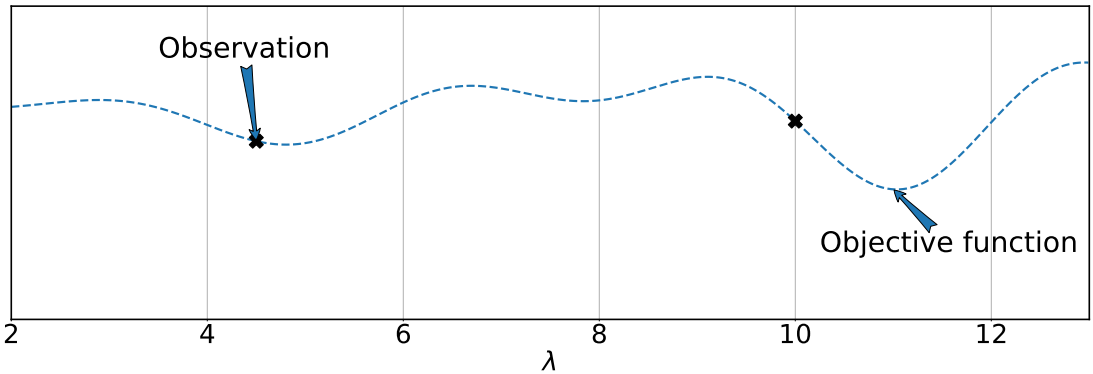
- In the most general form, function $f$ is a blackbox function:


  - 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.

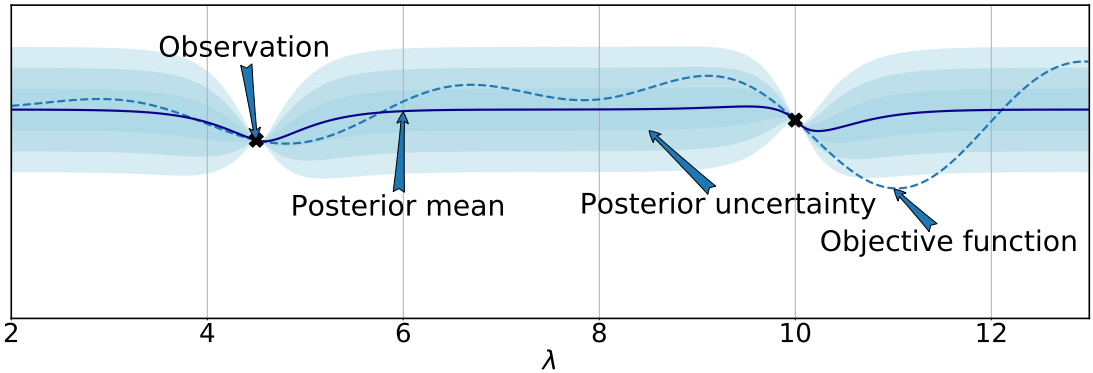
- Today, we'll discuss a Bayesian approach for solving such blackbox optimization problems

- 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

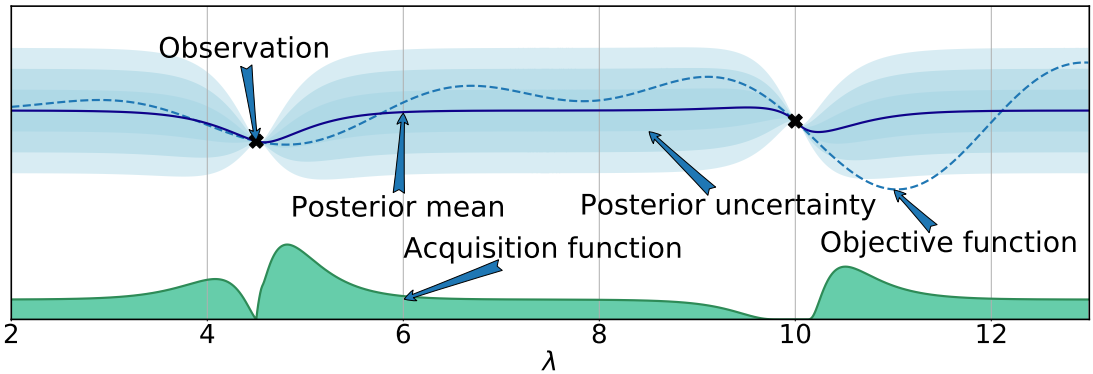
# Bayesian Optimization of a blackbox function in a nutshell



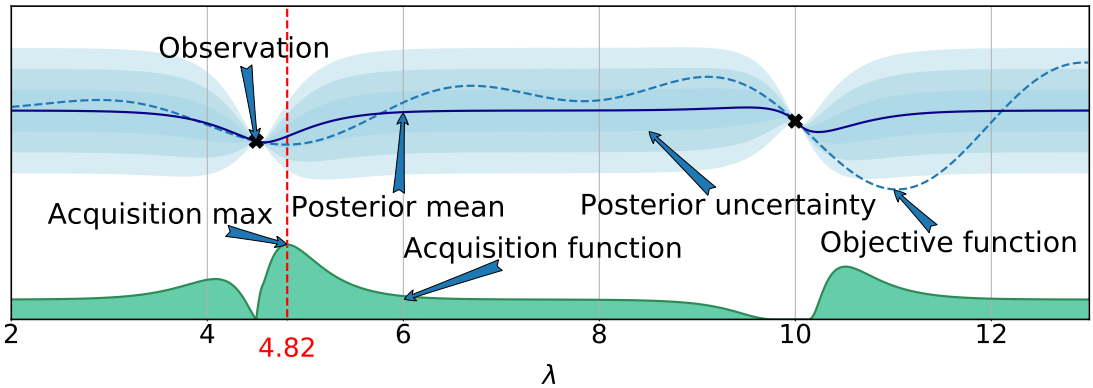
# Bayesian Optimization of a blackbox function in a nutshell



# Bayesian Optimization of a blackbox function in a nutshell



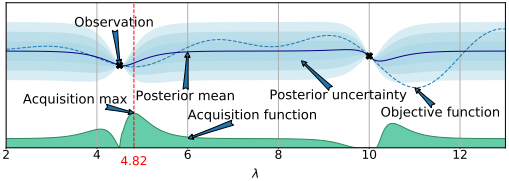
# Bayesian Optimization of a blackbox function in a nutshell



# Bayesian Optimization of a blackbox function in a nutshell

General approach

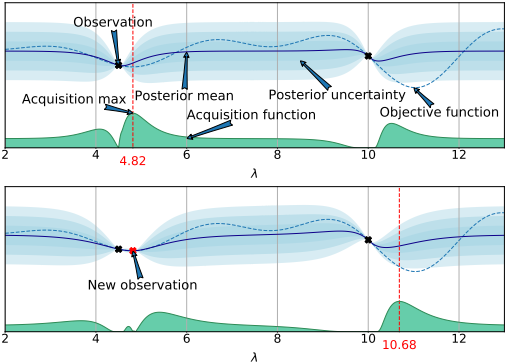
- Fit a probabilistic model to the collected function samples $\langle \boldsymbol{\lambda}, c(\boldsymbol{\lambda}) \rangle$
- Use the model to guide optimization, trading off exploration *vs* exploitation



# Bayesian Optimization of a blackbox function in a nutshell

General approach

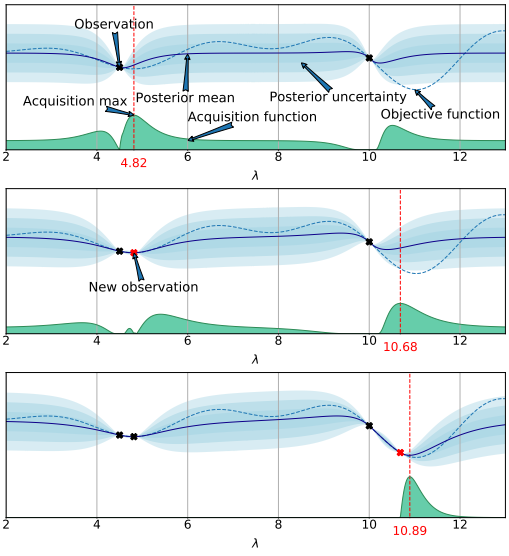
- Fit a probabilistic model to the collected function samples $\langle \boldsymbol{\lambda}, c(\boldsymbol{\lambda}) \rangle$
- Use the model to guide optimization, trading off exploration *vs* exploitation



# Bayesian Optimization of a blackbox function in a nutshell

General approach

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



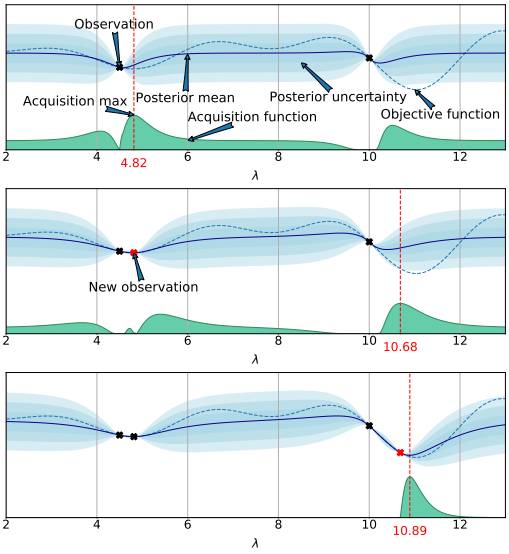
# Bayesian Optimization of a blackbox function in a nutshell

General approach

- Fit a probabilistic model to the collected function samples $\langle \boldsymbol{\lambda}, c(\boldsymbol{\lambda}) \rangle$
- 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 $\mathbf{\Lambda}$, cost function $c$, acquisition function $u$, predictive model $\hat{c}$, maximal number of function evaluations $T$

**Result :** Best configuration $\hat{\boldsymbol{\lambda}}$ (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: $\boldsymbol{\lambda}^{(t)} \in \arg\max_{\boldsymbol{\lambda} \in \mathbf{\Lambda}} u(\boldsymbol{\lambda}; \mathcal{D}^{(t-1)}, \hat{c}^{(t)})$

5 Query $c(\boldsymbol{\lambda}^{(t)})$

6 Update data: $\mathcal{D}^{(t)} \leftarrow \mathcal{D}^{(t-1)} \cup \{\langle \boldsymbol{\lambda}^{(t)}, c(\boldsymbol{\lambda}^{(t)}) \rangle\}$

# 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:

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

# 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:

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

- 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

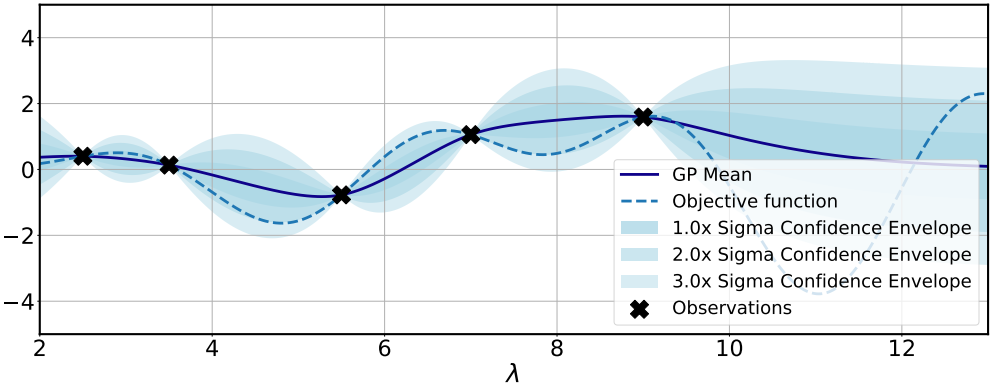
# Acquisition Functions: the Basics

- 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 $\boldsymbol{\lambda}^{(t)} \in \boldsymbol{\Lambda}$ next

