

Outline

- 1 Introduction to Bayesian Optimization
- 2 Computationally Cheap Acquisition Functions
- 3 Computationally Expensive Acquisition Functions
- 4 Surrogate Models
- 5 High-Dimensional Bayesian Optimization
- 6 Extensions to Bayesian Optimization
- 7 The Alternative Optimization Approach of the Tree-Parzen Estimator (TPE)
- 8 Success Stories
- 9 Further Reading

Outline

- 1 Introduction to Bayesian Optimization
- 2 Computationally Cheap Acquisition Functions
- 3 Computationally Expensive Acquisition Functions
- 4 Surrogate Models
- 5 High-Dimensional Bayesian Optimization
- 6 Extensions to Bayesian Optimization
- 7 The Alternative Optimization Approach of the Tree-Parzen Estimator (TPE)
- 8 Success Stories
- 9 Further Reading

AutoML: Bayesian Optimization for Hyperparameter Optimization

Introduction to Bayesian Optimization

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Blackbox Optimization for Hyperparameter Optimization

- Consider the global optimization problem of finding:

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

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

$$\boldsymbol{\lambda} \rightarrow \boxed{\quad} \rightarrow f(\boldsymbol{\lambda})$$

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

Blackbox Optimization for Hyperparameter Optimization

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

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

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

$$\boldsymbol{\lambda} \rightarrow \boxed{\quad} \rightarrow f(\boldsymbol{\lambda})$$

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

Blackbox Optimization for Hyperparameter Optimization

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

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

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

$$\boldsymbol{\lambda} \rightarrow \boxed{\quad} \rightarrow f(\boldsymbol{\lambda})$$

- Only mode of interaction with f : querying f 's value at a given $\boldsymbol{\lambda}$
- Function f may not be available in closed form, not 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 $f(\boldsymbol{\lambda}) := \mathcal{L}(\mathcal{A}_{\boldsymbol{\lambda}}, \mathcal{D}_{train}, \mathcal{D}_{valid})$

Blackbox Optimization for Hyperparameter Optimization

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

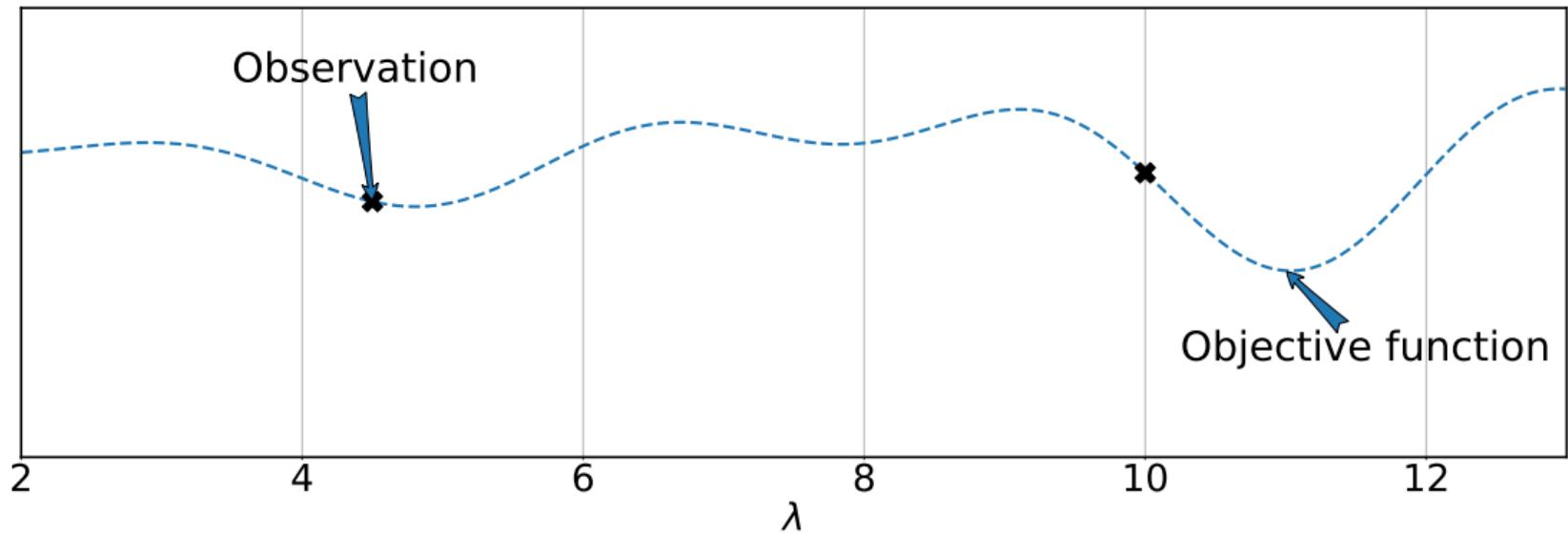
$$\boldsymbol{\lambda}^* \in \arg \min_{\boldsymbol{\lambda} \in \Lambda} f(\boldsymbol{\lambda})$$

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

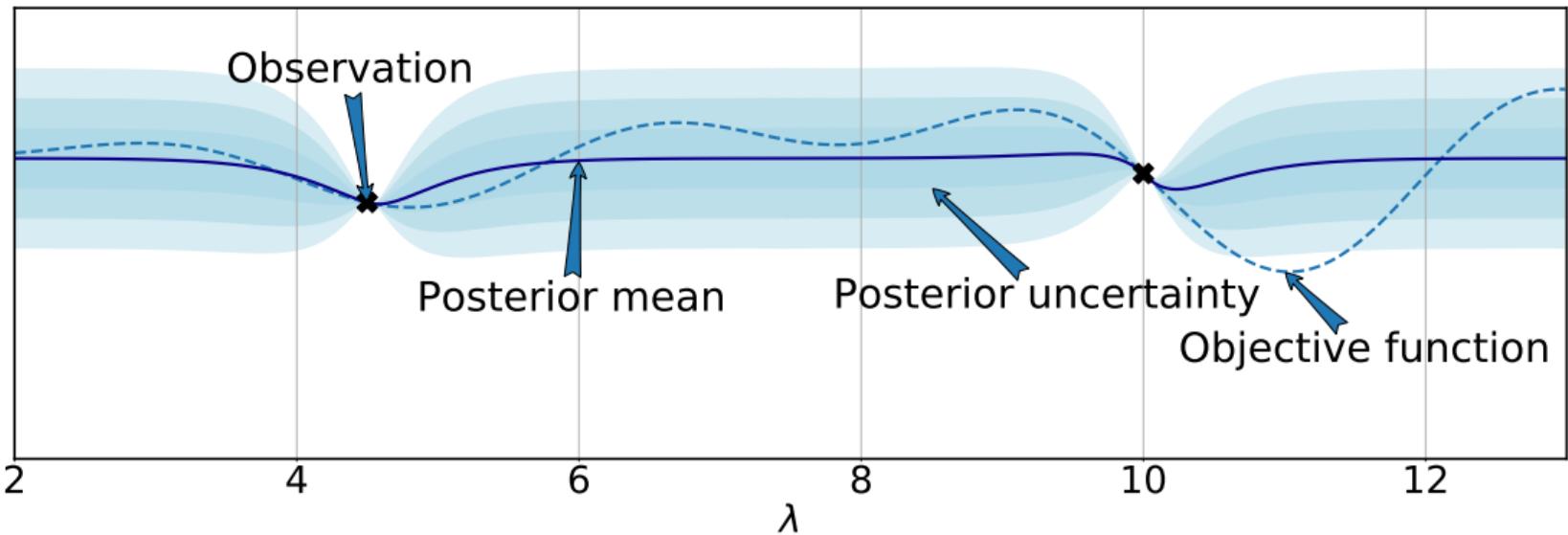
$$\boldsymbol{\lambda} \rightarrow \boxed{\quad} \rightarrow f(\boldsymbol{\lambda})$$

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

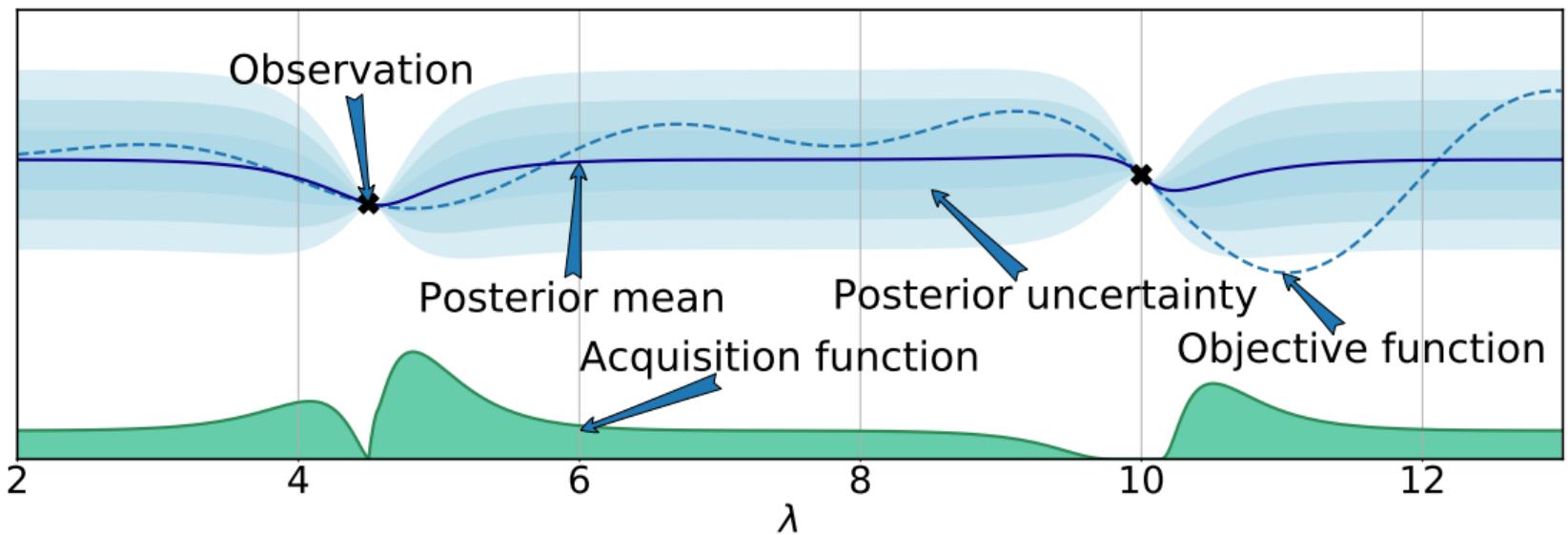
Bayesian Optimization of a blackbox function in a nutshell



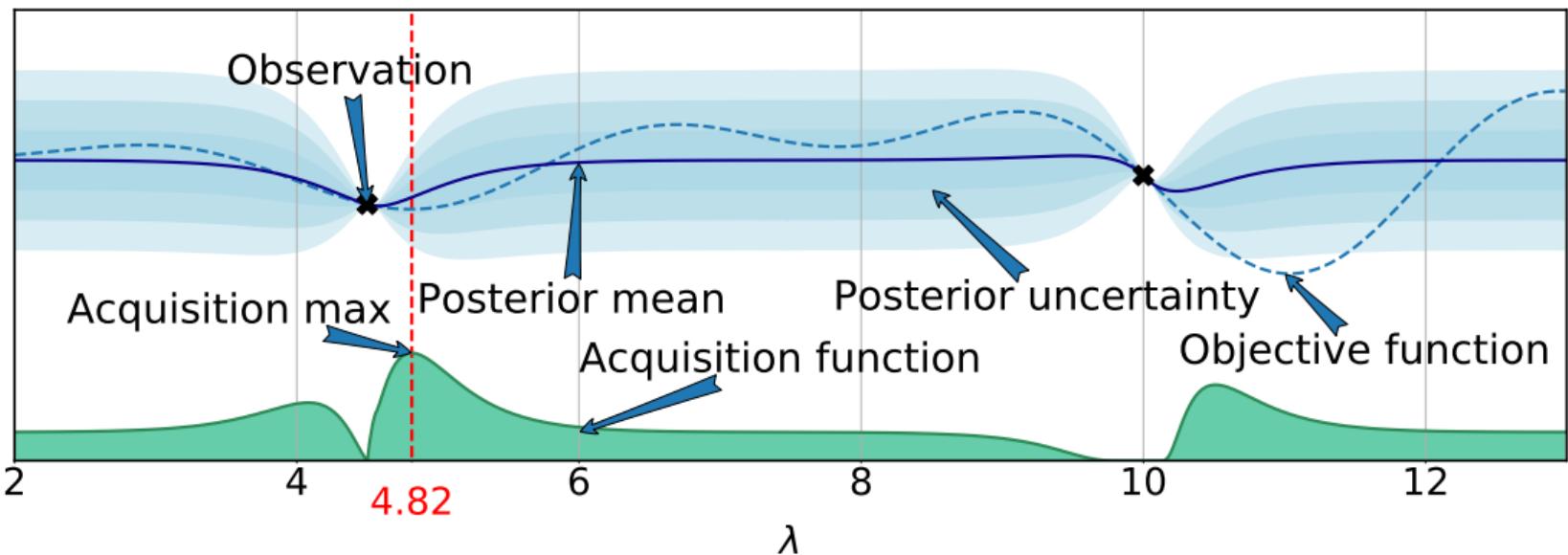
Bayesian Optimization of a blackbox function in a nutshell



Bayesian Optimization of a blackbox function in a nutshell



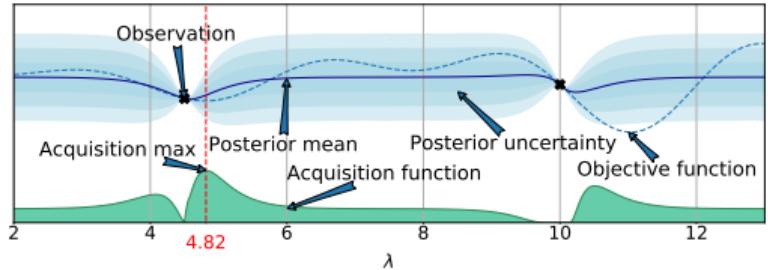
Bayesian Optimization of a blackbox function in a nutshell



Bayesian Optimization of a blackbox function in a nutshell

General approach

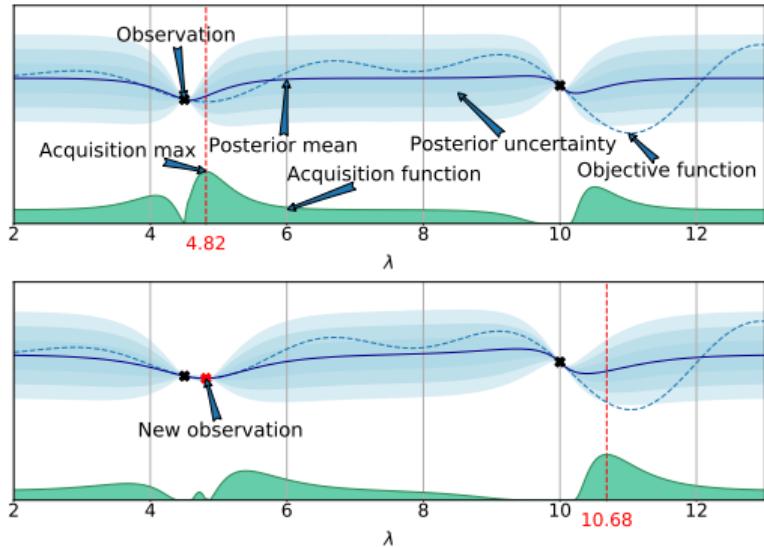
- Fit a **probabilistic model** to the collected function samples $\langle \lambda, c(\lambda) \rangle$
- Use the model to guide optimization, trading off **exploration vs exploitation**



Bayesian Optimization of a blackbox function in a nutshell

General approach

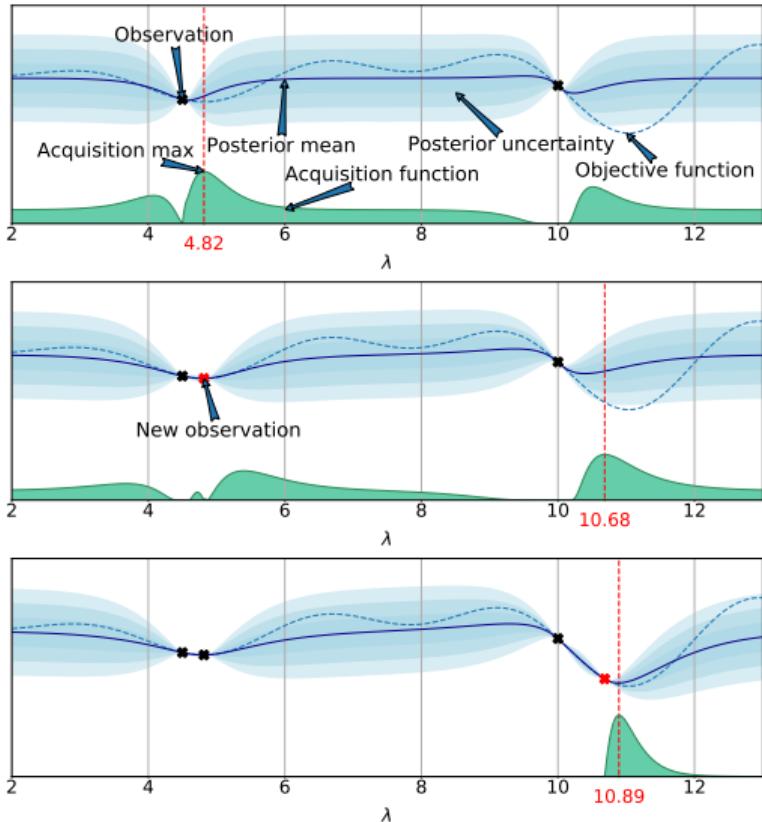
- Fit a **probabilistic model** to the collected function samples $\langle \lambda, c(\lambda) \rangle$
- Use the model to guide optimization, trading off **exploration vs exploitation**



Bayesian Optimization of a blackbox function in a nutshell

General approach

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



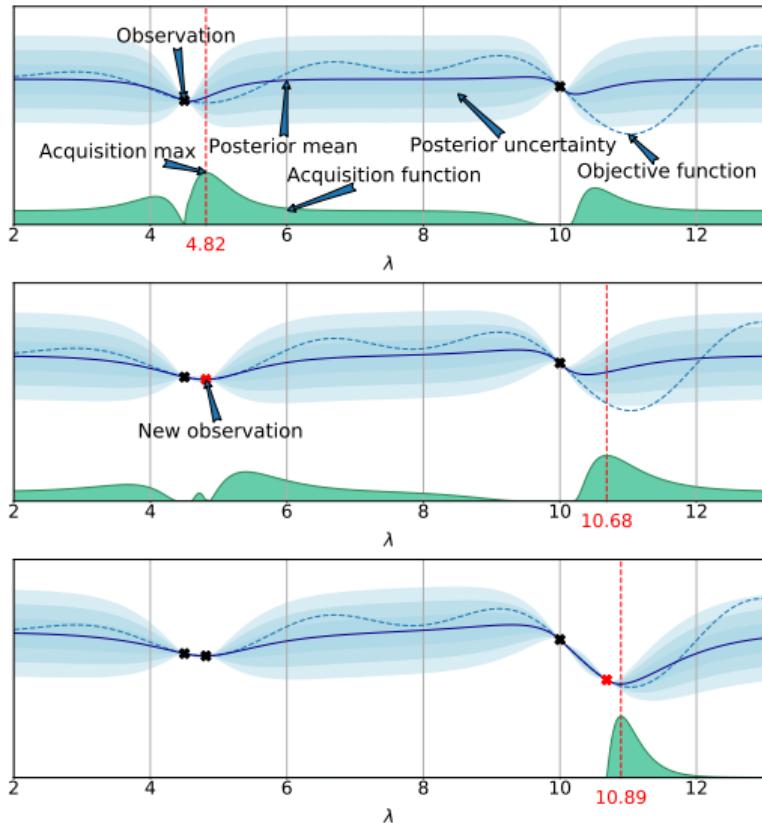
Bayesian Optimization of a blackbox function in a nutshell

General approach

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

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

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



Bayesian Optimization: Pseudocode

BO loop

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

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

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

Bayesian Optimization: Origin of the Name

- Bayesian optimization uses Bayes' theorem:

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

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

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

Bayesian Optimization: Origin of the Name

- Bayesian optimization uses Bayes' theorem:

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

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

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

- Meaning of the individual terms:

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

Bayesian Optimization: Advantages and Disadvantages

Advantages

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

Bayesian Optimization: Advantages and Disadvantages

Advantages

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

Disadvantages

- Overhead because of model training in each iteration
- Crucially relies on robust surrogate model
- Inherently sequential (in its basic form)

Learning Goals of this Lecture

After this lecture, students can ...

- Explain the basics of Bayesian optimization
- Derive simple acquisition functions
- Describe advanced 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

Outline

- 1 Introduction to Bayesian Optimization
- 2 Computationally Cheap Acquisition Functions**
- 3 Computationally Expensive Acquisition Functions
- 4 Surrogate Models
- 5 High-Dimensional Bayesian Optimization
- 6 Extensions to Bayesian Optimization
- 7 The Alternative Optimization Approach of the Tree-Parzen Estimator (TPE)
- 8 Success Stories
- 9 Further Reading

AutoML: Bayesian Optimization for Hyperparameter Optimization

Computationally Cheap Acquisition Functions

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

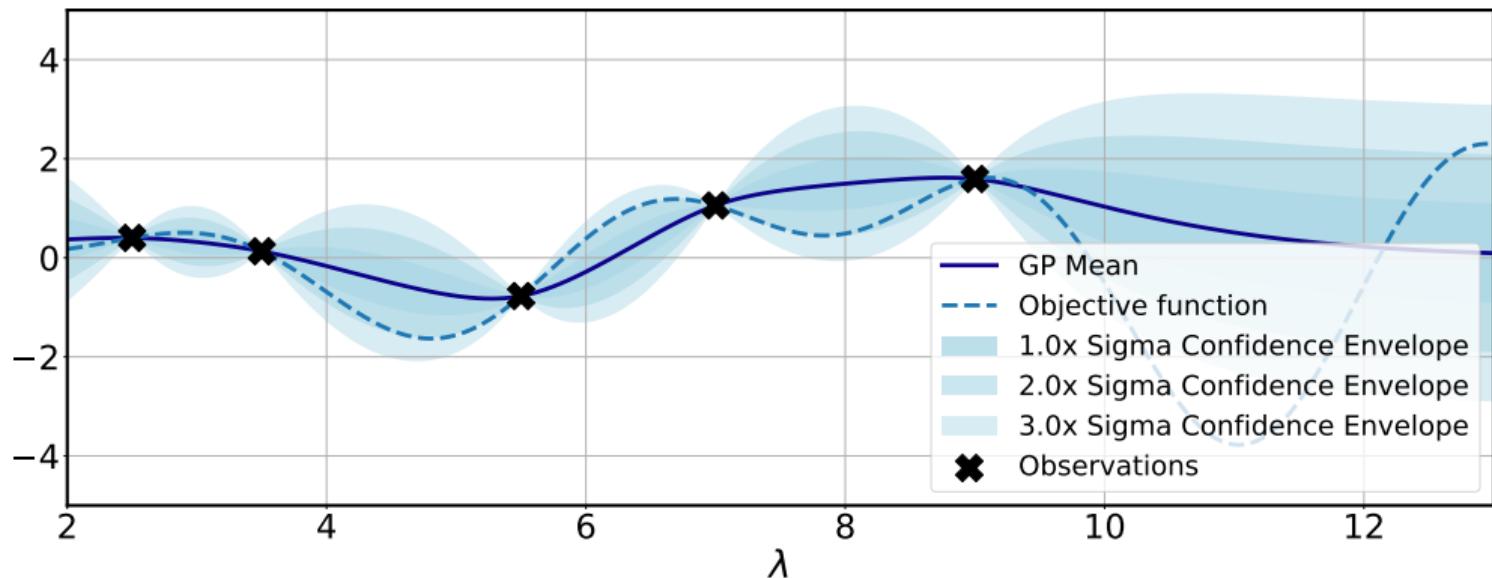
Acquisition Functions: the Basics

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

Acquisition Functions: the Basics

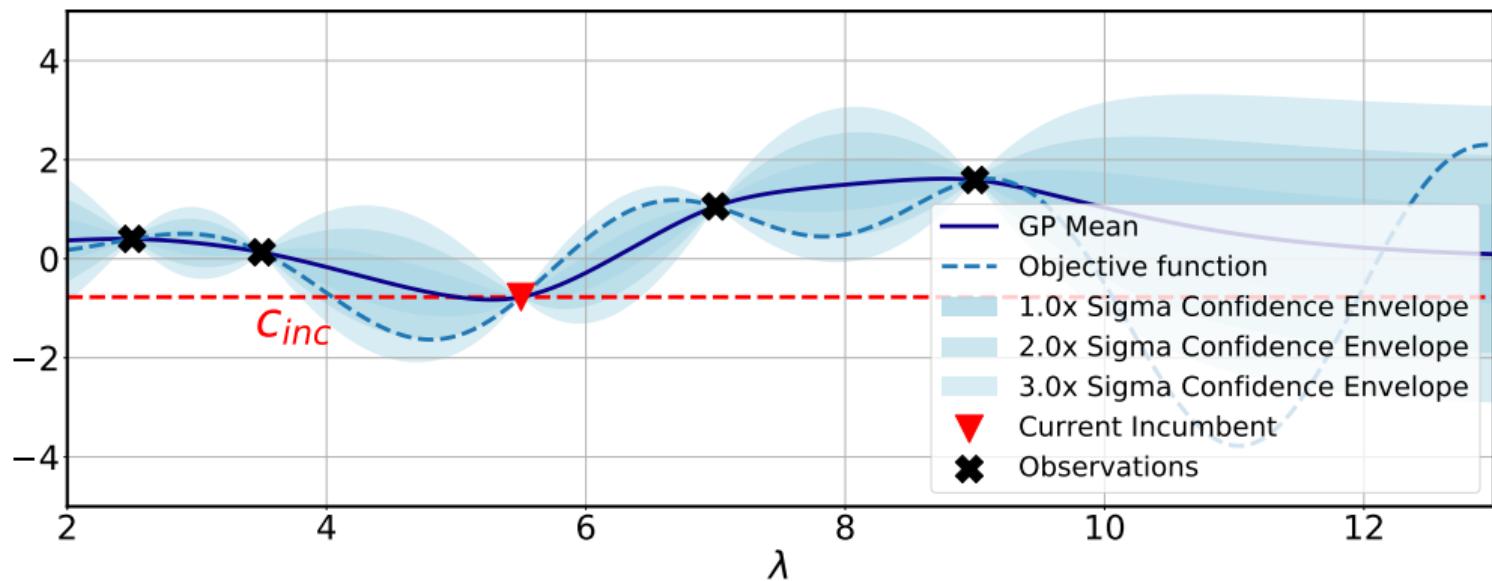
- Given the surrogate model $\hat{c}^{(t)}$ at the t -th iteration of BO, the acquisition function $u(\cdot)$ judges the utility (or usefulness) of evaluating f at $\lambda^{(t)} \in \Lambda$ next
- The acquisition function needs to trade off exploration and exploitation
 - E.g., just picking the λ with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

Probability of Improvement (PI): Concept



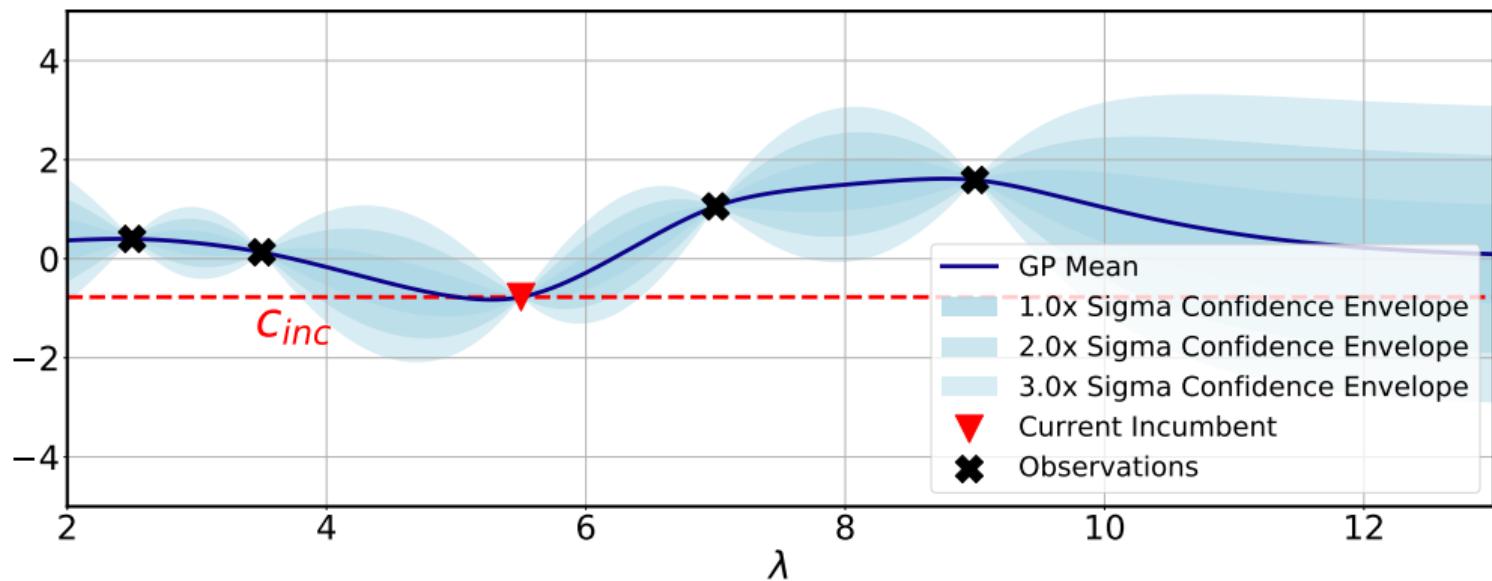
Given the surrogate fit at iteration t

Probability of Improvement (PI): Concept



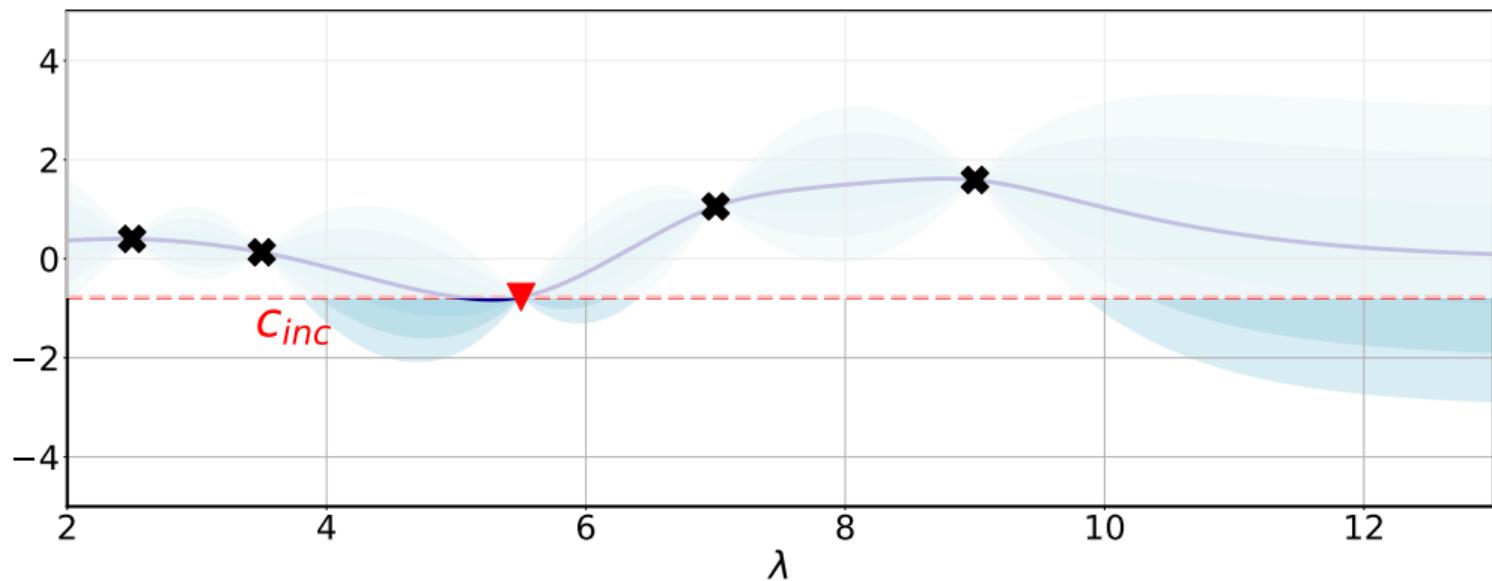
Current incumbent $\hat{\lambda}$ and its observed cost c_{inc}

Probability of Improvement (PI): Concept



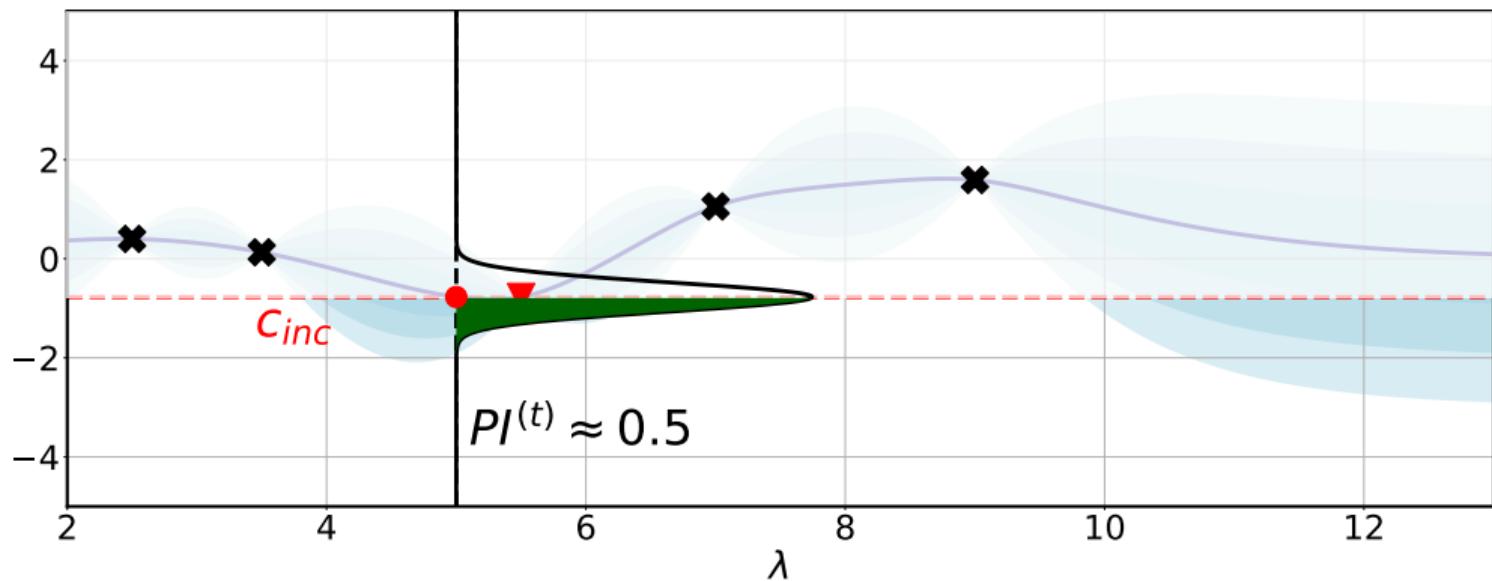
Now let's drop the objective function - it's unknown after all!