# Acquisition Functions: the Basics

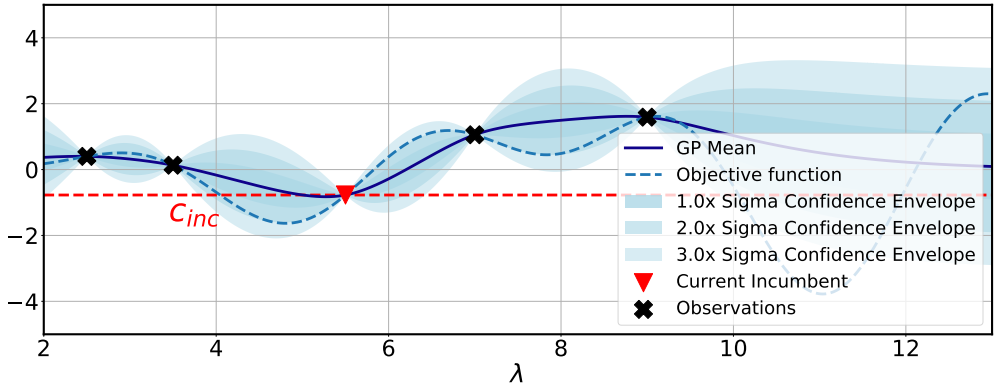
- Given the surrogate model $\hat{c}^{(t)}$ at the $t$-th iteration of BO, the acquisition function $u(\cdot)$ judges the utility (or usefulness) of evaluating $f$ at $\boldsymbol{\lambda}^{(t)} \in \boldsymbol{\Lambda}$ next
- The acquisition function needs to trade off exploration and exploitation
  - E.g., just picking the $\boldsymbol{\lambda}$ 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



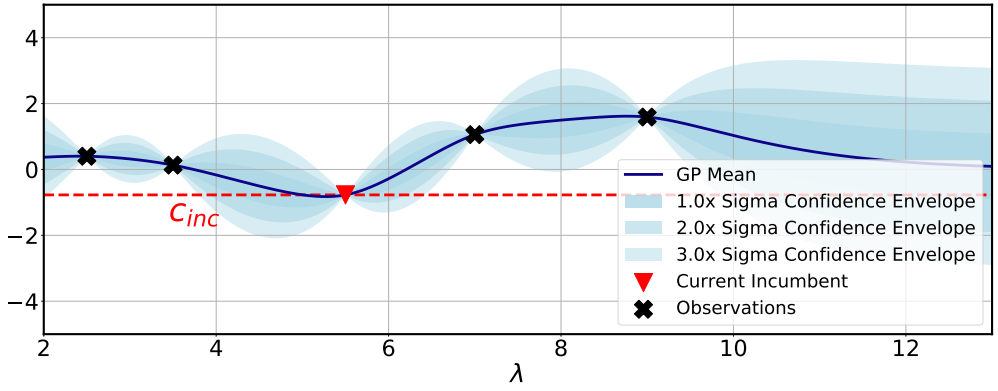

Given the surrogate fit at iteration $t$

# Probability of Improvement (PI): Concept



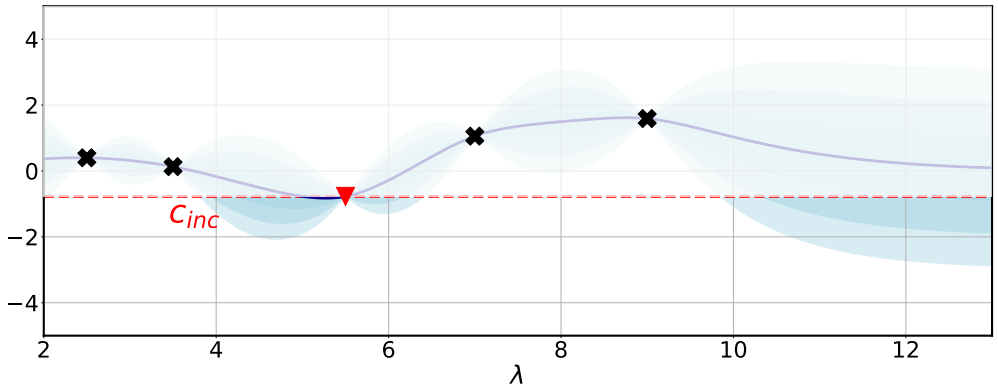

Current incumbent $\hat{\boldsymbol{\lambda}}$ and its observed cost $c_{inc}$

# Probability of Improvement (PI): Concept



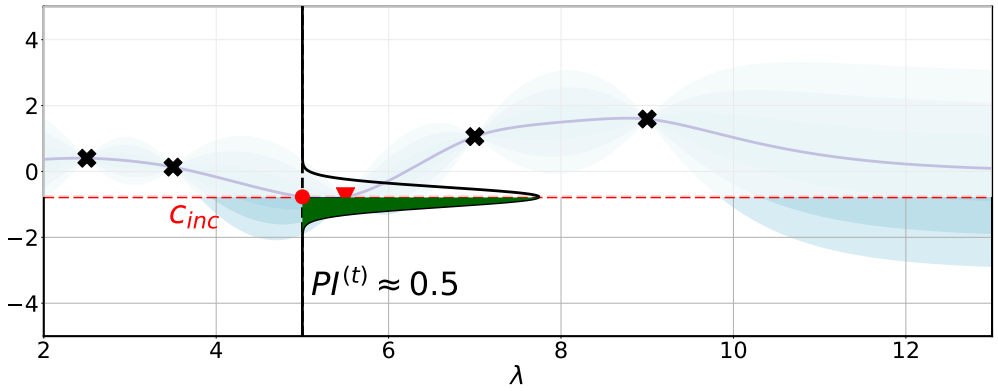

Now let's drop the objective function - it's unknown after all!

# Probability of Improvement (PI): Concept



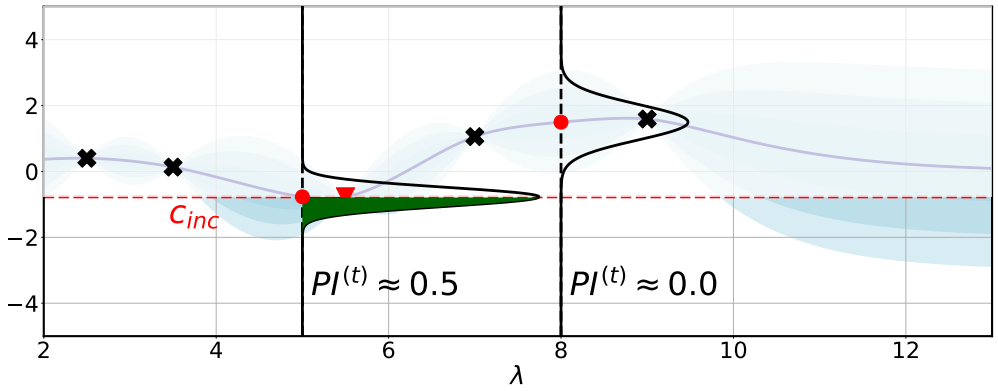

Intuitively, we care about the probability of improving over the current incumbent

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

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

# Probability of Improvement (PI): Formal Definition

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

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

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

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

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

# Probability of Improvement (PI): Formal Definition

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

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

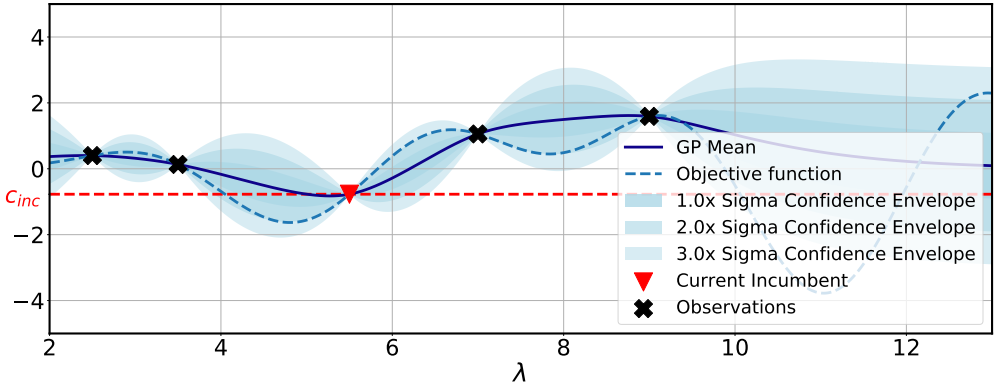
- Since the predictive distribution for $c(\boldsymbol{\lambda})$ is a Gaussian $\mathcal{N}(\mu^{(t-1)}(\boldsymbol{\lambda}), \sigma^{2(t-1)}(\boldsymbol{\lambda}))$, this can be written as:

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

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

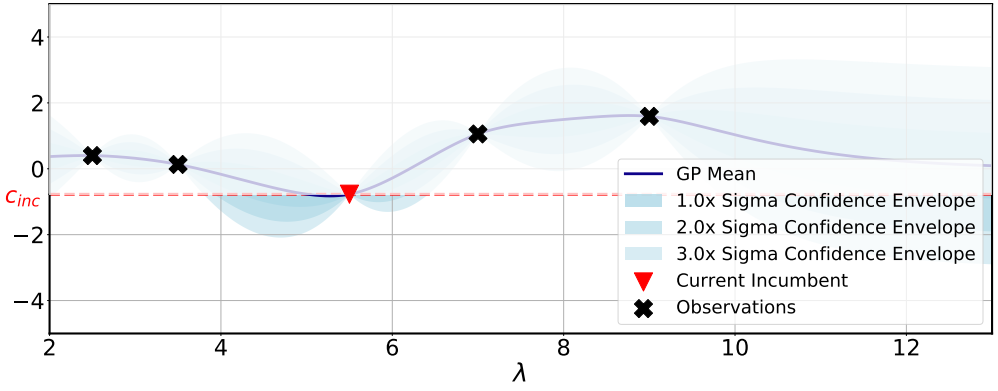
$$\boxed{\text{Choose } \boldsymbol{\lambda}^{(t)} \in \arg\max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} (u_{PI}^{(t)}(\boldsymbol{\lambda}))}$$

# Expected Improvement (EI): Concept



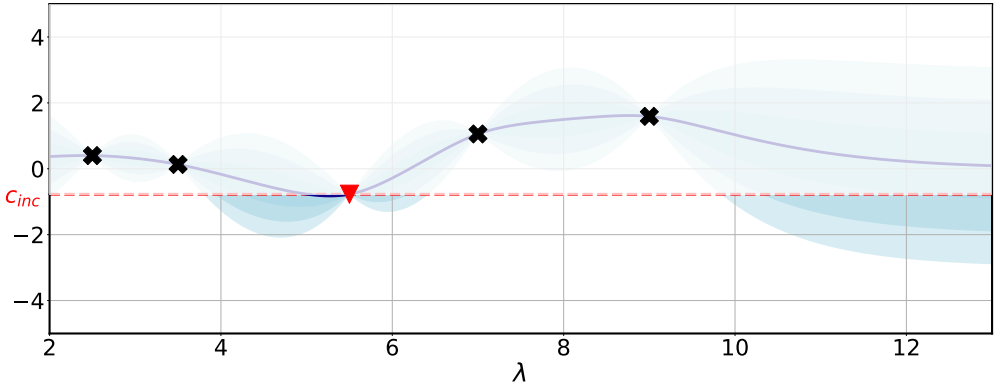

Given the surrogate fit at iteration $t$

# Expected Improvement (EI): Concept



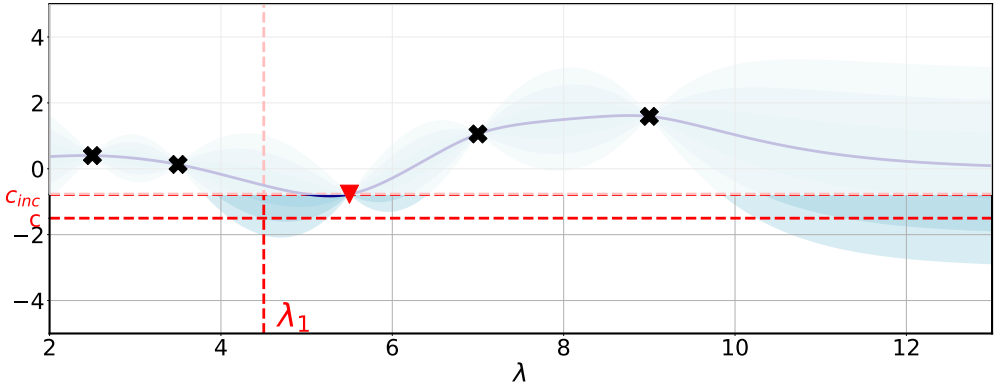

Region of probable improvement – but how large is the improvement?

# Expected Improvement (EI): Concept



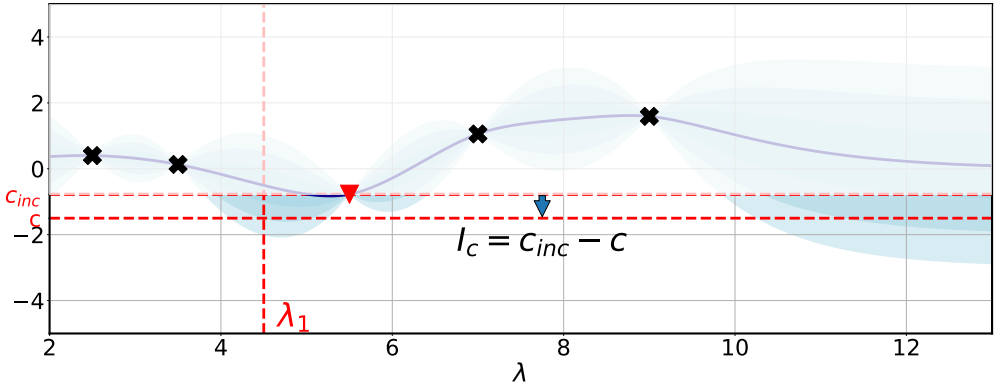

Region of probable improvement – but how large is the improvement?

# Expected Improvement (EI): Concept



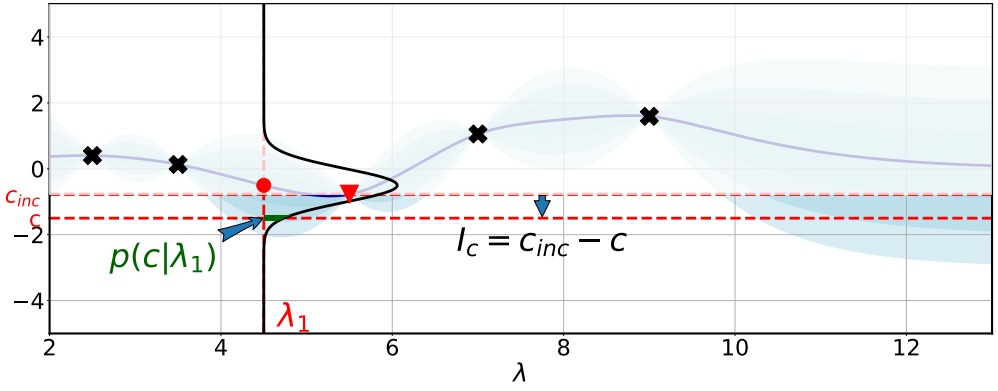

Hypothetical *real* cost $c$ at a given $\boldsymbol{\lambda}$ - unknown in practice without evaluating

# Expected Improvement (EI): Concept



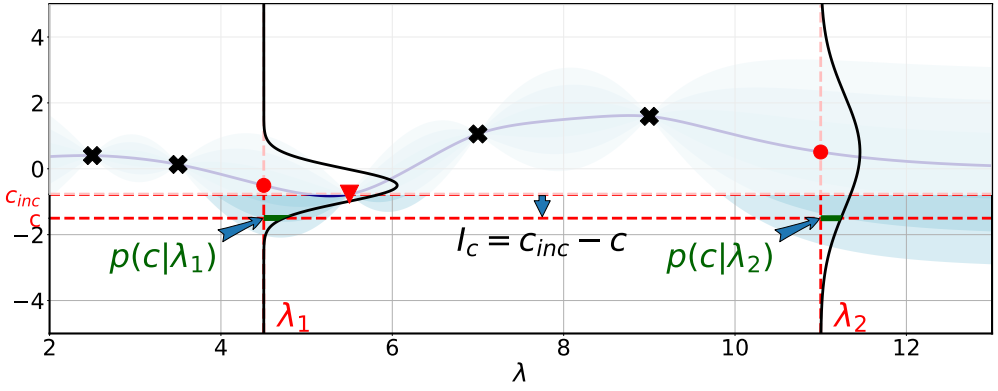

Given a hypothetical $c$, we can compute the improvement $I_c(\boldsymbol{\lambda})$

# Expected Improvement (EI): Concept



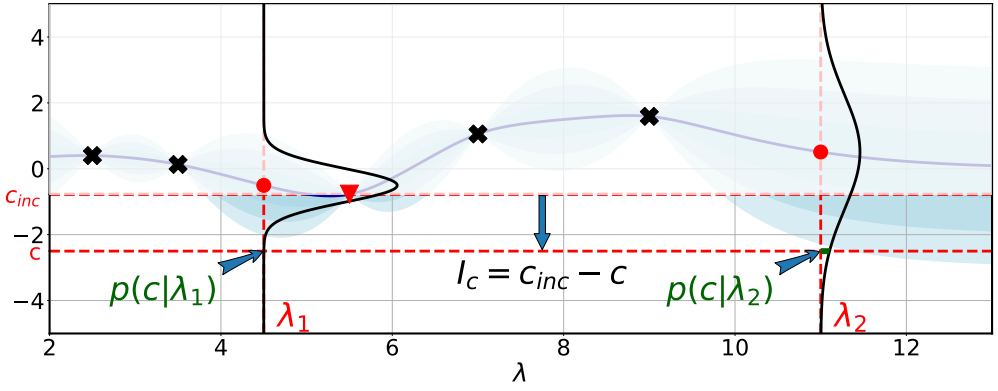

Given $\hat{c}(\boldsymbol{\lambda}) = \mathcal{N}(\mu(\boldsymbol{\lambda}), \sigma^2(\boldsymbol{\lambda}))$, we can also compute $p(c|\boldsymbol{\lambda})$.

# Expected Improvement (EI): Concept



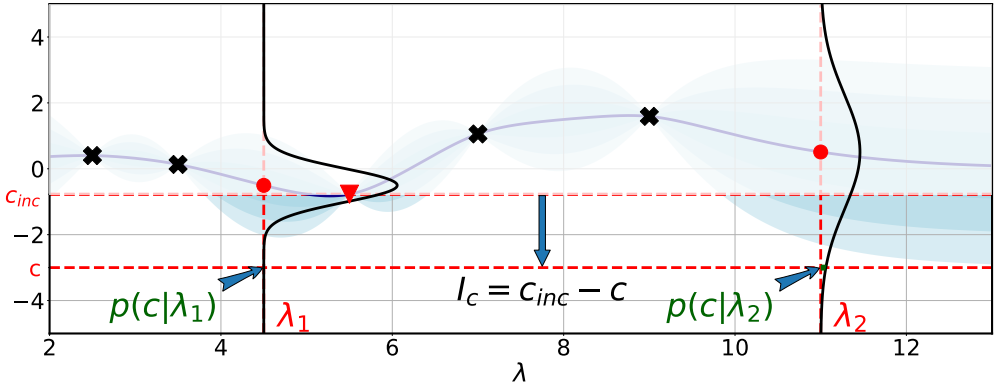

Compare the likelihood of a given improvement for two different configurations $\boldsymbol{\lambda}_1$ and $\boldsymbol{\lambda}_2$

# 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 $\mathbb{E}[I(\boldsymbol{\lambda})]$, intuitively, we sum $p(c \mid \boldsymbol{\lambda}) \times I_c$ over all possibles 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 $Z = \dfrac{c_{inc} - \mu^{(t)}(\boldsymbol{\lambda}) - \xi}{\sigma^{(t)}(\boldsymbol{\lambda})}$ and $\xi$ 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

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

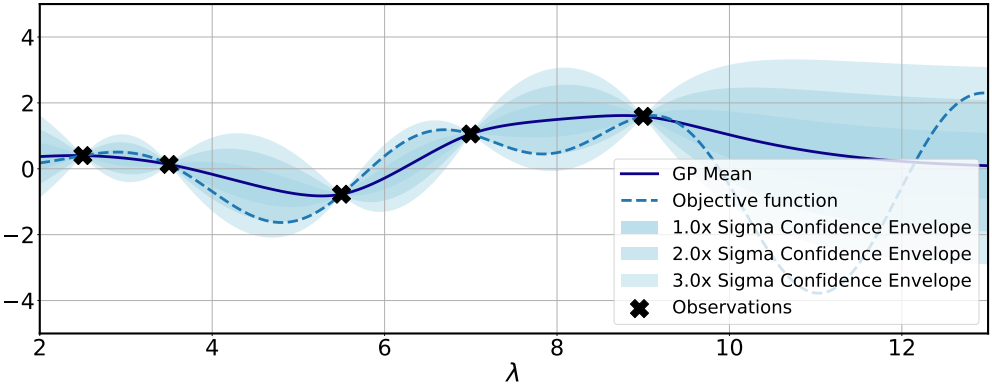
- Since the posterior distribution of $\hat{c}(\boldsymbol{\lambda})$ is a Gaussian, EI can be computed in closed form (see exercise):

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

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

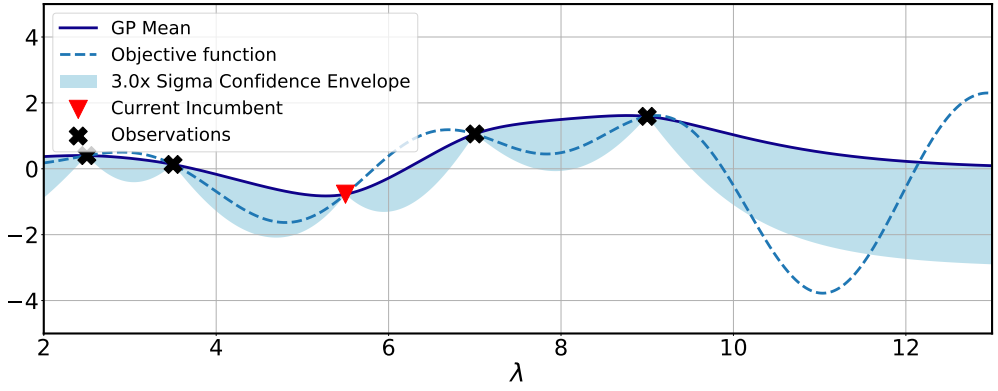
$$\boxed{\text{Choose } \boldsymbol{\lambda}^{(t)} \in \arg\max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} (u_{EI}^{(t)}(\boldsymbol{\lambda}))}$$

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




Given the surrogate fit at iteration $t$

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




Lower Confidence Bound, $\mu(\boldsymbol{\lambda}) - \alpha\sigma(\boldsymbol{\lambda})$ (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 $\alpha$ (e.g., increase it over time [Srinivas et al. 2009])

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

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

- We define the Lower Confidence Bound as

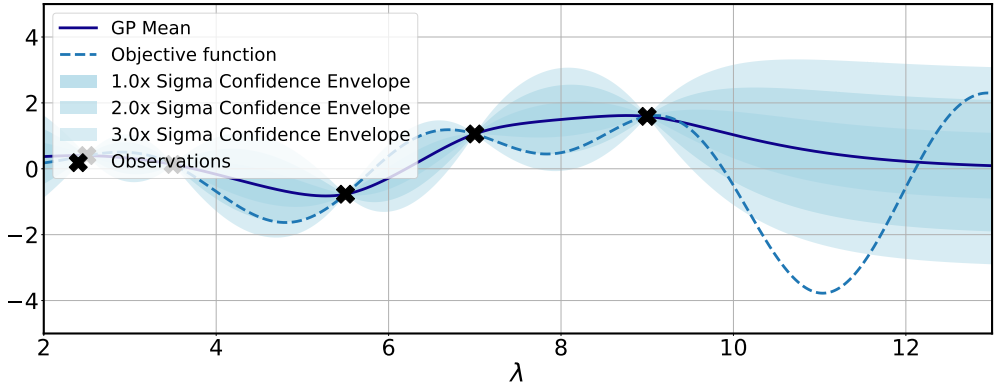
$$u_{LCB}^{(t)}(\boldsymbol{\lambda}) = \mu^{(t)}(\boldsymbol{\lambda}) - \alpha\sigma^{(t)}(\boldsymbol{\lambda}), \quad \alpha \geq 0$$