Probability of Improvement (PI): Concept



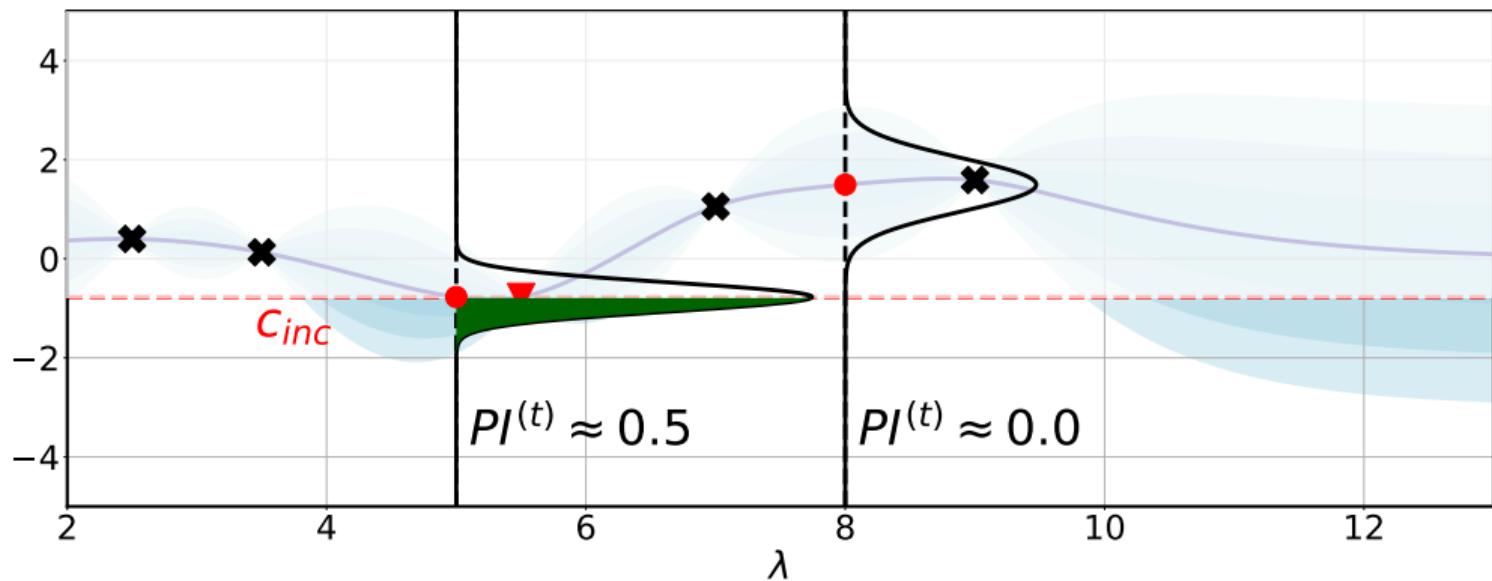
Intuitively, we care about the probability of improving over the current incumbent

Probability of Improvement (PI): Concept



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

Probability of Improvement (PI): Concept



PDF of a bad candidate configuration

Probability of Improvement (PI): Formal Definition

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

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

Probability of Improvement (PI): Formal Definition

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

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

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

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

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

Probability of Improvement (PI): Formal Definition

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

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

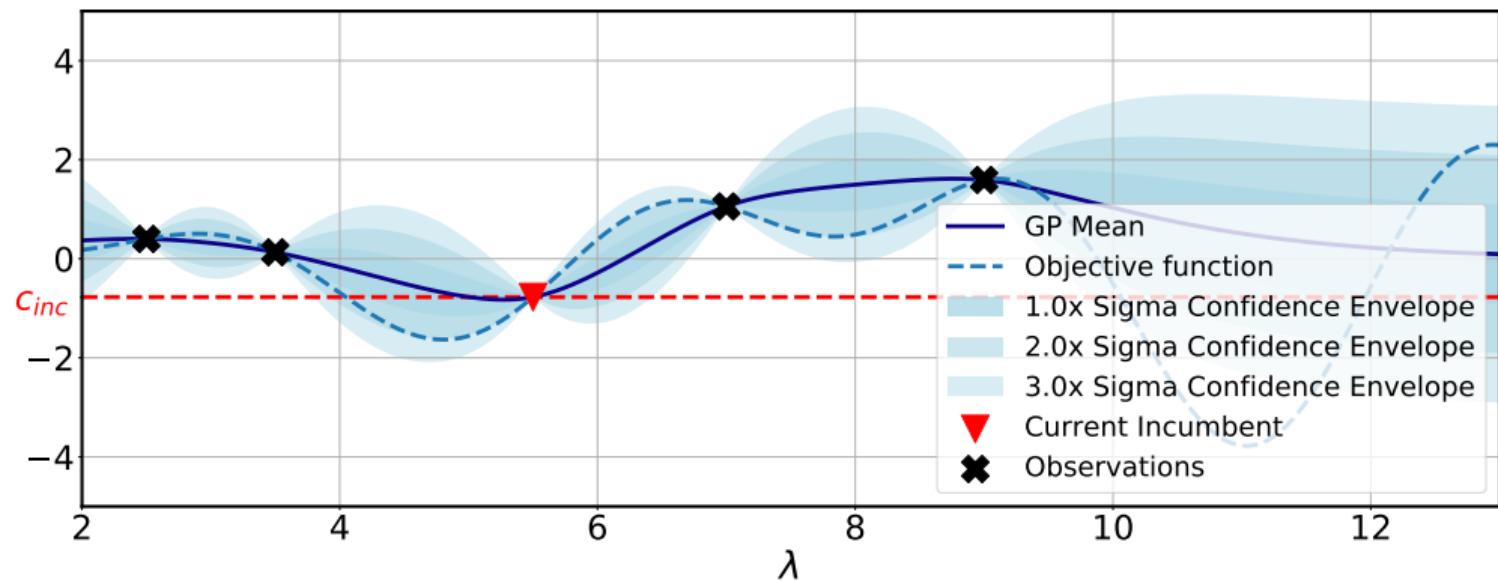
- Since the predictive distribution for $c(\lambda)$ is a Gaussian $\mathcal{N}(\mu^{(t-1)}(\lambda), \sigma^{2(t-1)}(\lambda))$, this can be written as:

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

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

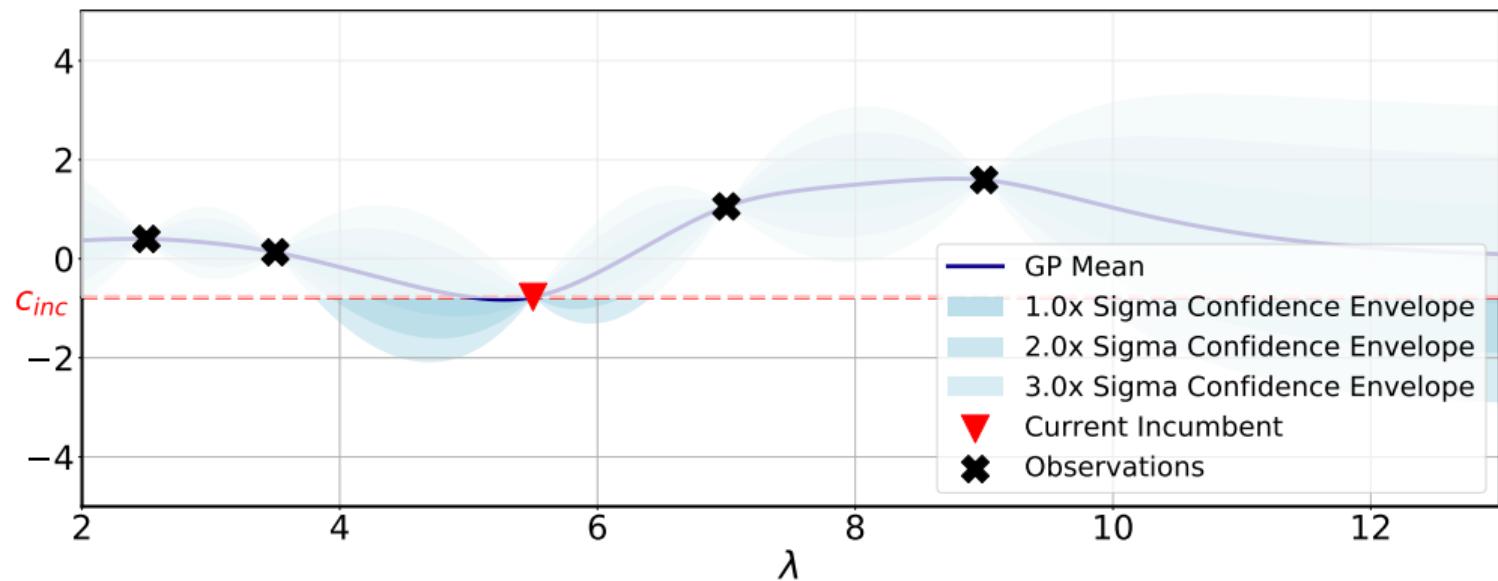
Choose $\lambda^{(t)} \in \arg \max_{\lambda \in \Lambda} (u_{PI}^{(t)}(\lambda))$

Expected Improvement (EI): Concept



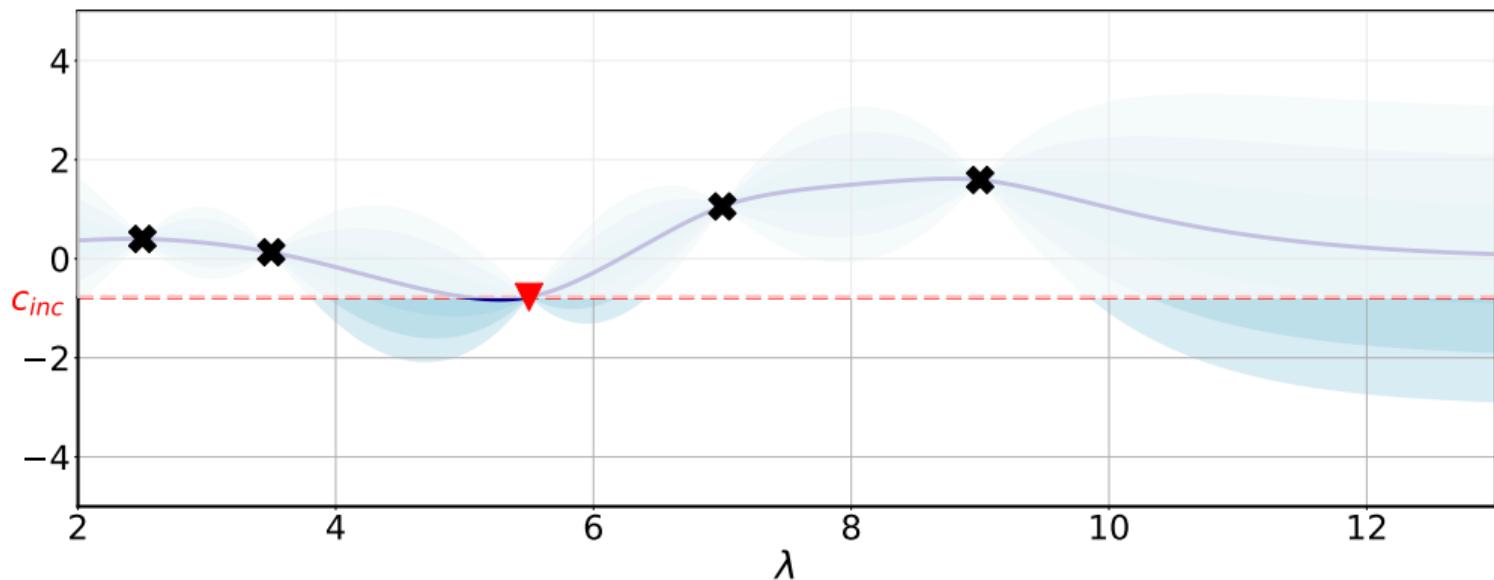
Given the surrogate fit at iteration t

Expected Improvement (EI): Concept



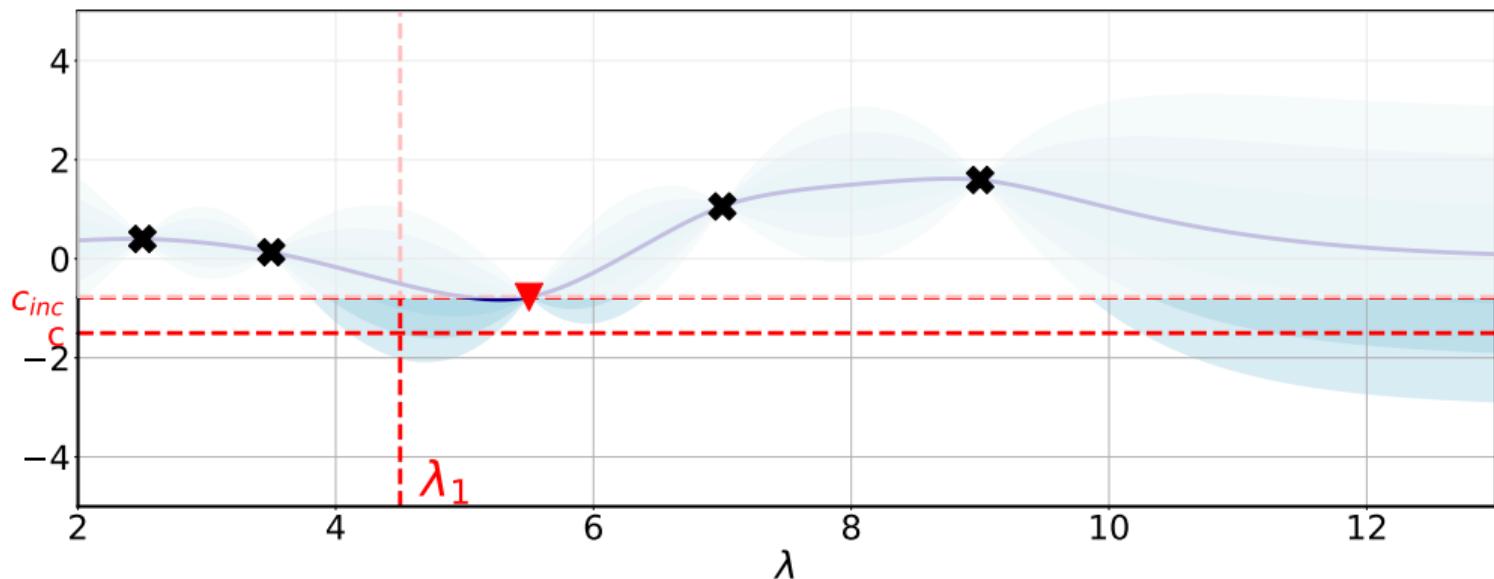
Region of probable improvement – but **how large** is the improvement?