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

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

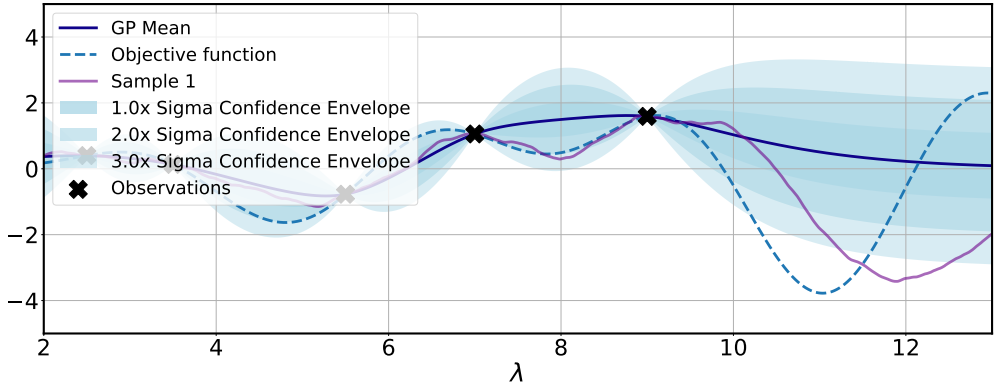
- Note: when one aims to maximize the objective function, one would use UCB instead
  - $u_{UCB}^{(t)}(\boldsymbol{\lambda})) = \mu^{(t)}(\boldsymbol{\lambda}) + \alpha\sigma^{(t)}(\boldsymbol{\lambda})$
  - For UCB, one would choose $\boldsymbol{\lambda}^{(t)} \in \arg\max_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}}(u_{UCB}^{(t)}(\boldsymbol{\lambda}))$

# Thompson Sampling (TS): Concept



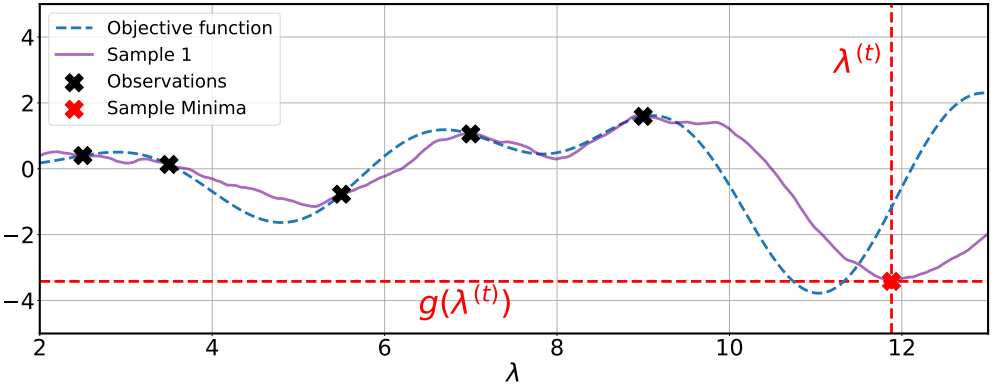

Given the surrogate at iteration $t$ fit on dataset $\mathcal{D}^{(t-1)}$

# 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 $\mathbf{\Lambda}$, cost function $c$, surrogate model $\hat{c}$, maximal number of function evaluations $T$

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

Initialize data $\mathcal{D}^{(0)}$ with initial observations

**for** $t = 1$ **to** $T$ **do**

  Fit predictive model $\hat{c}^{(t)}$ on $\mathcal{D}^{(t-1)}$

  Sample a function from the surrogate: $g \sim \hat{c}^{(t)}$

  Select next query point: $\boldsymbol{\lambda}^{(t)} \in \arg\min_{\boldsymbol{\lambda} \in \mathbf{\Lambda}} g(\boldsymbol{\lambda})$

  Query $c(\boldsymbol{\lambda}^{(t)})$

  Update data: $\mathcal{D}^{(t)} \leftarrow \mathcal{D}^{(t-1)} \cup \{\langle \boldsymbol{\lambda}^{(t)}, c(\boldsymbol{\lambda}^{(t)}) \rangle\}$

# Questions to Answer for Yourself / Discuss with Friends

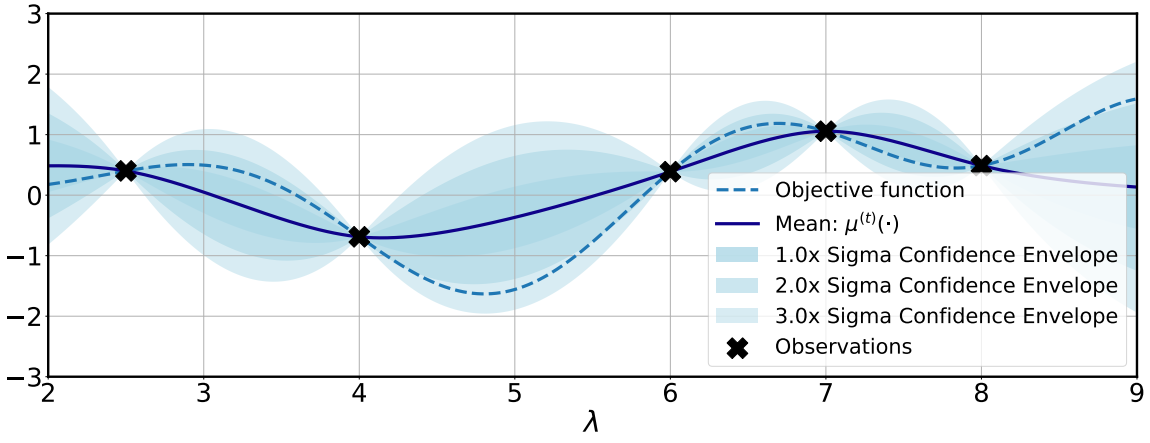
- Discussion. How would you set the exploration parameter $\xi$ for PI if you want to avoid too incremental improvements?
- Derivation. Derive the closed form solution of expected improvement.
- Discussion. In which situations would EI perform substantially differently than PI?

# AutoML: Bayesian Optimization for HPO

## Computationally Expensive Acquisition Functions

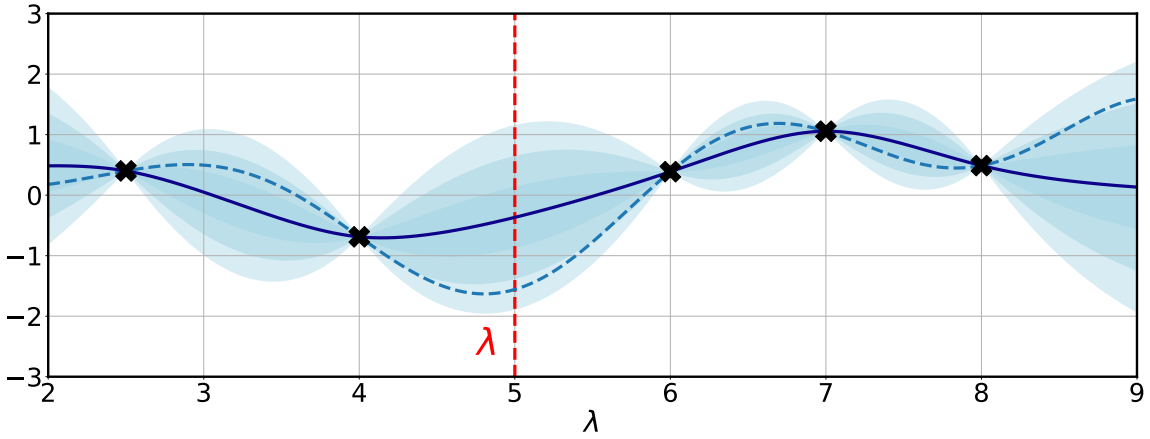
Bernd Bischl  Frank Hutter  Lars Kotthoff
Marius Lindauer  Joaquin Vanschoren

# A Computationally Expensive Step: One-Step Look Ahead



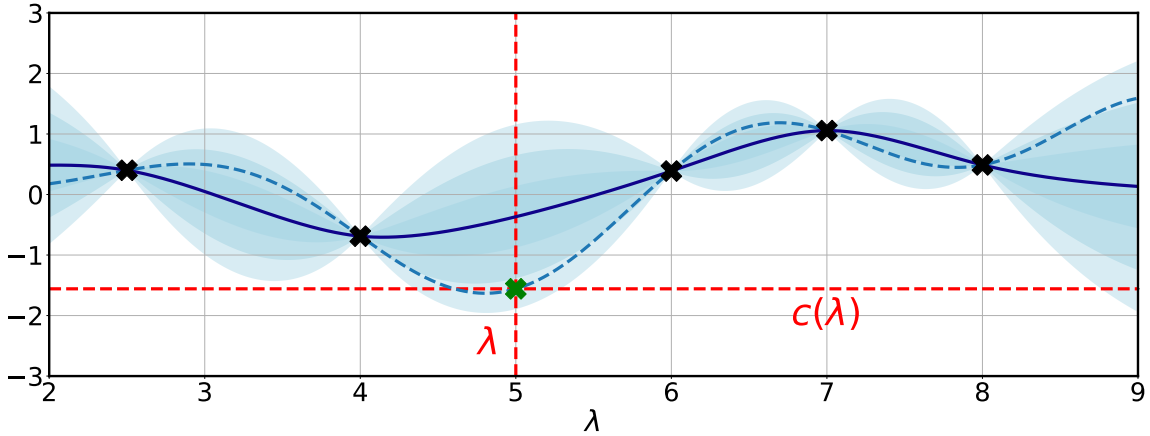

Given the surrogate $\hat{c}^{(t)}$ fit at iteration $t$

# A Computationally Expensive Step: One-Step Look Ahead



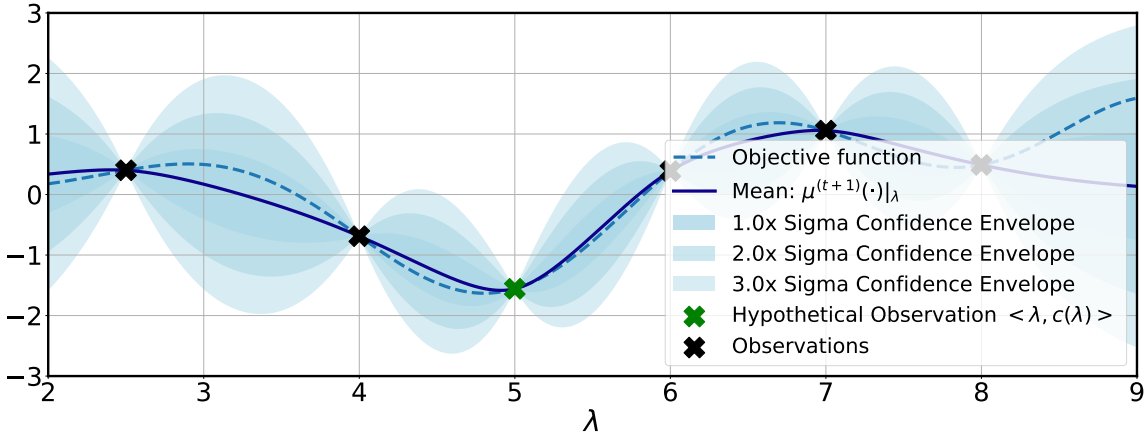

Imagine that we sample at a random configuration $\boldsymbol{\lambda}$

# A Computationally Expensive Step: One-Step Look Ahead



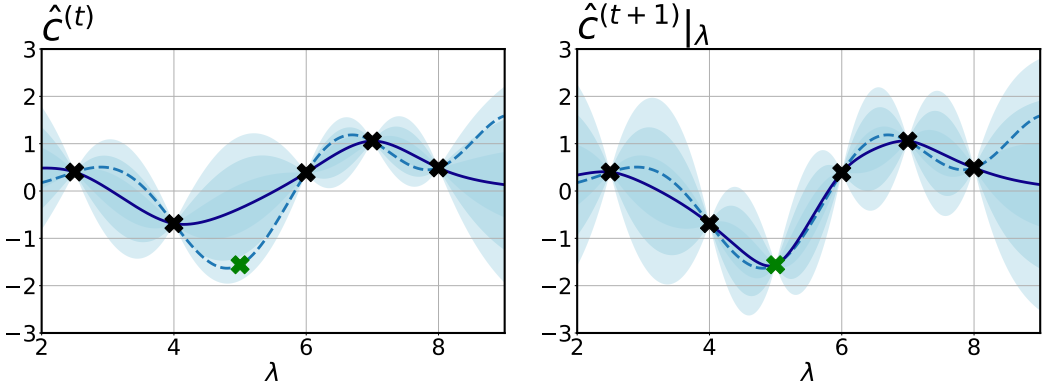

We would then observe the cost $c(\boldsymbol{\lambda})$ at this imaginary configuration $\boldsymbol{\lambda}$

# A Computationally Expensive Step: One-Step Look Ahead



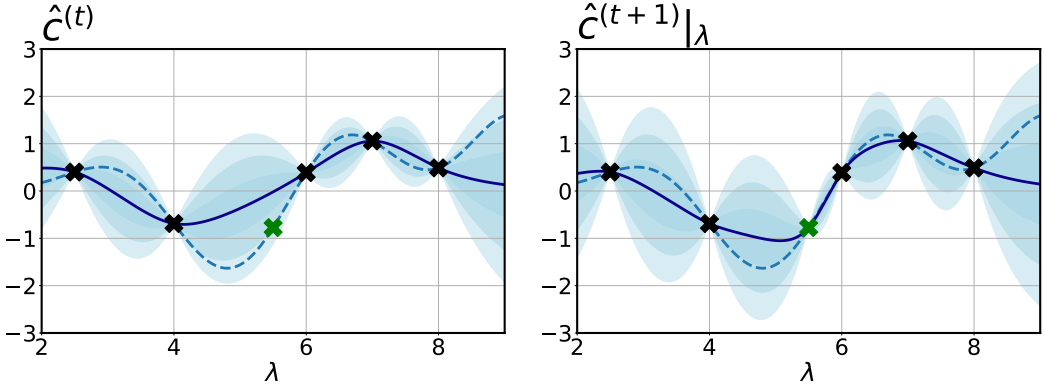

With this hypothetical data point at $\lambda$, we'd have this 1-step lookahead surrogate $\hat{c}^{(t+1)}|_{\boldsymbol{\lambda}}(\cdot)$

# Visualization of How Different the Lookahead Surrogate Can Be



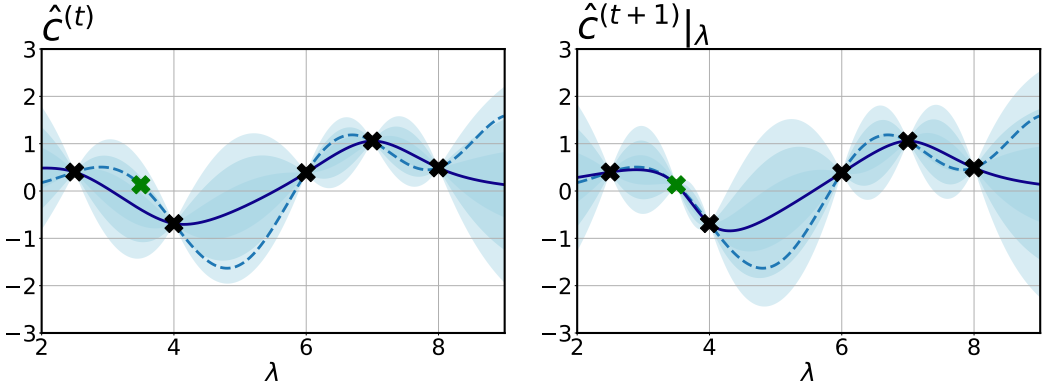

A comparison of $\hat{c}^{(t)}(\cdot)$ and $\hat{c}^{(t+1)}|_{\boldsymbol{\lambda}}(\cdot)$ for a given $\boldsymbol{\lambda}$.

# Visualization of How Different the Lookahead Surrogate Can Be



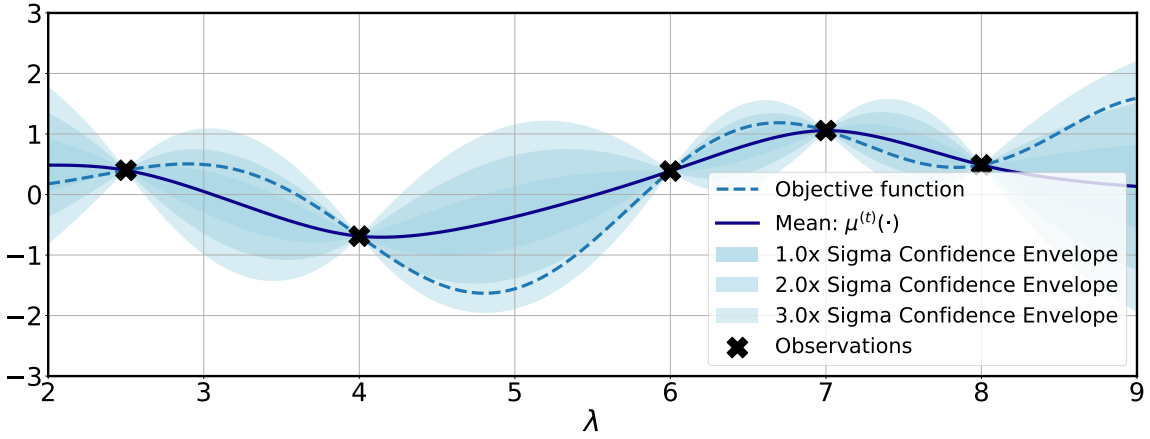

A comparison of $\hat{c}^{(t)}(\cdot)$ and $\hat{c}^{(t+1)}|_{\boldsymbol{\lambda}}(\cdot)$ for a given $\boldsymbol{\lambda}$.

# Visualization of How Different the Lookahead Surrogate Can Be



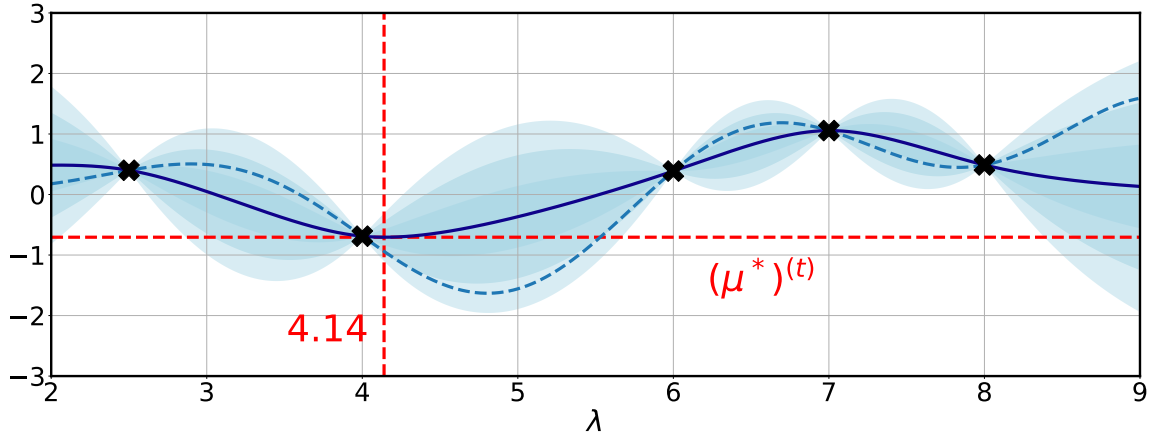

A comparison of $\hat{c}^{(t)}(\cdot)$ and $\hat{c}^{(t+1)}|_{\boldsymbol{\lambda}}(\cdot)$ for a given $\boldsymbol{\lambda}$.

# Knowledge Gradient (KG): Concept



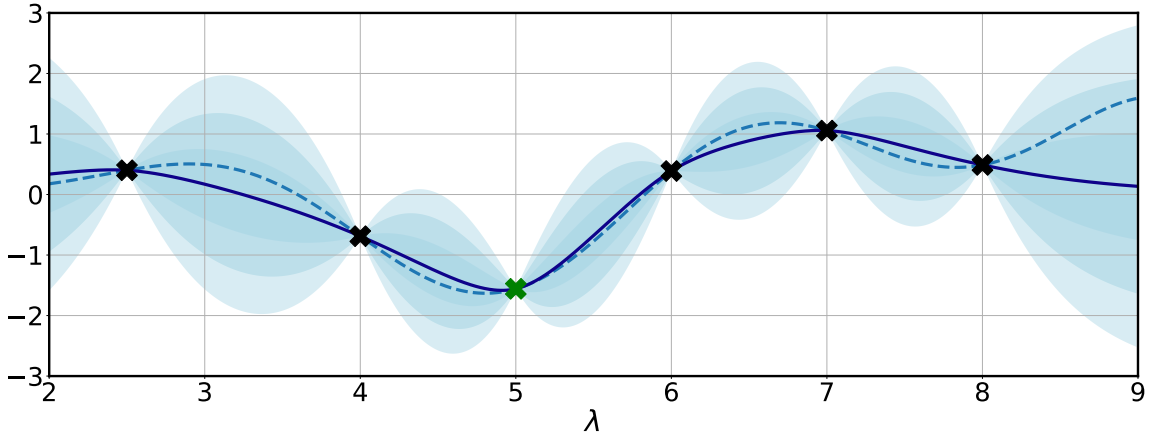

Given the surrogate $\hat{c}(\boldsymbol{\lambda}) = \mathcal{N}(\mu(\boldsymbol{\lambda}), \sigma^2(\boldsymbol{\lambda}))$ fit at iteration $t$

# Knowledge Gradient (KG): Concept



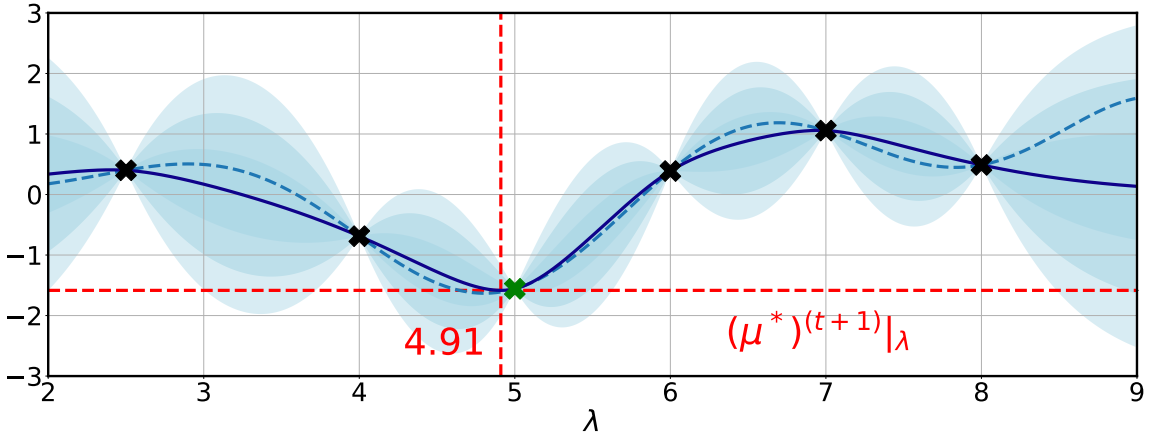

If we are risk-neutral, we'd return $\arg\min_{\boldsymbol{\lambda}} (\mu(\boldsymbol{\lambda}))^{(t)}$ as incumbent, with value $(\mu^*)^{(t)}$

# Knowledge Gradient (KG): Concept



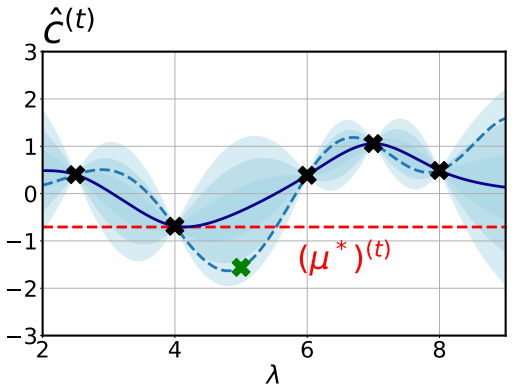

If we perform a one-step look-ahead for configuration $\boldsymbol{\lambda}$, we would get $\hat{c}^{(t+1)}|_{\boldsymbol{\lambda}}$