Expected Improvement (EI): Concept



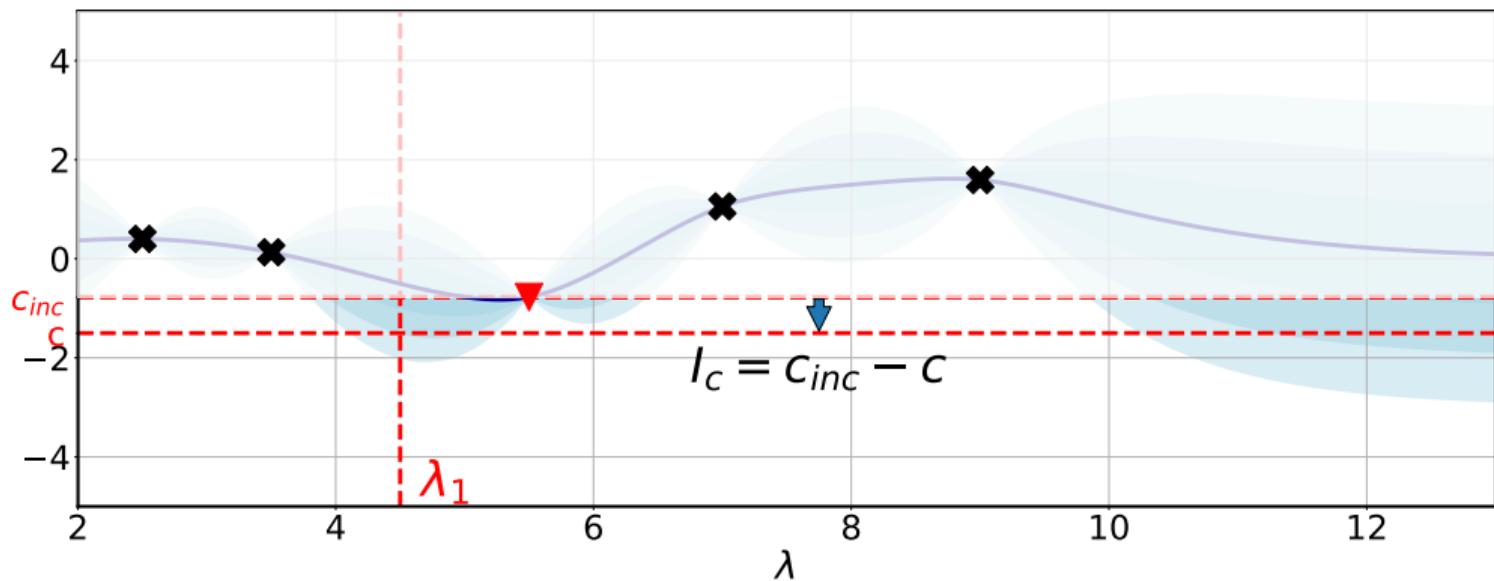
Region of probable improvement – but **how large** is the improvement?

Expected Improvement (EI): Concept



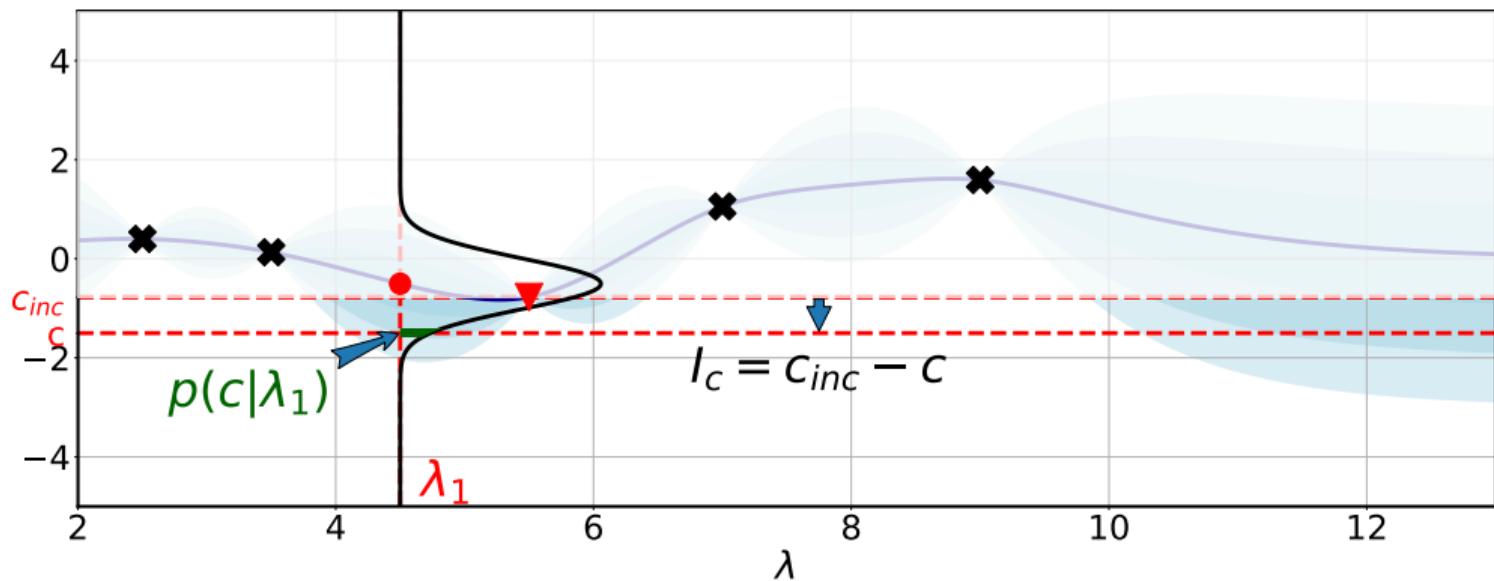
Hypothetical *real* cost c at a given λ - unknown in practice without evaluating

Expected Improvement (EI): Concept



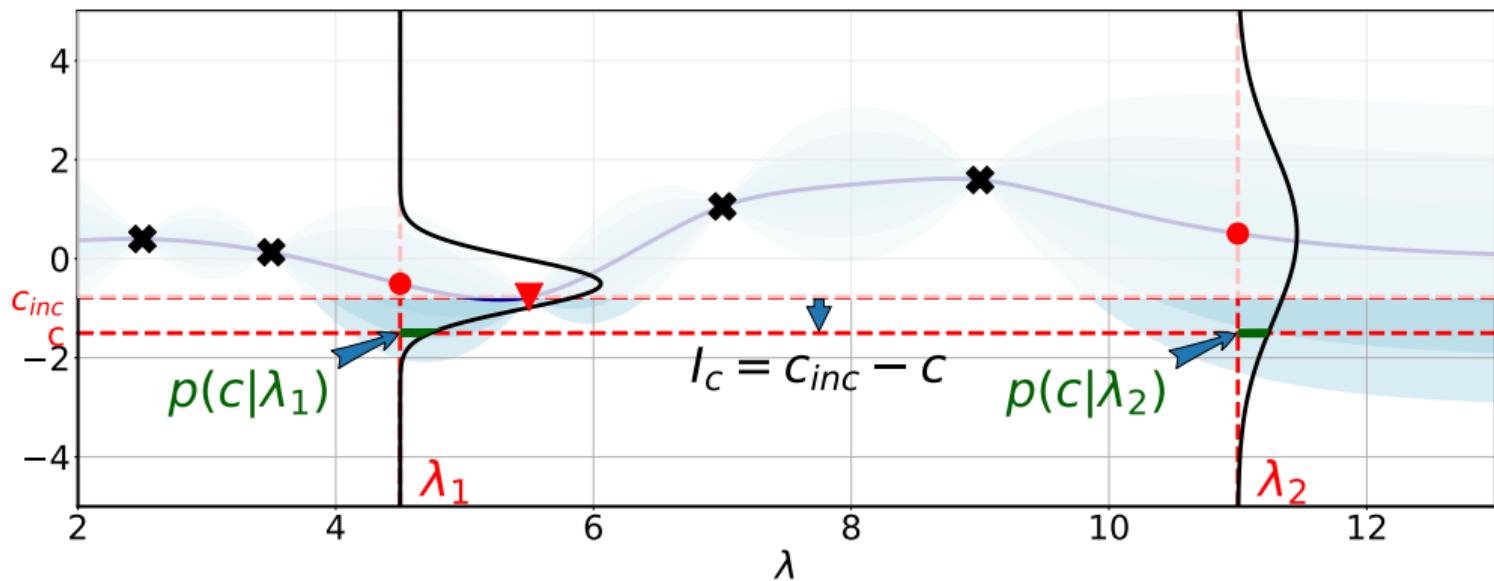
Given a hypothetical c , we can improve the improvement $I_c(\lambda)$

Expected Improvement (EI): Concept



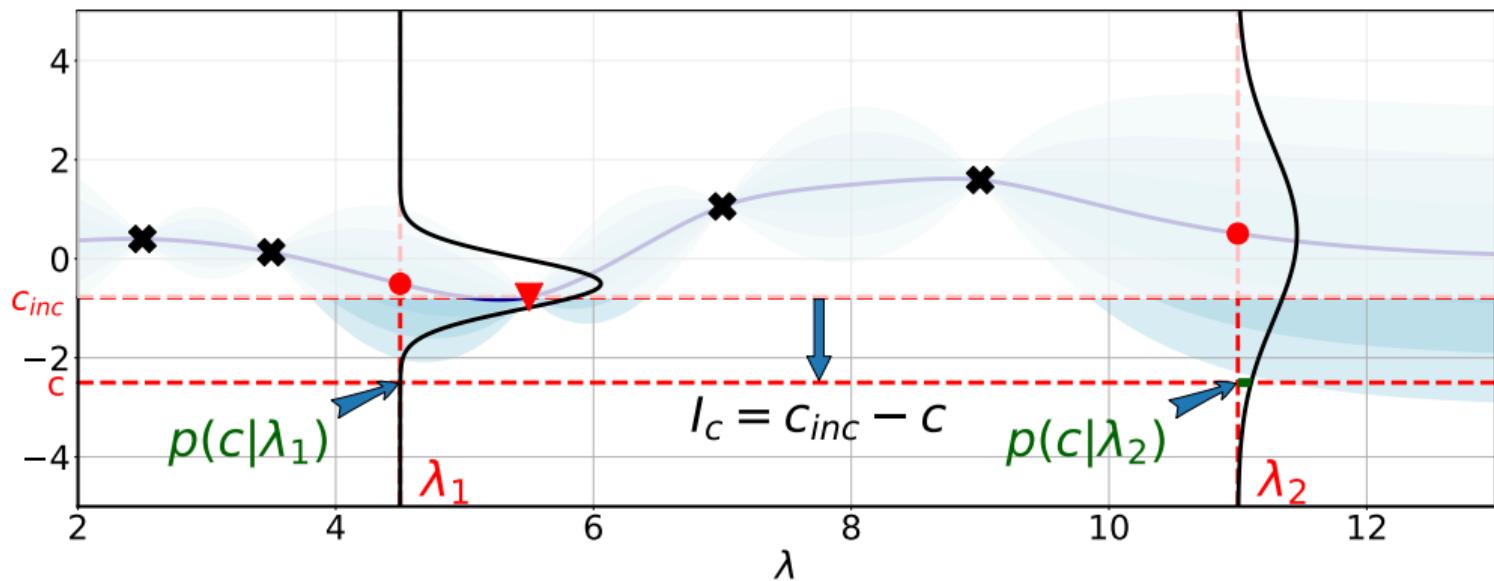
Given $\hat{c}(\lambda) = \mathcal{N}(\mu(\lambda), \sigma^2(\lambda))$, we can also compute $p(c|\lambda)$.

Expected Improvement (EI): Concept



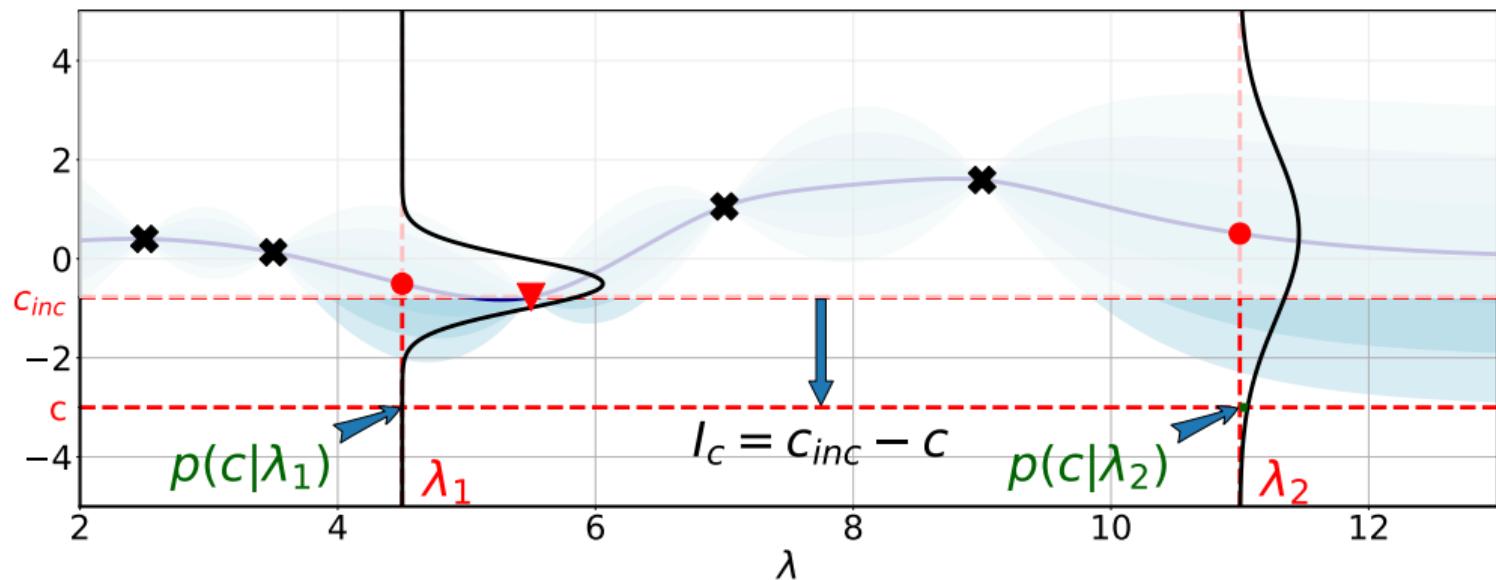
Compare the likelihood of a given improvement for two different configurations λ_1 and λ_2

Expected Improvement (EI): Concept



Now consider the likelihood of a larger improvement.

Expected Improvement (EI): Concept



Larger improvements are more likely in areas of high uncertainty.

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

Expected Improvement (EI): Formal Definition

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

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

- Expected Improvement is then defined as

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

Expected Improvement (EI): Formal Definition

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

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

- Expected Improvement is then defined as

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

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

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

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

Expected Improvement (EI): Formal Definition

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

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

- Expected Improvement is then defined as

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

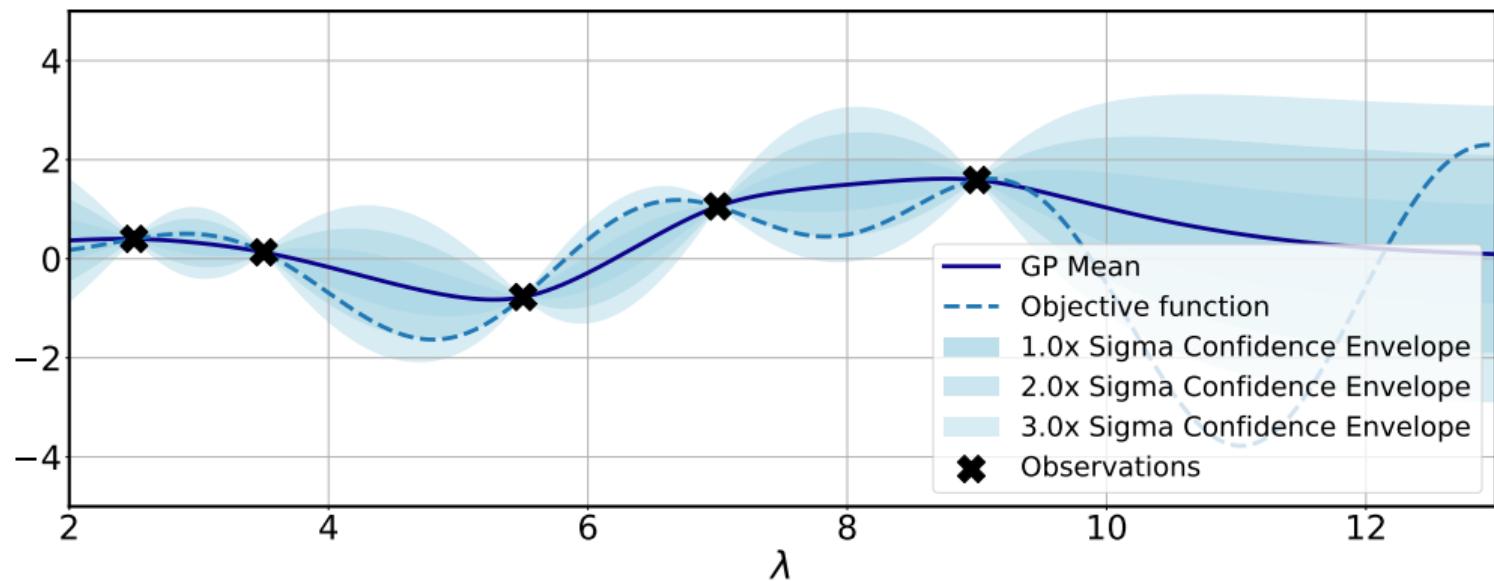
- Since the posterior distribution of $\hat{c}(\boldsymbol{\lambda})$ is a Gaussian, EI can be computed in closed form (see exercise):