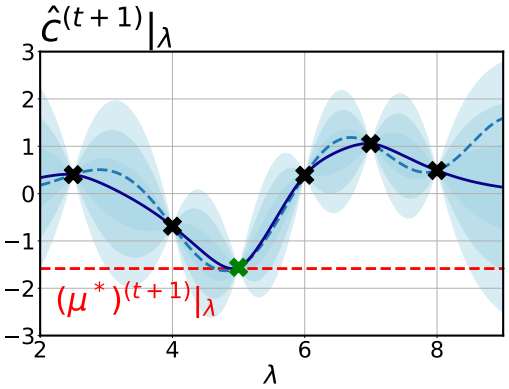
# Knowledge Gradient (KG): Concept




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 $(\mu^*)^{(t+1)} - (\mu^*)^{(t+1)}|_{\boldsymbol{\lambda}}$:

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

# Knowledge Gradient (KG): Formal Definition

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

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

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

# Knowledge Gradient: Pseudocode for Monte Carlo Approximation

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

Sampling Based Knowledge Gradient Acquisition Function

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

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

**for** $s = 1$ **to** $S$ **do**

  Sample $\tilde{c}_s \sim \hat{c}(\boldsymbol{\lambda})$

  Update $\hat{c}$ with $\{\langle \boldsymbol{\lambda}, \tilde{c}_s \rangle\}$ to yield $\hat{c}_s = \mathcal{N}(\mu_s, \sigma_s^2)$

  $e[s] \leftarrow \min_{\boldsymbol{\lambda}'\in\boldsymbol{\Lambda}} \mu_s$

$u \leftarrow const - \frac{1}{S} \sum_{s=1}^{S} e[s]$

# Knowledge Gradient: Pseudocode for Monte Carlo Approximation

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

Sampling Based Knowledge Gradient Acquisition Function

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

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

**for** $s = 1$ **to** $S$ **do**

  Sample $\tilde{c}_s \sim \hat{c}(\boldsymbol{\lambda})$

  Update $\hat{c}$ with $\{\langle \boldsymbol{\lambda}, \tilde{c}_s \rangle\}$ to yield $\hat{c}_s = \mathcal{N}(\mu_s, \sigma_s^2)$

  $e[s] \leftarrow \min_{\boldsymbol{\lambda}'\in\boldsymbol{\Lambda}} \mu_s$

$u \leftarrow const - \frac{1}{S} \sum_{s=1}^{S} e[s]$

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 $\boldsymbol{\lambda}$ which most reduces our uncertainty about the location of $\boldsymbol{\lambda}^*$

# Entropy Search Preliminaries

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

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

# Entropy Search Preliminaries

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

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

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

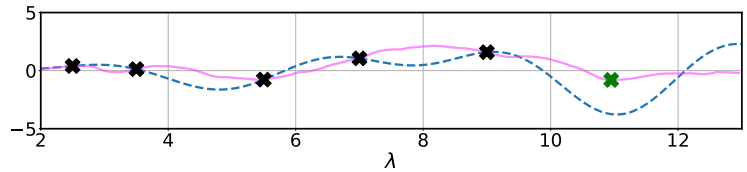
# Entropy Search Preliminaries

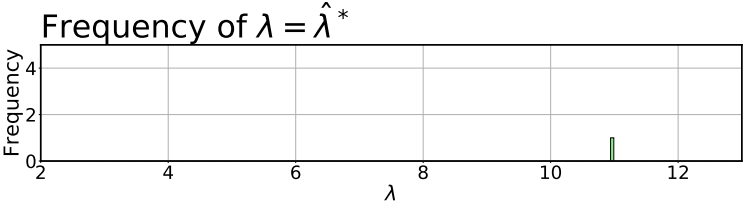
- Key idea: Evaluate $\boldsymbol{\lambda}$ which most reduces our uncertainty about the location of $\boldsymbol{\lambda}^*$
- We'll use the $p_{min}$ distribution to characterize the location of $\boldsymbol{\lambda}^*$:

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

- 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 $\boldsymbol{\lambda}^*$

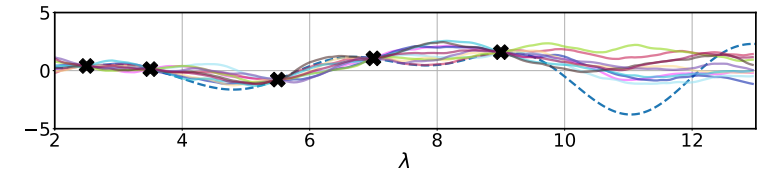
# Entropy Search: Visualization of the $p_{min}$ Distribution

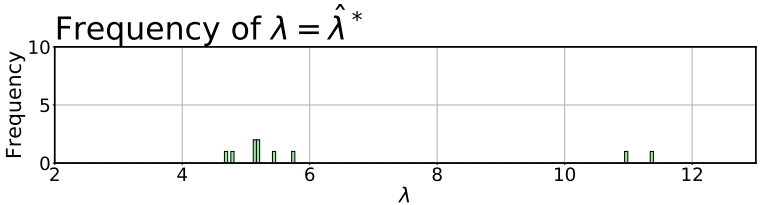



For each sample drawn from $\hat{c}$, we can compute where $\boldsymbol{\lambda}^*$ lies

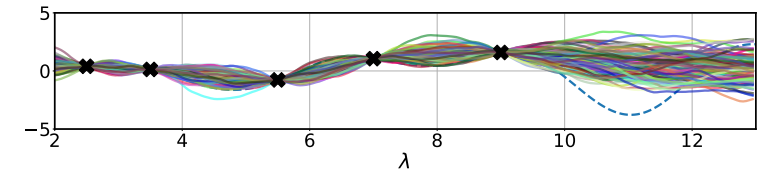
# Entropy Search: Visualization of the $p_{min}$ Distribution

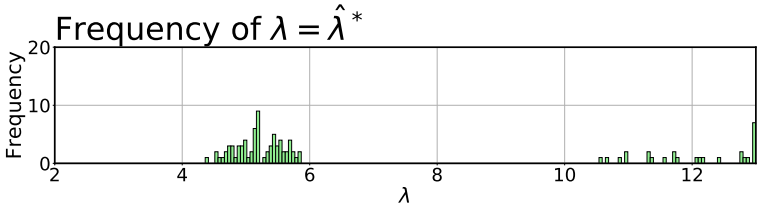



For each sample drawn from $\hat{c}$, we can compute where $\boldsymbol{\lambda}^*$ lies

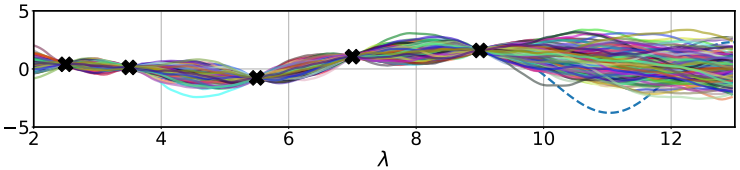
# Entropy Search: Visualization of the $p_{min}$ Distribution

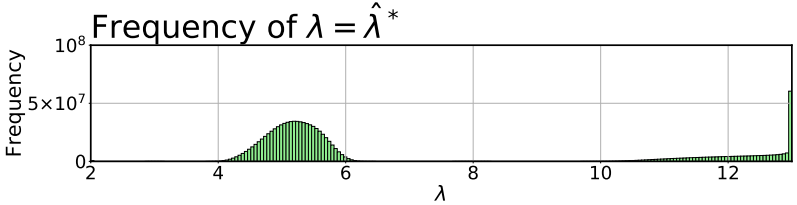



From many samples we can approximate the $p_{min}$ distribution

# 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 $\boldsymbol{\lambda}^*$:

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

# Entropy Search: Formal Definition

- The $p_{min}$ distribution characterizes the location of $\boldsymbol{\lambda}^*$:

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

- Our uncertainty about the location of $\boldsymbol{\lambda}^*$ 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 $\boldsymbol{\lambda}^*$:

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

- Our uncertainty about the location of $\boldsymbol{\lambda}^*$ 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:

$$u_{ES}(\boldsymbol{\lambda}) = H(p_{min}(\cdot|\mathcal{D})) - \mathop{\mathbb{E}}_{\tilde{c} \sim \hat{c}(\lambda)^{(t)}} H(p_{min}(\cdot|\mathcal{D} \cup \{\langle \boldsymbol{\lambda}, \tilde{c} \rangle\}))$$

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

# Entropy Search: Pseudocode for Monte Carlo Approximation

$$u_{ES}(\boldsymbol{\lambda}) = const - \mathop{\mathbb{E}}_{\tilde{c}\sim\hat{c}(\lambda)^{(t)}} H(p_{min}(\cdot|\mathcal{D} \cup \{\langle \boldsymbol{\lambda}, \tilde{c}\rangle\}))$$

**Sampling Based Entropy Search Acquisition Function**

**Require** : Surrogate $\hat{c}$, candidate configuration $\boldsymbol{\lambda}$, finite set of representer points $\boldsymbol{\Lambda}_r$, dataset $\mathcal{D}$
**Result** : Utility $u(\boldsymbol{\lambda})$

**for** $s = 1$ **to** $S$ **do**
  Sample $\tilde{c}_s \sim \hat{c}(\boldsymbol{\lambda})$; $\hat{c}_s \leftarrow$ Update $\hat{c}$ with $\{\langle \boldsymbol{\lambda}, \tilde{c}_s\rangle\}$
  Initialize $F[\boldsymbol{\lambda}] = 0 \quad \forall \boldsymbol{\lambda}' \in \boldsymbol{\Lambda}_r$
  **for** $n = 1$ **to** $N$ **do**
    Sample $g_n \sim \hat{c}_s$
    $\boldsymbol{\lambda}_s \leftarrow \arg\min_{\boldsymbol{\lambda}' \in \boldsymbol{\Lambda}_r} g_n$
    $F[\boldsymbol{\lambda}_s] \leftarrow F[\boldsymbol{\lambda}_s] + 1$
  $p_{min,s}(\boldsymbol{\lambda}') \leftarrow F_{\boldsymbol{\lambda}'}/N \quad \forall \boldsymbol{\lambda}' \in \boldsymbol{\Lambda}_r$
  $H_s \leftarrow H(p_{min,s})$, computed as $-\sum_{\boldsymbol{\lambda}' \in \boldsymbol{\Lambda}_r} p_{min,s}(\boldsymbol{\lambda}') \log p_{min,s}(\boldsymbol{\lambda}')$
$u \leftarrow const - \frac{1}{S}\sum_{s=1}^{S} H_s$

# 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

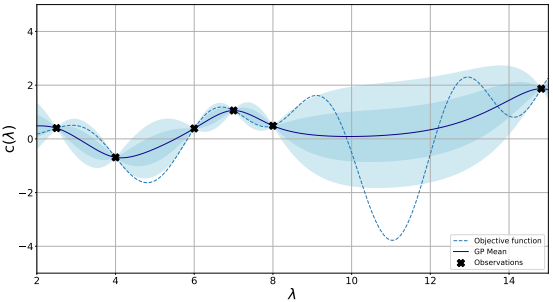
## Surrogate Models

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



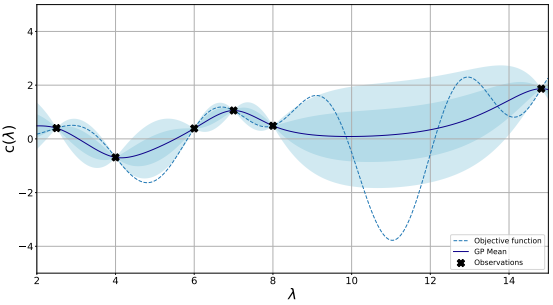
# Desiderata for Surrogate Models in Bayesian Optimization

## 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

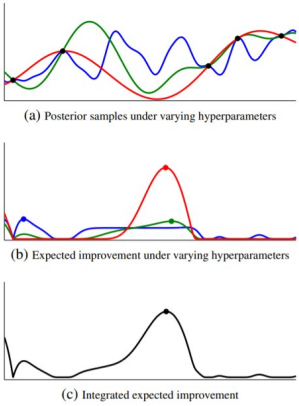
# Gaussian Processes (GPs): Kernel Hyperparameters

- 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 $\theta$ and compute an integrated acquisition function:

$$\bar{u}(\boldsymbol{\lambda}) = \int u(\boldsymbol{\lambda}, \hat{c}_\theta) p(\theta) d\theta$$