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

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

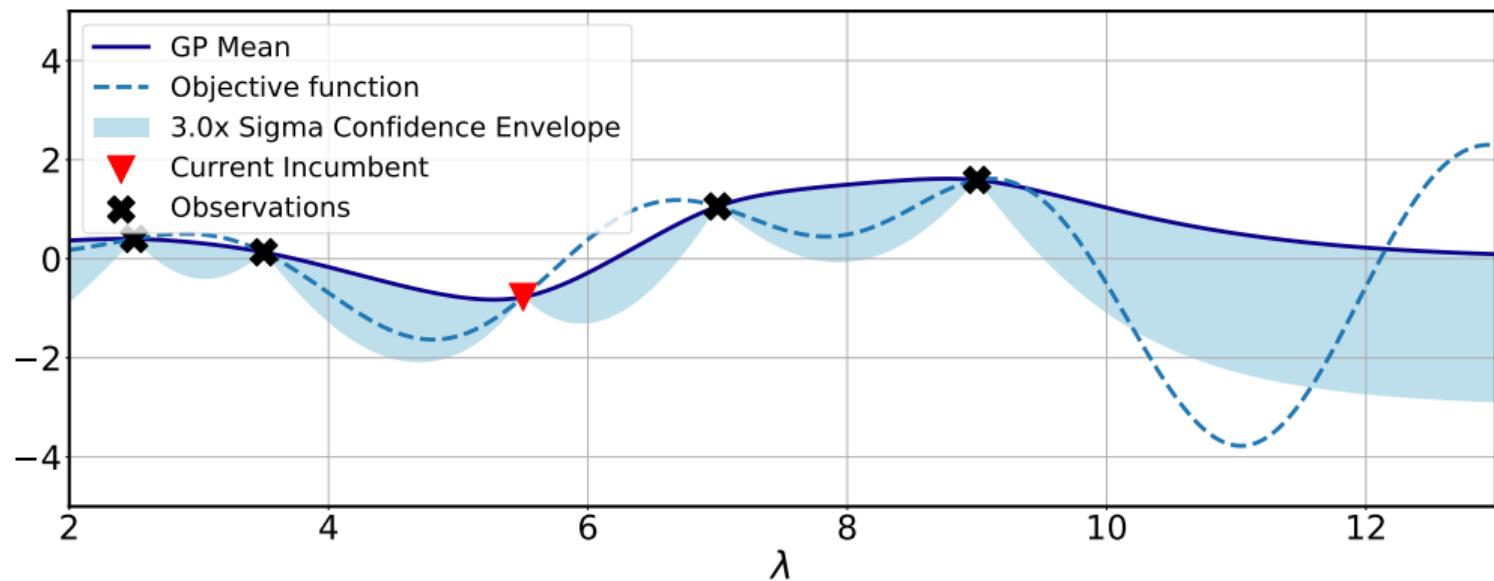
Choose $\boldsymbol{\lambda}^{(t)} \in \arg \max_{\boldsymbol{\lambda} \in \Lambda} (u_{EI}^{(t)}(\boldsymbol{\lambda}))$

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



Given the surrogate fit at iteration t

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



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

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

- We define the Lower Confidence Bound as

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

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

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

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

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

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

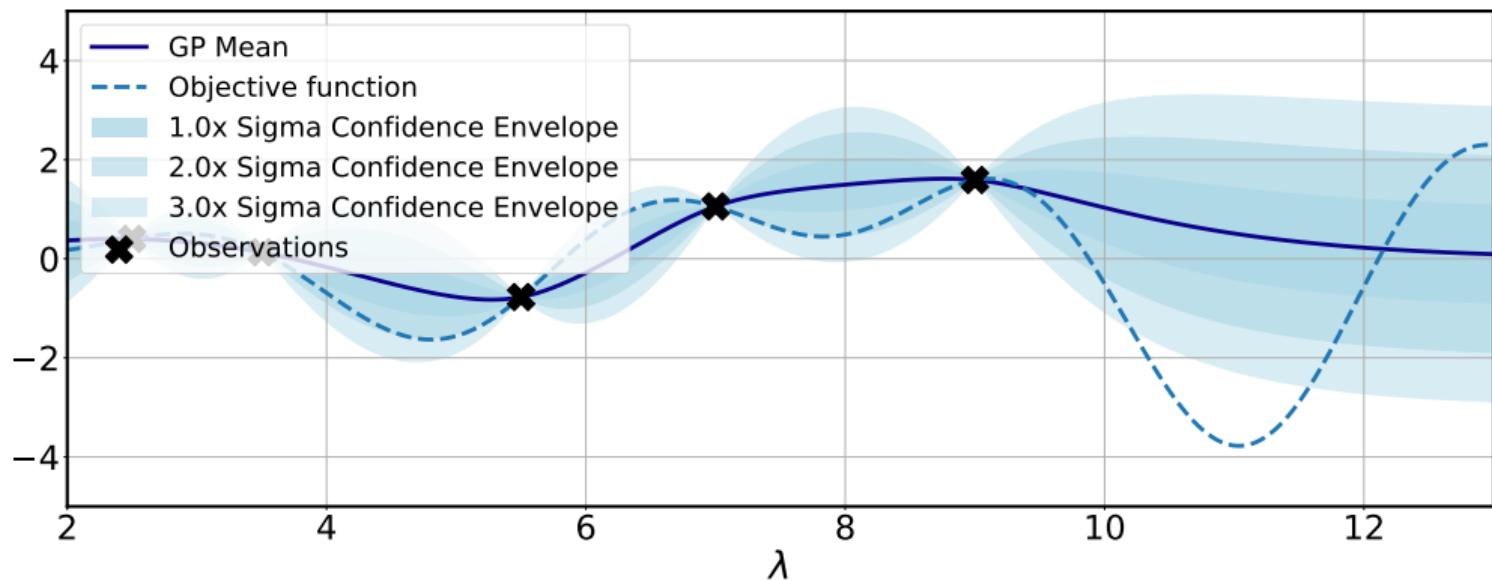
- One can schedule α (e.g., increase it over time [Srinivas et al. 2009])

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

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

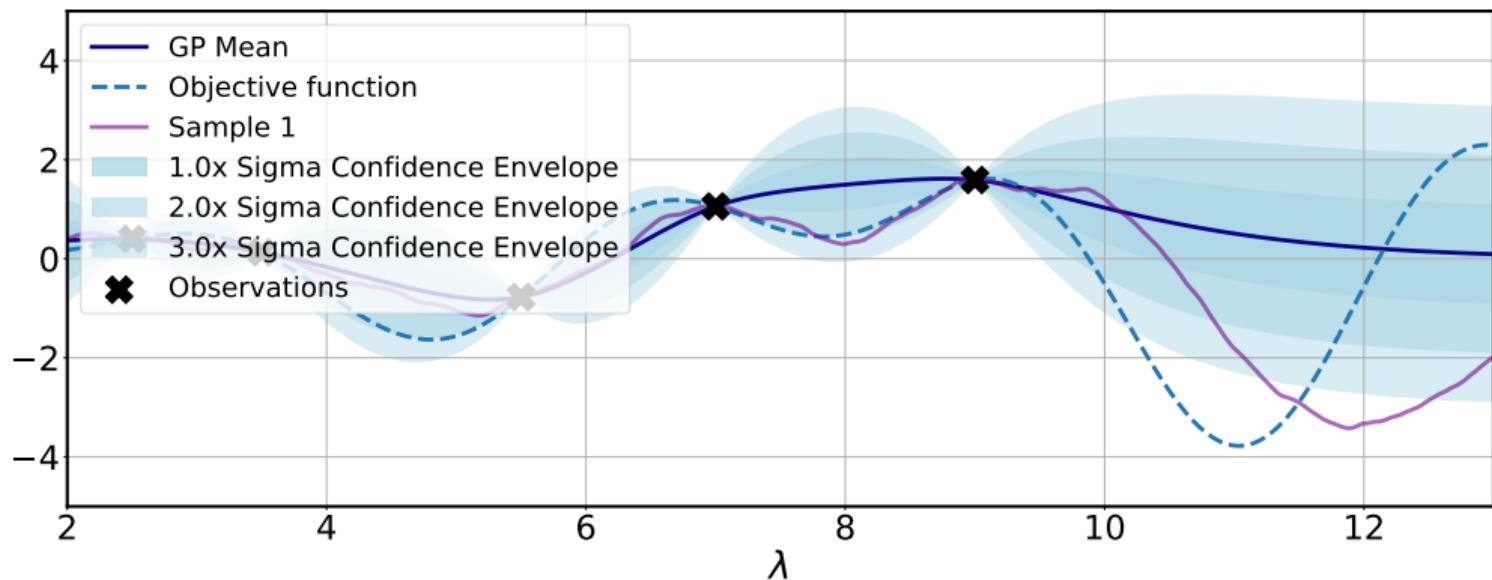
- ▶ $u_{UCB}^{(t)}(\boldsymbol{\lambda}) = \mu^{(t)}(\boldsymbol{\lambda}) + \alpha \sigma^{(t)}(\boldsymbol{\lambda})$
- ▶ For UCB, one would choose $\boldsymbol{\lambda}^{(t)} \in \arg \max_{\boldsymbol{\lambda} \in \Lambda} (u_{UCB}^{(t)}(\boldsymbol{\lambda}))$

Thompson Sampling (TS): Concept



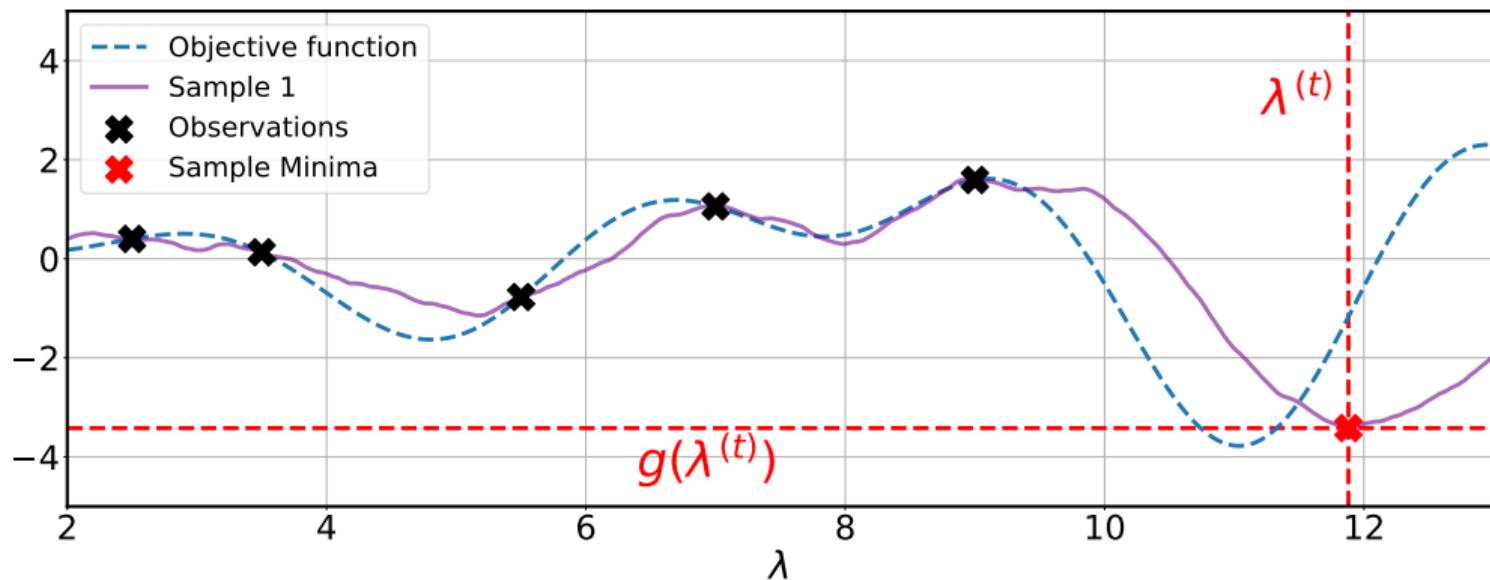
Given the surrogate at iteration t fit on dataset $\mathcal{D}^{(t-1)}$

Thompson Sampling (TS): Concept



Draw a sample g from the predictive surrogate model

Thompson Sampling (TS): Concept



Then choose the minimum of this sample to evaluate at next

Thompson Sampling (TS): Pseudocode

Bayesian Optimization using Thompson Sampling

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

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

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

Questions to Answer for Yourself / Discuss with Friends

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

Outline

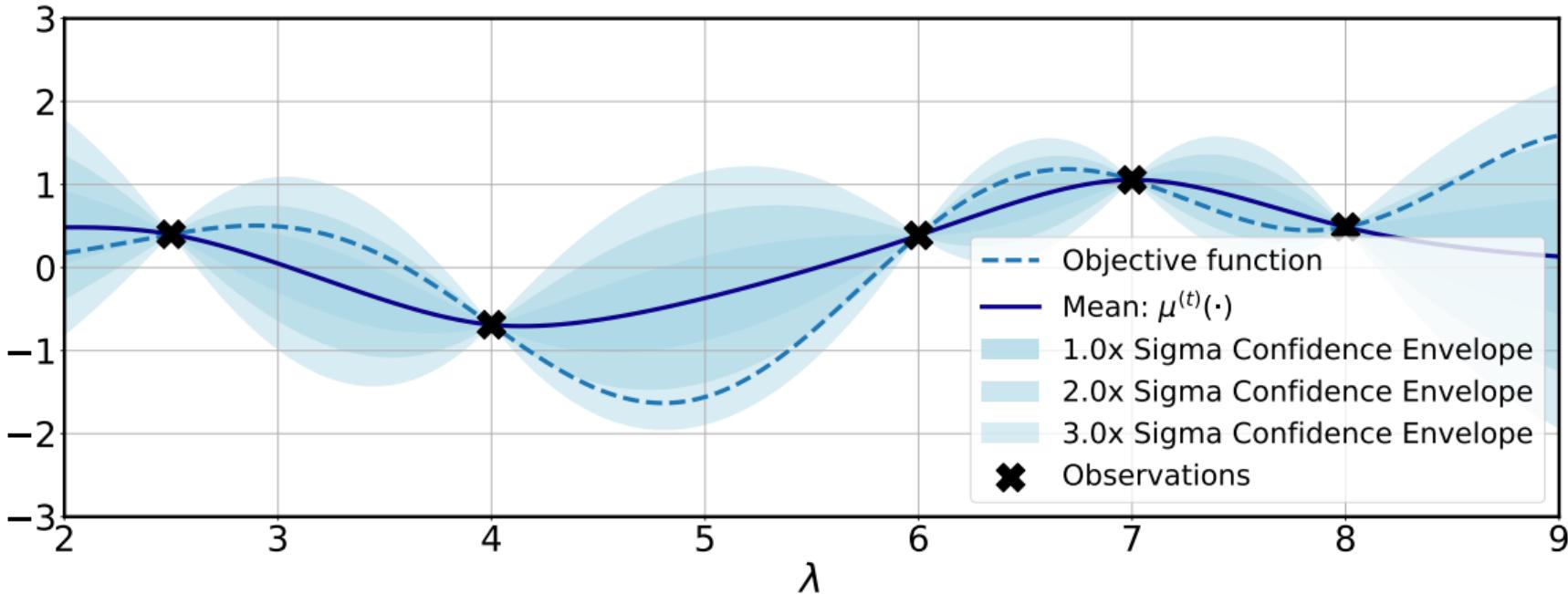
- 1 Introduction to Bayesian Optimization
- 2 Computationally Cheap Acquisition Functions
- 3 Computationally Expensive Acquisition Functions**
- 4 Surrogate Models
- 5 High-Dimensional Bayesian Optimization
- 6 Extensions to Bayesian Optimization
- 7 The Alternative Optimization Approach of the Tree-Parzen Estimator (TPE)
- 8 Success Stories
- 9 Further Reading

AutoML: Bayesian Optimization for Hyperparameter Optimization

Computationally Expensive Acquisition Functions

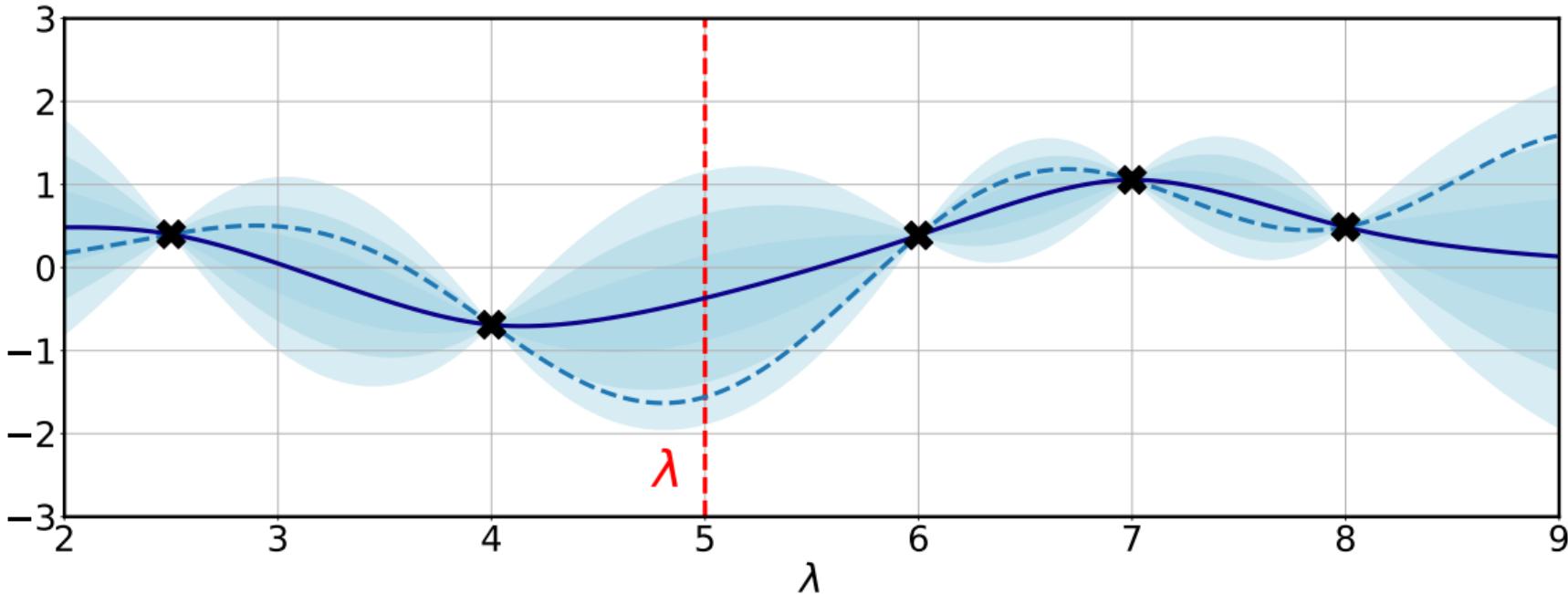
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