(a) Posterior samples under varying hyperparameters

(b) Expected improvement under varying hyperparameters

(c) Integrated expected improvement

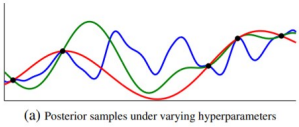
Image source: [Snoek et al. 2015]

# Gaussian Processes (GPs): Kernel Hyperparameters

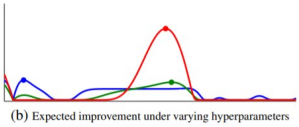
- We could optimize GP hyperparameters (maximum likelihood, MLE, or maximum a posteriori, MAP)
- But sampling GP hyperparameters from the posterior distribution performs better; e.g., via Markov-Chain Monte-Carlo (MCMC)
- Marginalize over GP hyperparameters $\theta$ and compute an integrated acquisition function:

$$\bar{u}(\boldsymbol{\lambda}) = \int u(\boldsymbol{\lambda}, \hat{c}_\theta) p(\theta) d\theta$$

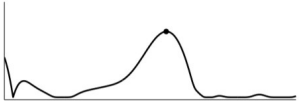
- Downside: computational expense
  - MCMC is computationally expensive
  - Acquisition function now has to be calculated for each sample



(a) Posterior samples under varying hyperparameters



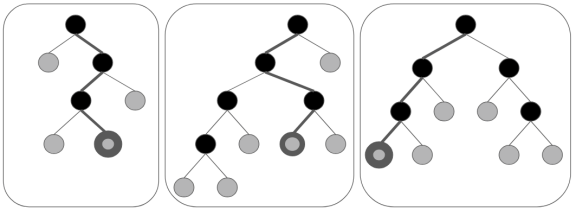
(b) Expected improvement under varying hyperparameters



(c) Integrated expected improvement

Image source: [Snoek et al. 2015]

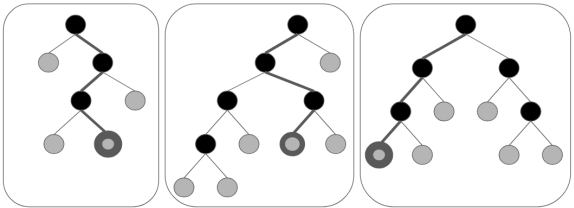
# Random Forests (RFs): Reminder & How To Compute Uncertainties



## 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

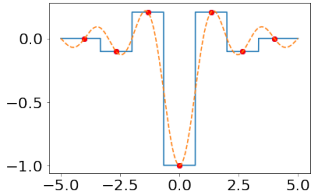
- Fit a set of randomized regression trees
- Randomization via bootstrapping & random selection of split variables / split points
- Each tree yields a possible explanation for the observations

## RF Prediction

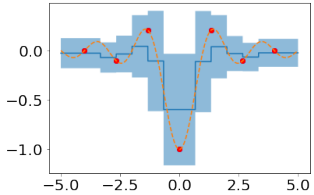
- Predict with each tree
- Aggregate predictions (e.g., average)
- Uncertainty estimate: empirical variance across tree predictions

# Random Forests (RFs): Impact of Basic Model Choices

(a) no bootstrapping,
no random splits

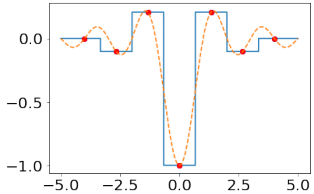



(b) with bootstrapping,
no random splits

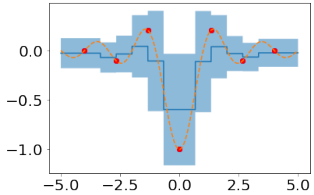


# Random Forests (RFs): Impact of Basic Model Choices

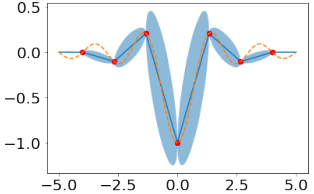
(a) no bootstrapping,
no random splits



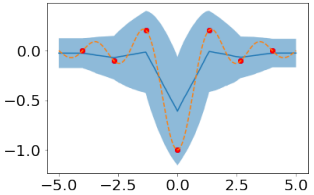

(b) with bootstrapping,
no random splits




(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

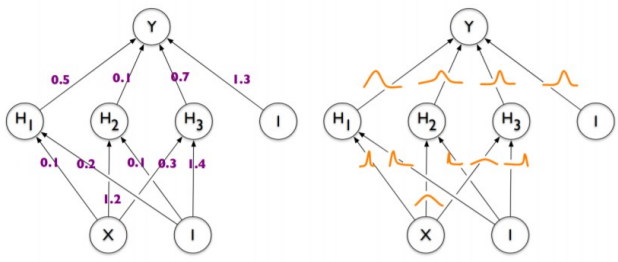



Image source: [Blundell et al. 2015]

# 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 $\theta$
  - 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

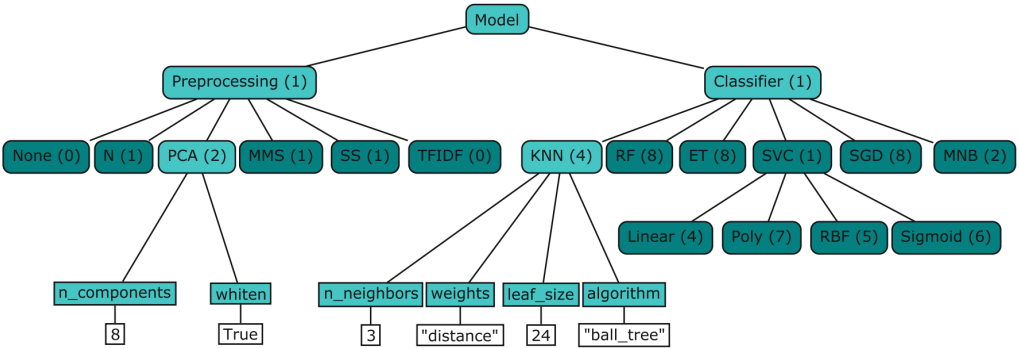
## Standard Bayesian optimization problems

- 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

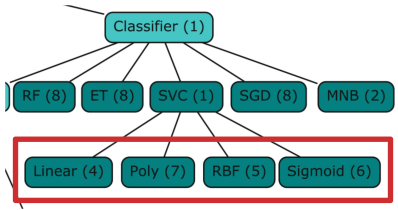



Example of a structured search space [Hutter et al. 2019]

# Structured Search Spaces: Categorical Hyperparameters

Properties of categorical hyperparameters:

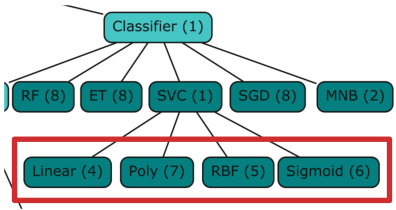
- Finite, discrete set of values
- No natural order between values
- Potentially different distances between values



# 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]:

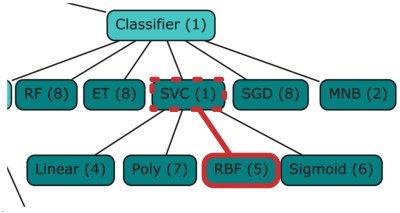
$$\kappa_\theta(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_j) = \exp \sum_{l=1}^{d} (-\theta \cdot \delta(\lambda_{i,l} \neq \lambda_{j,l}))$$

- Neural networks can learn entity embeddings for categorical inputs [Guc and Berkhahn. 2016]

# Structured Search Spaces: Conditional Hyperparameters

Conditional hyperparameters:

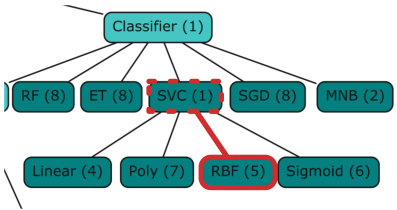
- Are only relevant if certain other hyperparameters take on certain values
- Should be ignored by the model if not active



# 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



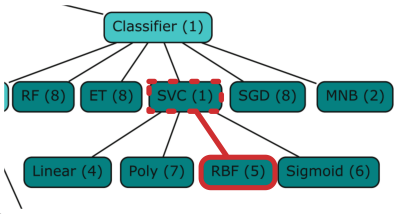

Modelling conditional hyperparameters:

- 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]

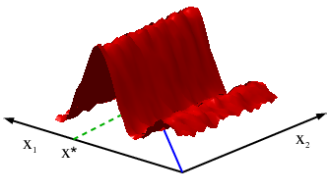
Overall, structured search spaces are still an active research topic and far from solved

# High Dimensions

- Issues
  - Standard Gaussian processes
    do not tend to fit well in high dimensions
  - Maximizing the acquisition function
    is computationally challenging

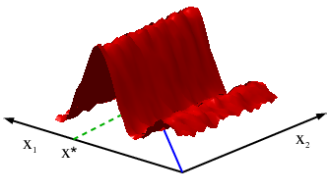
# High Dimensions

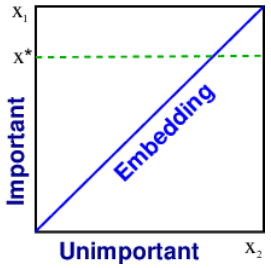
- Issues
  - Standard Gaussian processes do not tend to fit well in high dimensions
  - Maximizing the acquisition function is computationally challenging
- There is still hope
  - Many optimization problems have low effective dimensionality
  - Not all dimensions interact with each other



# 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

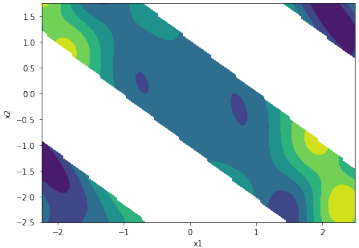



Image source: [GPFlowOpt Tutorial, Apache 2 License]

# 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

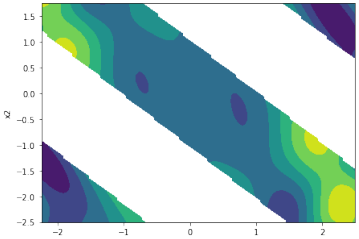



Image source: [GPFlowOpt Tutorial, Apache 2 License]

Most general solution: Expected Constrained Improvement [Lee et al. 2010]:

$$ECI(\boldsymbol{\lambda}) = EI(\boldsymbol{\lambda})h(\boldsymbol{\lambda}),$$

where $h(\boldsymbol{\lambda})$ is the probability that $\boldsymbol{\lambda}$ 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 $\rightarrow$ separate lecture
- Multi-objective Bayesian optimization $\rightarrow$ 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 $p(y \mid \boldsymbol{\lambda})$ of observations $y$ given configurations $\boldsymbol{\lambda}$
- Instead, TPE fits kernel density estimators (KDEs) $l(\boldsymbol{\lambda} \mid y \leq \gamma)$ and $g(\boldsymbol{\lambda} \mid y > \gamma)$
  - These KDEs are for "good configurations" (leading to objective function values below a threshold $\gamma$) and "bad configurations"
  - By default, $\gamma$ is set to the 15% quantile of the observations

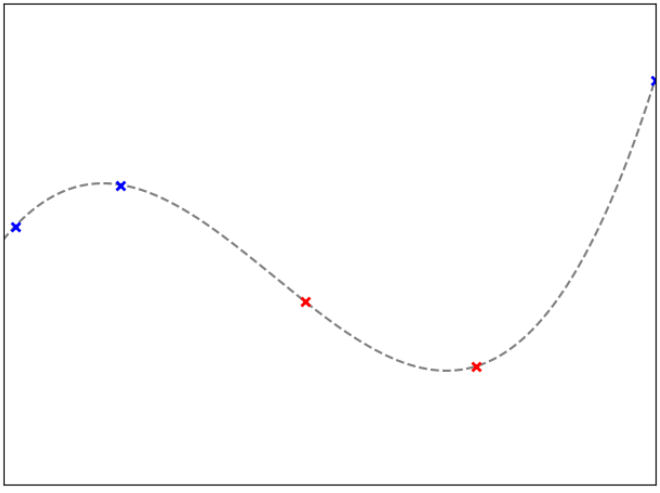
# Overview of TPE [Bergstra et al. 2011]