A Computationally Expensive Step: One-Step Look Ahead



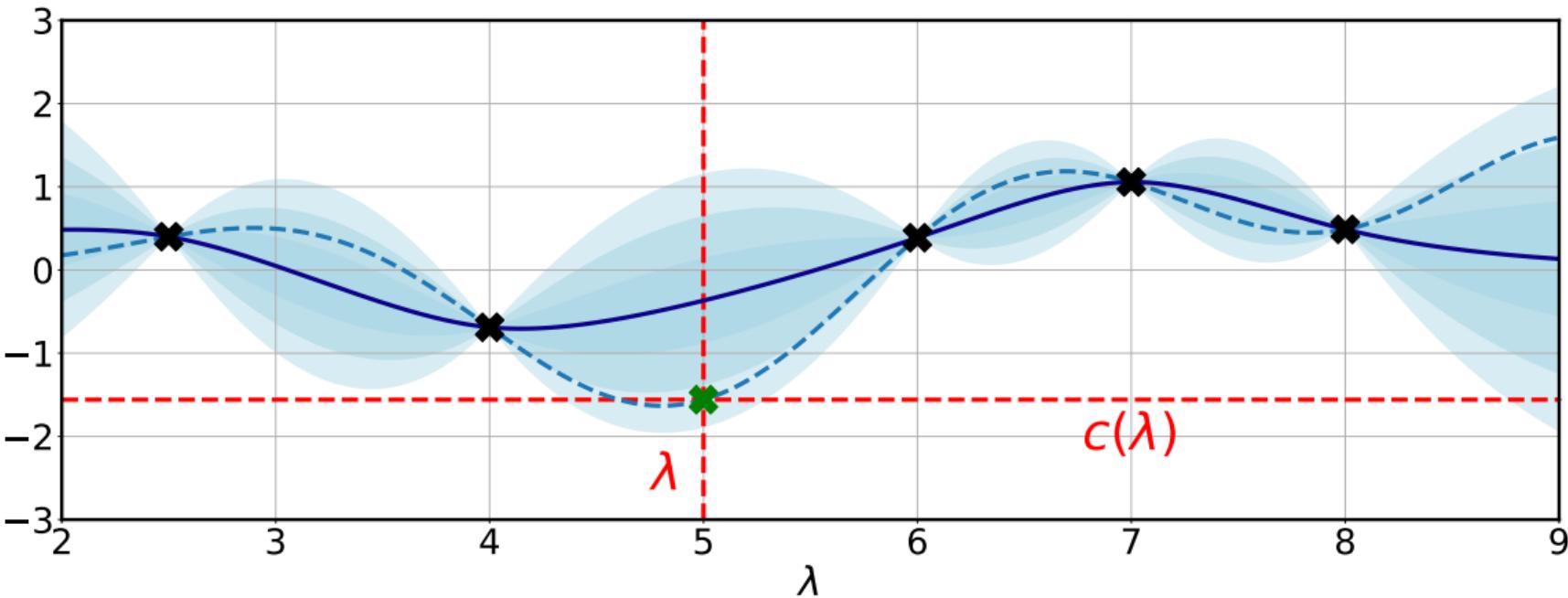
Given the surrogate $\hat{c}^{(t)}$ fit at iteration t

A Computationally Expensive Step: One-Step Look Ahead



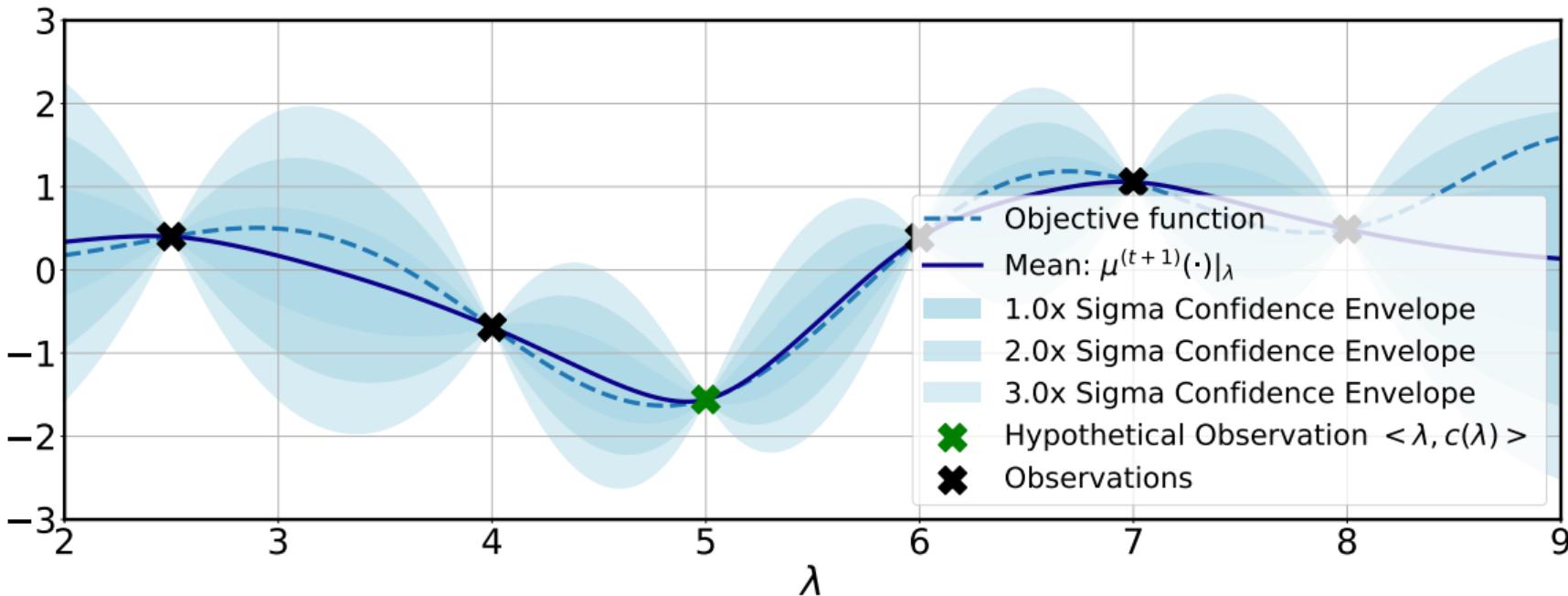
Imagine that we sample at a random configuration λ

A Computationally Expensive Step: One-Step Look Ahead



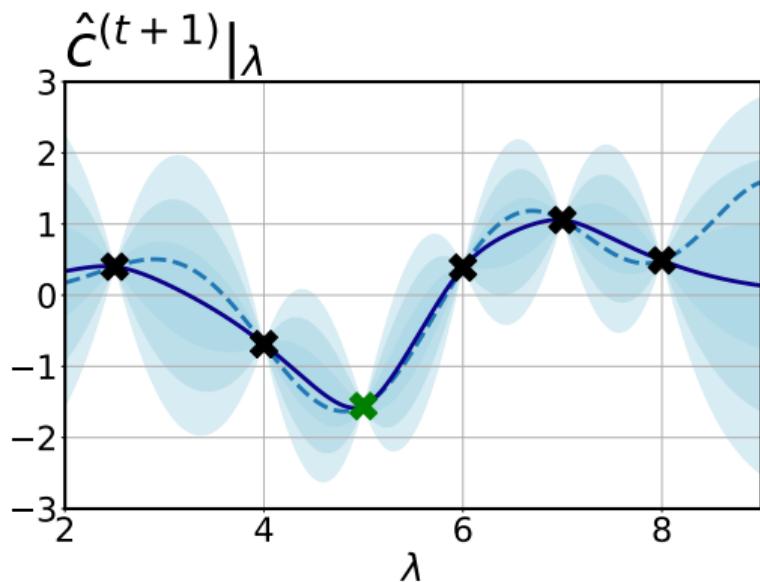
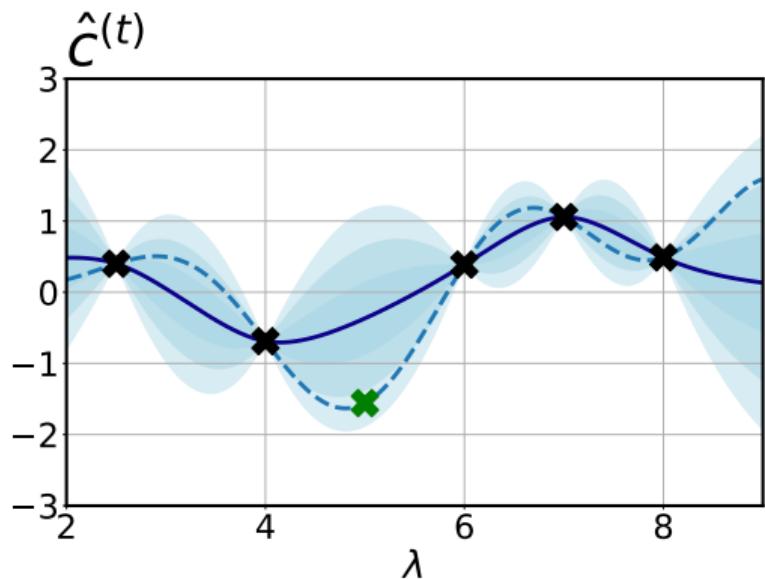
We would then observe the cost $c(\lambda)$ at this imaginary configuration λ

A Computationally Expensive Step: One-Step Look Ahead



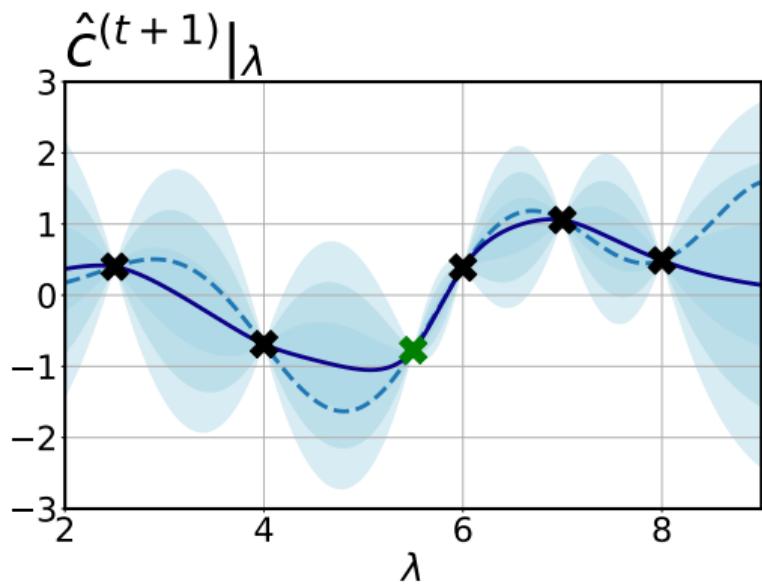
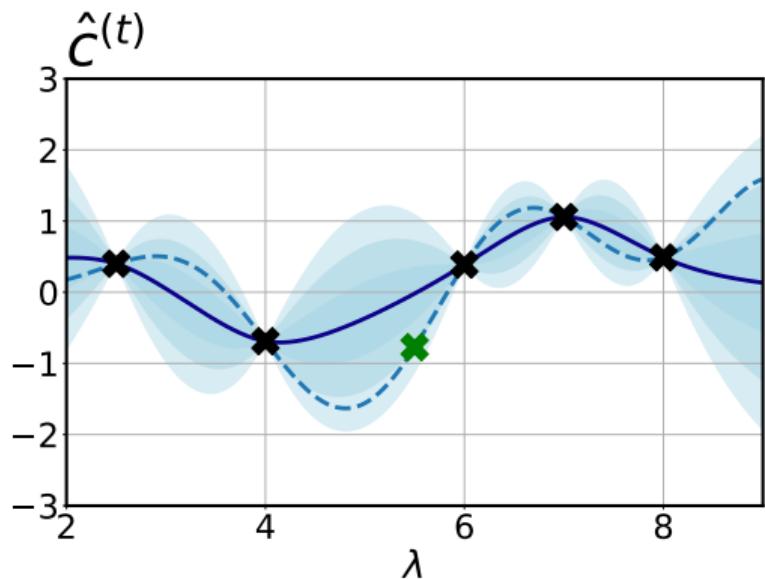
With this hypothetical data point at λ , we'd have this 1-step lookahead surrogate $\hat{c}^{(t+1)}|_{\lambda}(\cdot)$

Visualization of How Different the Lookahead Surrogate Can Be



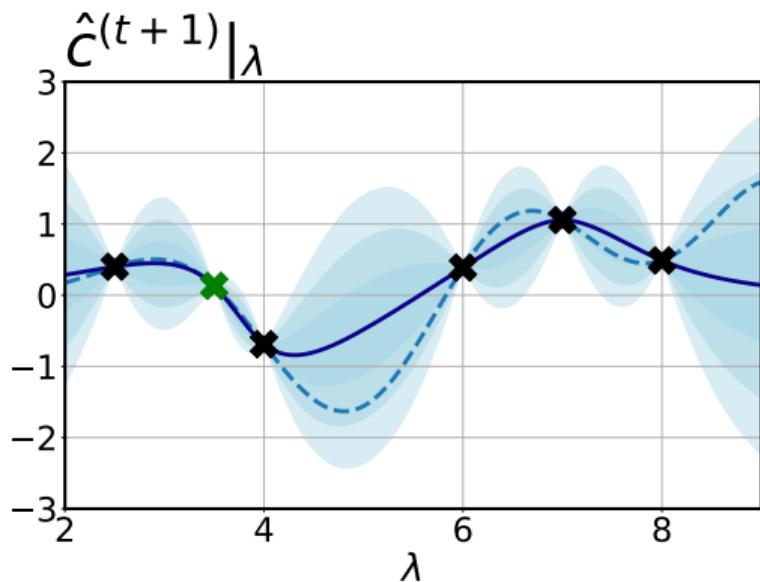
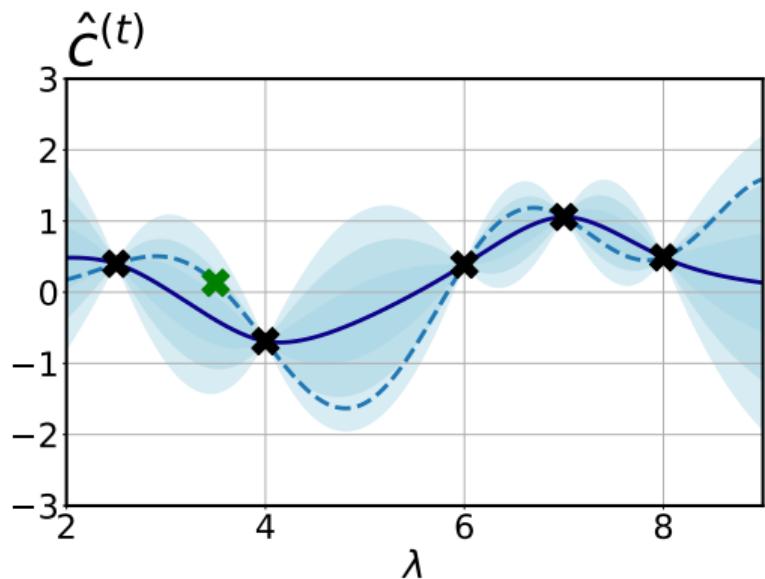
A comparison of $\hat{c}^{(t)}(\cdot)$ and $\hat{c}^{(t+1)}|_{\lambda}(\cdot)$ for a given λ .