- Standard Bayesian optimization models the probability $p(y \mid \boldsymbol{\lambda})$ of observations $y$ given configurations $\boldsymbol{\lambda}$
- Instead, TPE fits kernel density estimators (KDEs) $l(\boldsymbol{\lambda} \mid y \leq \gamma)$ and $g(\boldsymbol{\lambda} \mid y > \gamma)$
  - These KDEs are for "good configurations" (leading to objective function values below a threshold $\gamma$) and "bad configurations"
  - By default, $\gamma$ is set to the 15% quantile of the observations
- Optimizing $l(\boldsymbol{\lambda})/g(\boldsymbol{\lambda})$ is equivalent to optimizing standard expected improvement in Bayesian optimization [Bergstra et al. 2011]

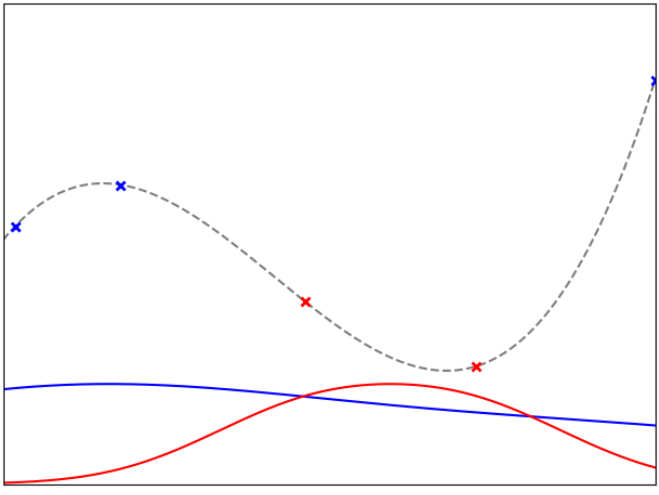
# Overview of TPE [Bergstra et al. 2011]

- Standard Bayesian optimization models the probability $p(y \mid \boldsymbol{\lambda})$ of observations $y$ given configurations $\boldsymbol{\lambda}$
- Instead, TPE fits kernel density estimators (KDEs) $l(\boldsymbol{\lambda} \mid y \leq \gamma)$ and $g(\boldsymbol{\lambda} \mid y > \gamma)$
  - These KDEs are for "good configurations" (leading to objective function values below a threshold $\gamma$) and "bad configurations"
  - By default, $\gamma$ is set to the 15% quantile of the observations
- Optimizing $l(\boldsymbol{\lambda})/g(\boldsymbol{\lambda})$ 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

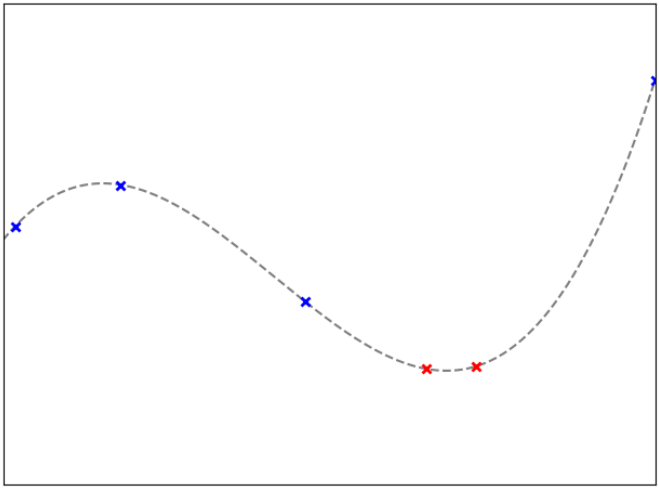
# TPE Example



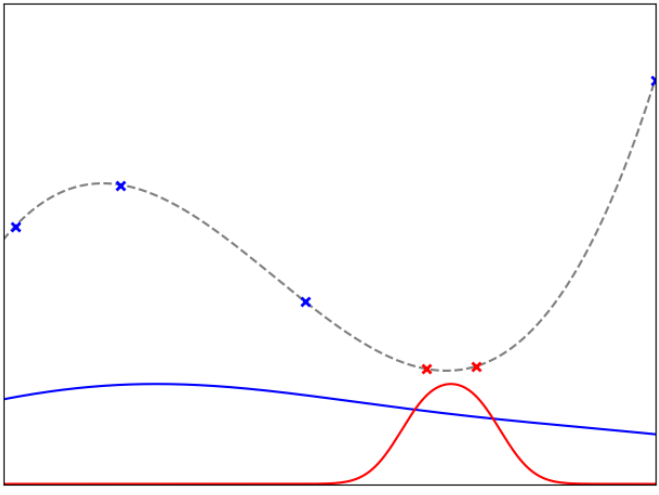
# TPE Example



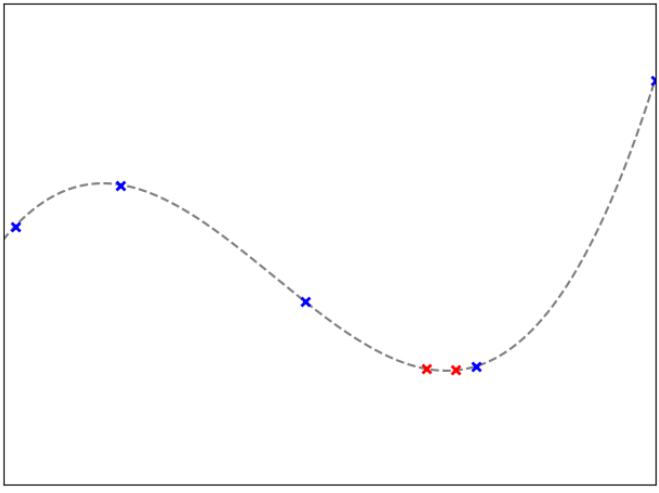
# TPE Example



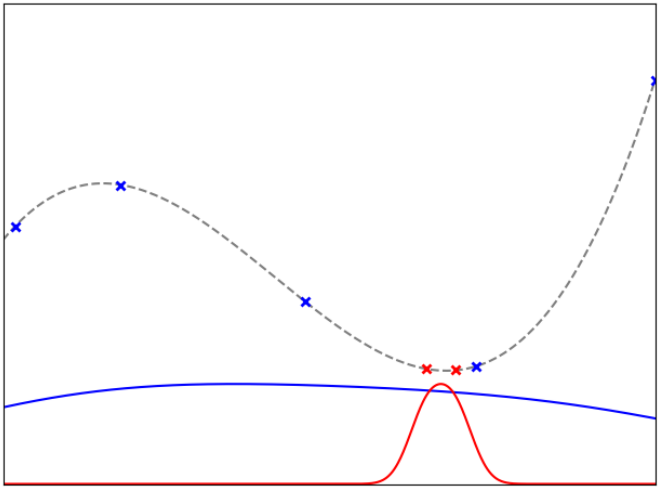
# TPE Example



# TPE Example



# TPE Example



# TPE Pseudocode

**TPE loop**

**Require:** Search space $\mathbf{\Lambda}$, cost function $c$, percentile $\gamma$, maximal number of function evaluations $T$

**Result :** Best observed configuration $\boldsymbol{\lambda}$ according to $\mathcal{D}^{(T)}$

Initialize data $\mathcal{D}^{(0)}$ with initial observations

**for** $t = 1$ **to** $T$ **do**

  $\mathcal{D}_{\text{good}}, \mathcal{D}_{\text{bad}} \leftarrow$ split $\mathcal{D}^{(t-1)}$ according to quantile $\gamma$

  $l(\boldsymbol{\lambda})$, $g(\boldsymbol{\lambda}) \leftarrow$ fit KDE on $\mathcal{D}_{\text{good}}$, $\mathcal{D}_{\text{bad}}$ respectively

  $\mathbf{\Lambda}_{\text{cand}} \leftarrow$ draw samples from $l$;

  Select next query point: $\boldsymbol{\lambda}^{(t)} \in \arg\max_{\boldsymbol{\lambda} \in \mathbf{\Lambda}_{\text{cand}}} l(\boldsymbol{\lambda}) / g(\boldsymbol{\lambda})$

  Query $c(\boldsymbol{\lambda}^{(t)})$

  $\mathcal{D}^{(t)} \leftarrow \mathcal{D}^{(t-1)} \cup \{\langle \boldsymbol{\lambda}^{(t)}, c(\boldsymbol{\lambda}^{(t)}) \rangle\}$

**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 $p(\boldsymbol{\lambda}|c(\boldsymbol{\lambda}))$
  - we can multiply it with a prior to add expert knowledge
- Performance of TPE depends on:
  - setting of $\gamma$ to trade-off exploration and exploitation
  - bandwidth of the KDEs

# Further Details

Remarks:

- TPE models $p(\boldsymbol{\lambda}|c(\boldsymbol{\lambda}))$
  - we can multiply it with a prior to add expert knowledge
- Performance of TPE depends on:
  - setting of $\gamma$ 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

- Disussion. Is TPE really Bayesian optimization?
- Disussion. How does $\gamma$ impact the optimization procedure?
- Derivation. Go through the derivation that optimizing $l(\boldsymbol{\lambda})/g(\boldsymbol{\lambda})$ is equivalent to optimizing expected improvement; see Section 4.1 in [Bergstra et al. 2011].

# AutoML: Bayesian Optimization for HPO

## Success Stories

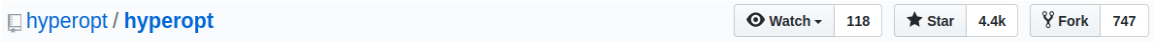
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

# 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`:

# 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`



# 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)

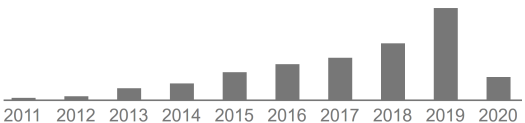
# SMAC [Hutter et al. 2011]

- 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: $\arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \sum_{i=1}^{N} c(\boldsymbol{\lambda}, i)$

# 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: $\arg\min_{\boldsymbol{\lambda} \in \boldsymbol{\Lambda}} \sum_{i=1}^{N} c(\boldsymbol{\lambda}, i)$
- 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

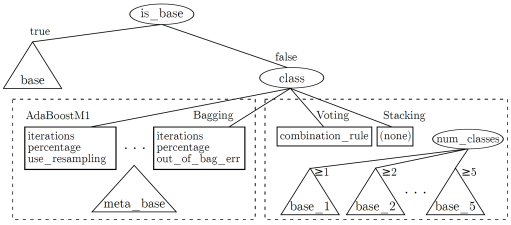
- SIGOPT: startup offering Bayesian optimization as a service
- Facebook provides an open source Bayesian optimization package [BoTorch]
- Amazon provides an open source Bayesian optimization package [EmuKit]
- Uber tunes algorithms for *Uber Pool*, *UberX* and *Uber Eats* [source]
- Many more, but less openly

# Auto-WEKA [Thornton et al, 2013, Kotthoff et al, 2017, Kotthoff et al. 2019]

- First general AutoML system, carrying out **C**ombined **A**lgorithm **S**election and **H**yperparameter 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

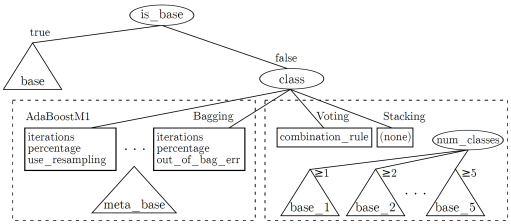
# Auto-WEKA [Thornton et al, 2013, Kotthoff et al, 2017, Kotthoff et al. 2019]

- First general AutoML system, carrying out **C**ombined **A**lgorithm **S**election and **H**yperparameter 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



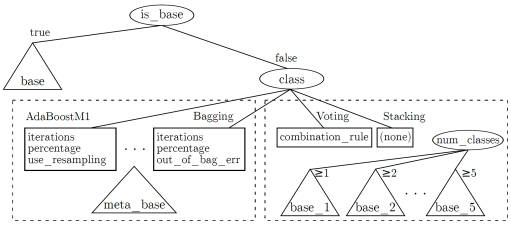
# Auto-WEKA [Thornton et al, 2013, Kotthoff et al, 2017, Kotthoff et al. 2019]

- First general AutoML system, carrying out **C**ombined **A**lgorithm **S**election and **H**yperparameter 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
- 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 **C**ombined **A**lgorithm **S**election and **H**yperparameter 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
- 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 at 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]