Visualization of How Different the Lookahead Surrogate Can Be



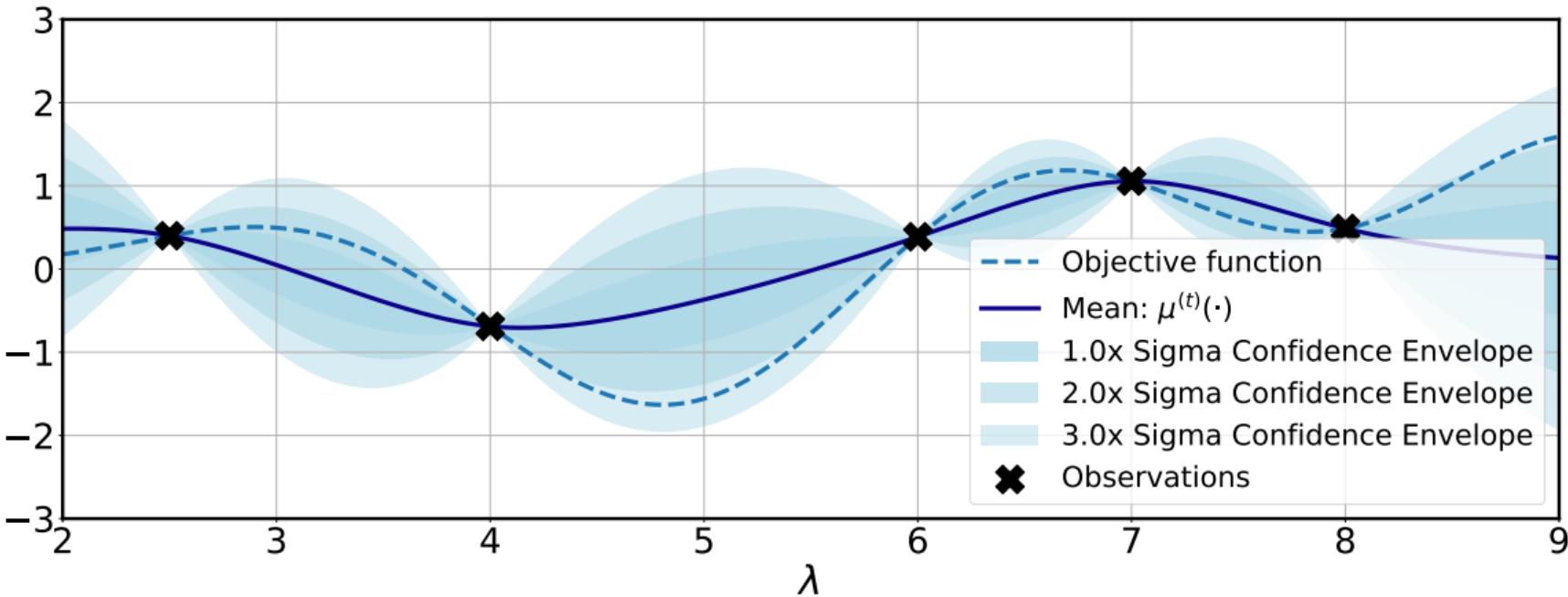
A comparison of $\hat{C}^{(t)}(\cdot)$ and $\hat{C}^{(t+1)}|_{\lambda}(\cdot)$ for a given λ .

Visualization of How Different the Lookahead Surrogate Can Be



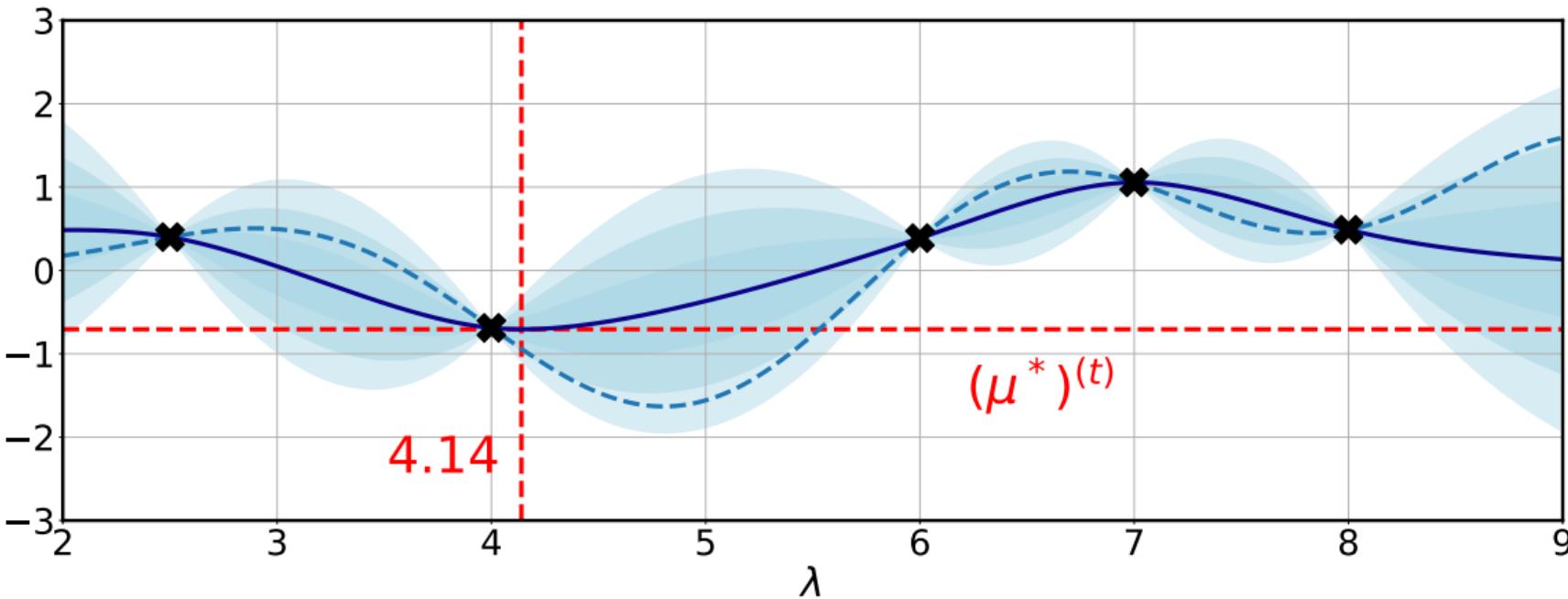
A comparison of $\hat{c}^{(t)}(\cdot)$ and $\hat{c}^{(t+1)}|_{\lambda}(\cdot)$ for a given λ .

Knowledge Gradient (KG): Concept



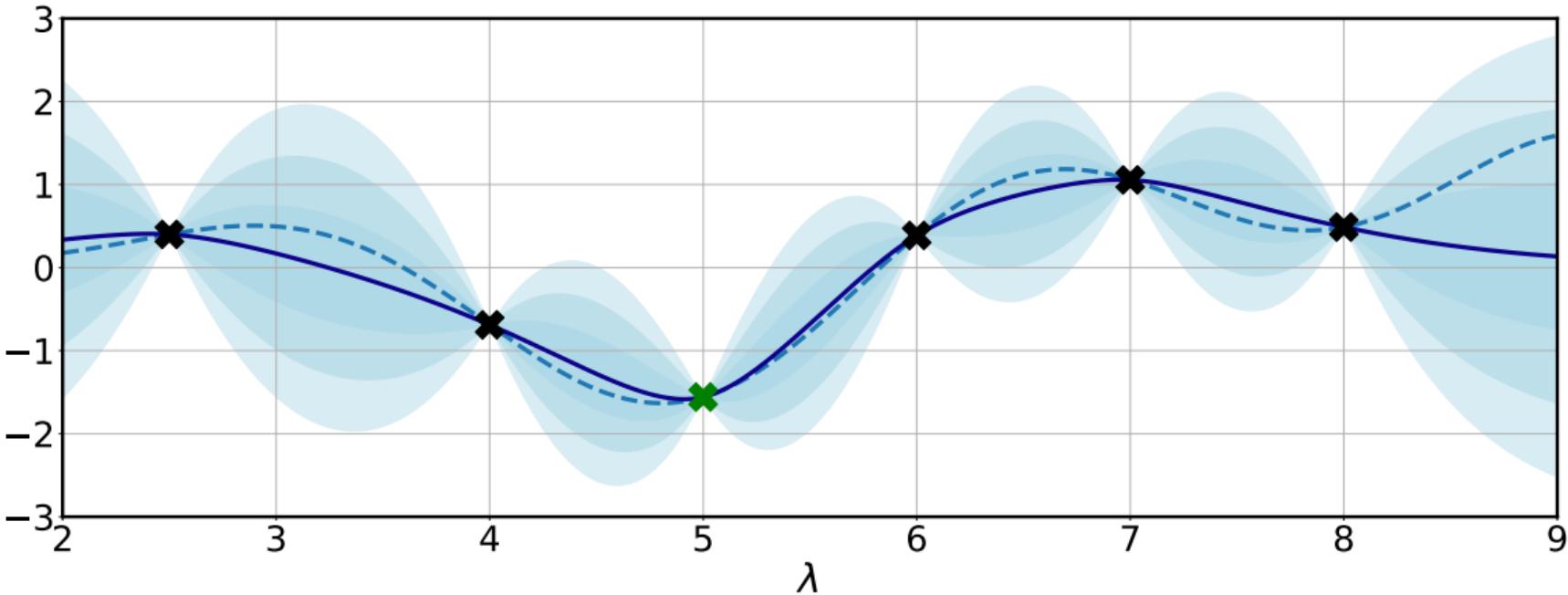
Given the surrogate $\hat{c}(\lambda) = \mathcal{N}(\mu(\lambda), \sigma^2(\lambda))$ fit at iteration t

Knowledge Gradient (KG): Concept



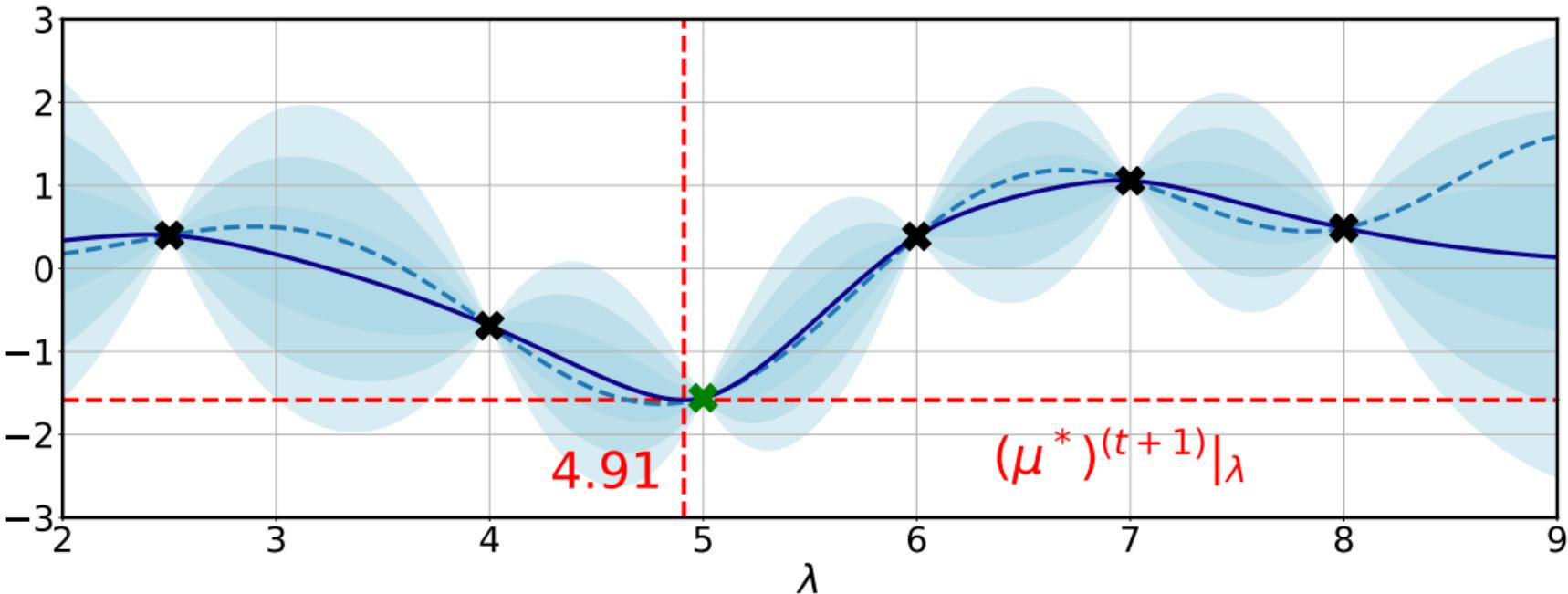
If we are risk-neutral, we'd return $\arg \min_{\lambda} (\mu(\lambda))^{(t)}$ as incumbent, with value $(\mu^*)^{(t)}$

Knowledge Gradient (KG): Concept



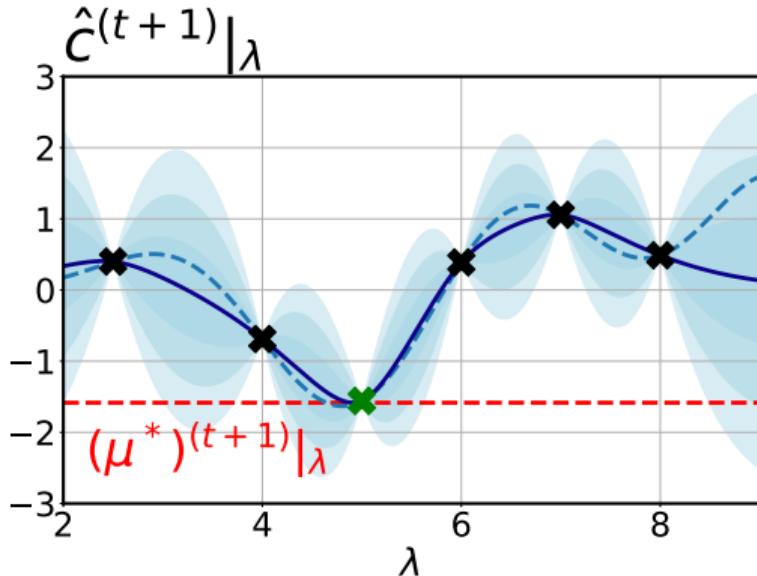
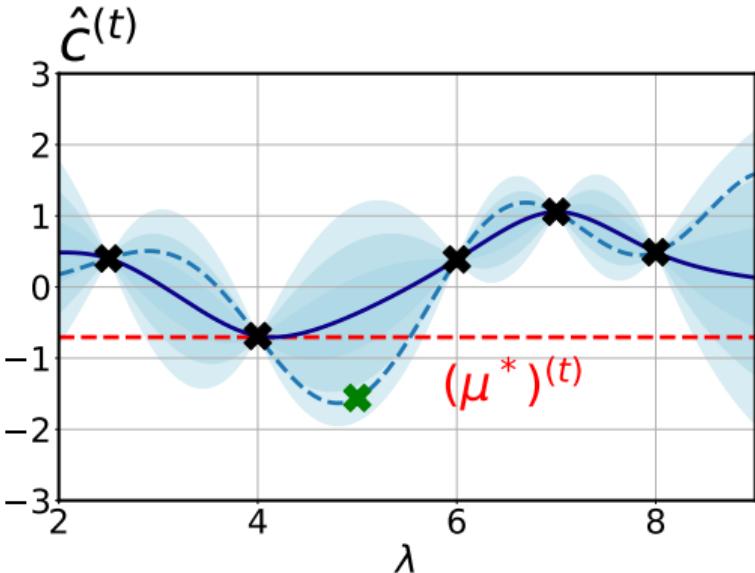
If we perform a one-step look-ahead for configuration λ , we would get $\hat{c}^{(t+1)}|_{\lambda}$

Knowledge Gradient (KG): Concept



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

Knowledge Gradient (KG): Concept



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

Knowledge Gradient (KG): Formal Definition

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

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

Knowledge Gradient (KG): Formal Definition

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

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

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

Knowledge Gradient: Pseudocode for Monte Carlo Approximation

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

Sampling Based Knowledge Gradient Acquisition Function

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

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

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

Knowledge Gradient: Pseudocode for Monte Carlo Approximation

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

Sampling Based Knowledge Gradient Acquisition Function

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

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

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

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

Entropy Search Preliminaries

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

Entropy Search Preliminaries

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

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

Entropy Search Preliminaries

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

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

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

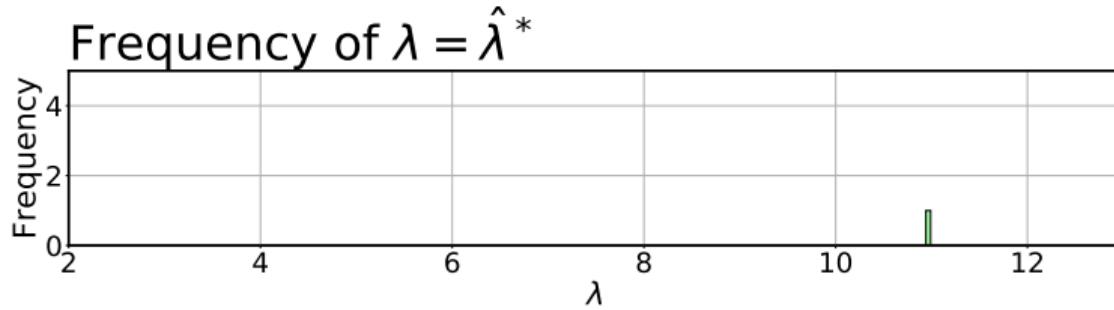
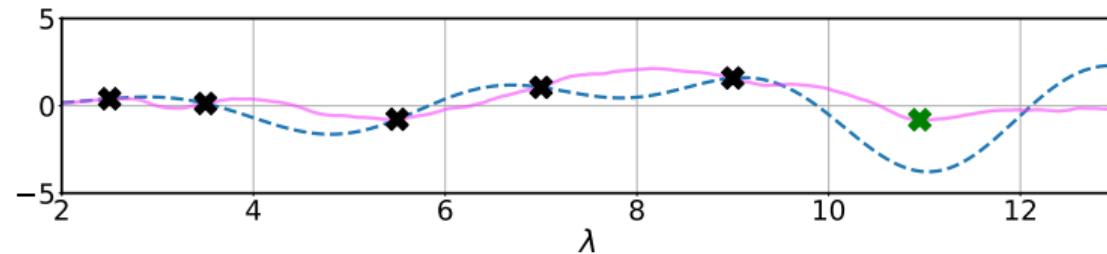
Entropy Search Preliminaries

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

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

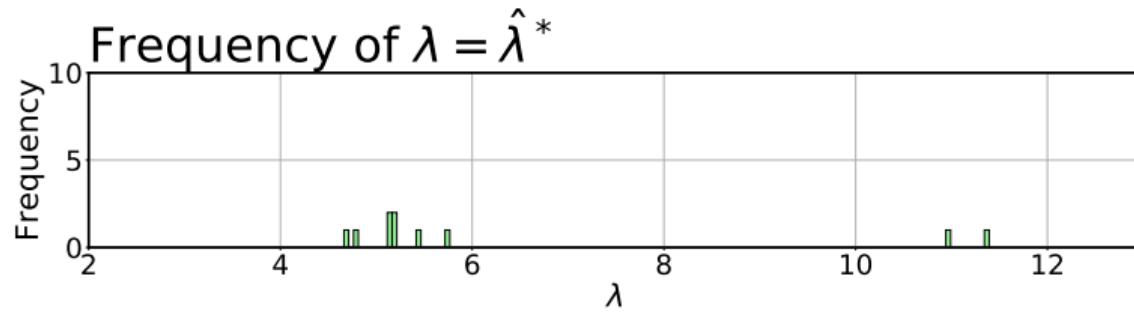
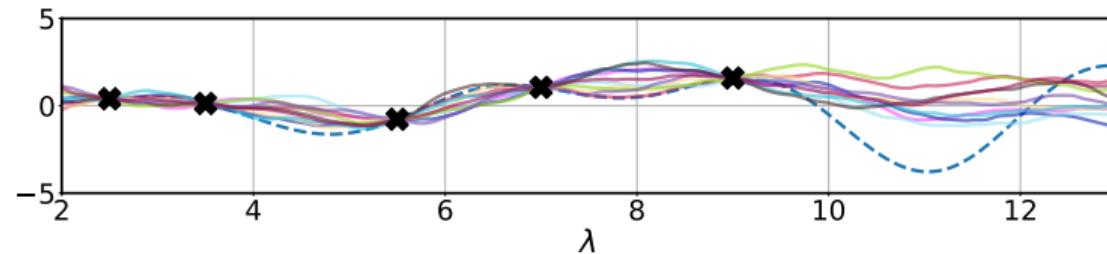
- Our uncertainty is then captured by the entropy $H(p_{min}(\cdot | \mathcal{D}))$ of the p_{min} distribution
- Minimizing $H(p_{min}(\cdot | \mathcal{D}))$ yields a peaked p_{min} distribution, i.e., strong knowledge about the location of λ^*

Entropy Search: Visualization of the p_{min} Distribution



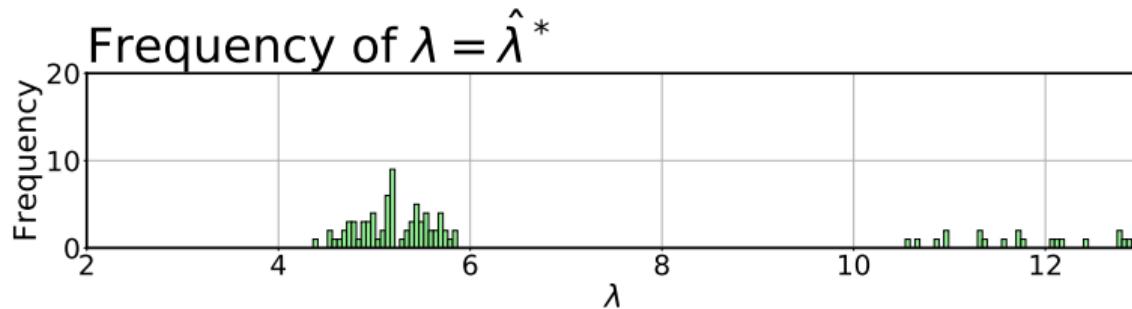
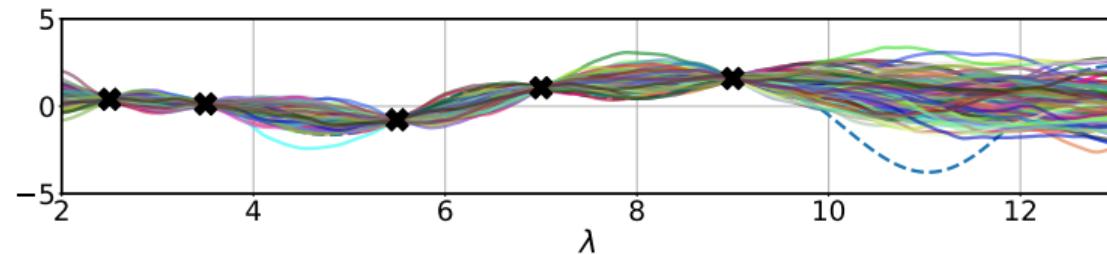
For each sample drawn from \hat{c} , we can compute where λ^* lies

Entropy Search: Visualization of the p_{min} Distribution



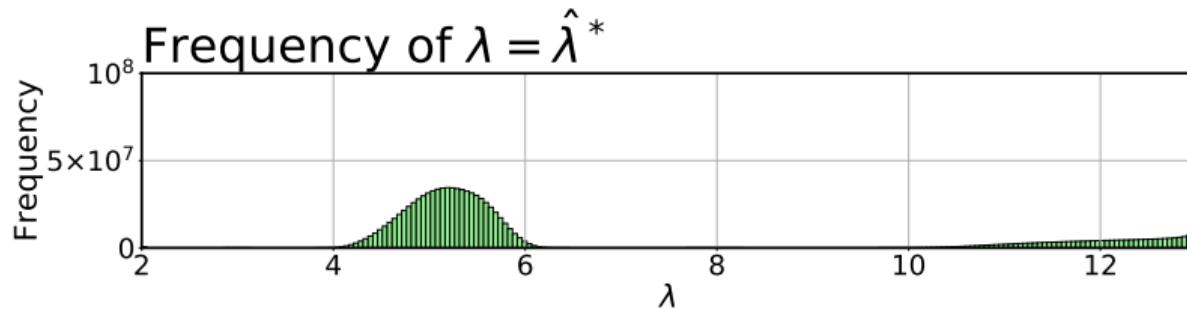
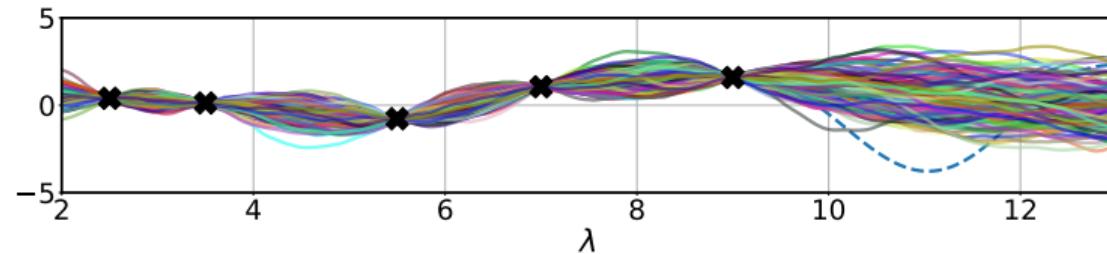
For each sample drawn from \hat{c} , we can compute where λ^* lies

Entropy Search: Visualization of the p_{min} Distribution



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

Entropy Search: Visualization of the p_{min} Distribution



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

Entropy Search: Formal Definition

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

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

Entropy Search: Formal Definition

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

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

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

Entropy Search: Formal Definition

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

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

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

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

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

Entropy Search: Pseudocode for Monte Carlo Approximation

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

Sampling Based Entropy Search Acquisition Function

Require : Surrogate \hat{c} , candidate configuration $\boldsymbol{\lambda}$, finite set of representer points Λ_r , dataset \mathcal{D}

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

```
1 for  $s = 1$  to  $S$  do
2   Sample  $\tilde{c}_s \sim \hat{c}(\boldsymbol{\lambda})$ ;  $\hat{c}_s \leftarrow$  Update  $\hat{c}$  with  $\{\langle \boldsymbol{\lambda}, \tilde{c}_s \rangle\}$ 
3   Initialize  $F[\boldsymbol{\lambda}] = 0 \quad \forall \boldsymbol{\lambda}' \in \Lambda_r$ 
4   for  $n = 1$  to  $N$  do
5     Sample  $g_n \sim \hat{c}_s$ 
6      $\boldsymbol{\lambda}_s \leftarrow \arg \min_{\boldsymbol{\lambda}' \in \Lambda_r} g_n$ 
7      $F[\boldsymbol{\lambda}_s] \leftarrow F[\boldsymbol{\lambda}_s] + 1$ 
8    $p_{\min,s}(\boldsymbol{\lambda}') \leftarrow F_{\boldsymbol{\lambda}'} / N \quad \forall \boldsymbol{\lambda}' \in \Lambda_r$ 
9    $H_s \leftarrow H(p_{\min,s})$ , computed as  $- \sum_{\boldsymbol{\lambda}' \in \Lambda_r} p_{\min,s}(\boldsymbol{\lambda}') \log p_{\min,s}(\boldsymbol{\lambda}')$ 
10   $u \leftarrow \text{const} - \frac{1}{S} \sum_{s=1}^S H_s$ 
```

Entropy Search: Variations

- The sample-based approximation is slow; for a faster approximation with expectation propagation see the original ES paper [Hennig et al. 2012]
- **Predictive Entropy Search** [Hernández-Lobato et al. 2014] is a frequently-used equivalent formulation that gives rise to more convenient approximations
- **Max-Value Entropy Search** [Wang and Jegelka 2017] is a recent variant that is cheaper to compute and has similar behavior
- Further reading and summary for ES: [Metzen 2016]

Questions to Answer for Yourself / Discuss with Friends

- Repetition. Describe the similarities and differences between KG and EI.
- Discussion. When is there an incentive for entropy search to sample at $\max(p_{min})$?

Outline

- 1 Introduction to Bayesian Optimization
- 2 Computationally Cheap Acquisition Functions
- 3 Computationally Expensive Acquisition Functions
- 4 Surrogate Models
- 5 High-Dimensional Bayesian Optimization
- 6 Extensions to Bayesian Optimization
- 7 The Alternative Optimization Approach of the Tree-Parzen Estimator (TPE)
- 8 Success Stories
- 9 Further Reading

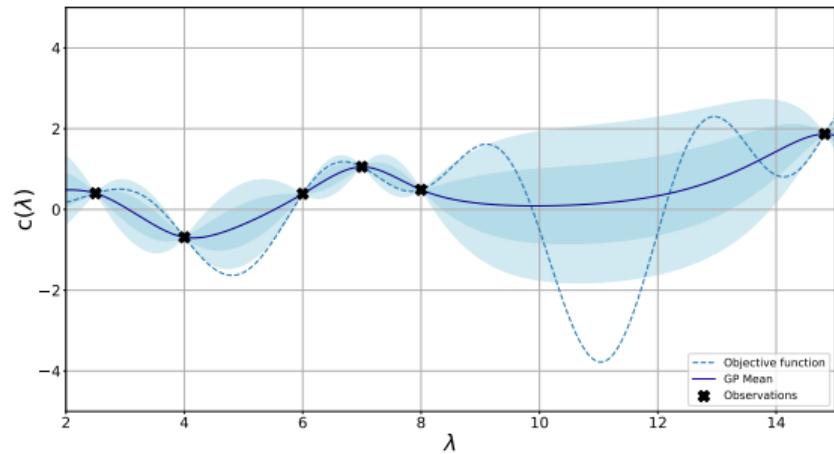
AutoML: Bayesian Optimization for Hyperparameter Optimization Surrogate Models

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Desiderata for Surrogate Models in Bayesian Optimization

In all cases

- Regression model with uncertainty estimates
- Accurate predictions



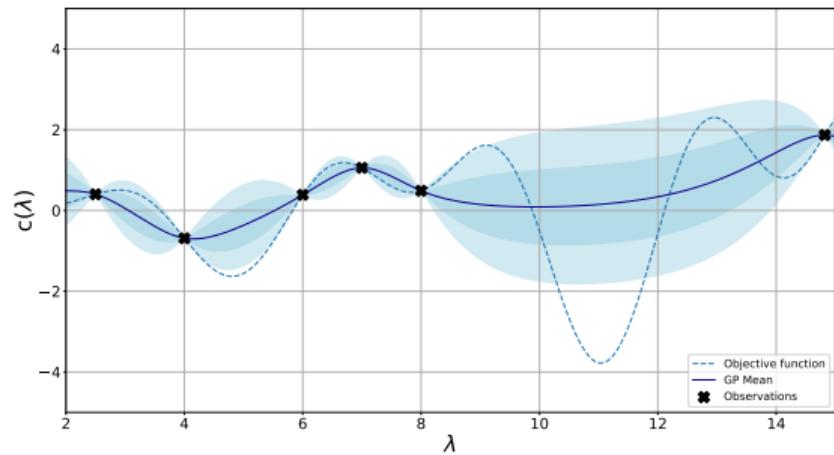
Desiderata for Surrogate Models in Bayesian Optimization

In all cases

- Regression model with uncertainty estimates
- Accurate predictions

Depending on the application

- Is cheap to train
- Scales well in the number of data points
- Scales well in the number of dimensions
- Can handle different types of inputs (categorical and continuous)



Overview of the Surrogate Models We'll Discuss

- Gaussian Processes
- Random Forests
- Bayesian Neural Networks

Gaussian Processes (GPs): Reminder of Pros and Cons

Advantages

- Smooth and reliable uncertainty estimates
- Strong sample efficiency
- We can encode expert knowledge about the design space in the kernel

Gaussian Processes (GPs): Reminder of Pros and Cons

Advantages

- Smooth and reliable uncertainty estimates
- Strong sample efficiency
- We can encode expert knowledge about the design space in the kernel

These advantages make GPs the most commonly-used model in Bayesian optimization

Gaussian Processes (GPs): Reminder of Pros and Cons

Advantages

- Smooth and reliable uncertainty estimates
- Strong sample efficiency
- We can encode expert knowledge about the design space in the kernel

These advantages make GPs the most commonly-used model in Bayesian optimization

Disadvantages

- Performance can be quite sensitive to the choice of kernel
- Cost scales cubically with the number of observations
- Weak performance for high dimensionality
- Not easily applicable in discrete or conditional spaces
- Sensitive to its own hyperparameters

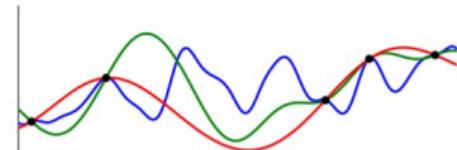
Gaussian Processes (GPs): Kernel Hyperparameters

- We could optimize GP hyperparameters (maximum likelihood, MLE, or maximum a posteriori, MAP)
- But [sampling](#) GP hyperparameters from the posterior distribution performs better; e.g., via [Markov-Chain Monte-Carlo \(MCMC\)](#)

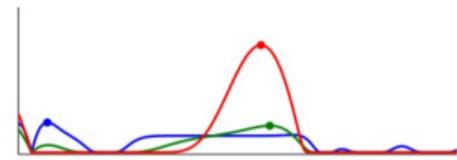
Gaussian Processes (GPs): Kernel Hyperparameters

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

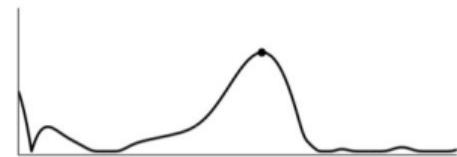
$$\bar{u}(\lambda) = \int u(\lambda, \hat{c}_\theta) p(\theta) d\theta$$



(a) Posterior samples under varying hyperparameters



(b) Expected improvement under varying hyperparameters



(c) Integrated expected improvement

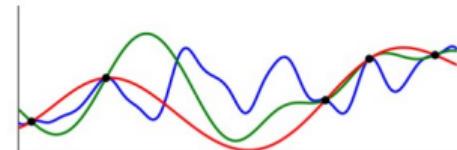
Image source: [Snoek et al. 2015]

Gaussian Processes (GPs): Kernel Hyperparameters

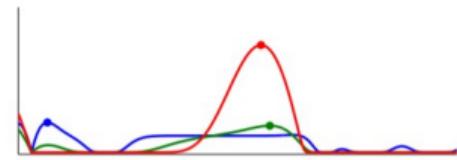
- We could optimize GP hyperparameters (maximum likelihood, MLE, or maximum a posteriori, MAP)
- But **sampling** GP hyperparameters from the posterior distribution performs better; e.g., via **Markov-Chain Monte-Carlo (MCMC)**
- Marginalize over GP hyperparameters θ and compute an **integrated acquisition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

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

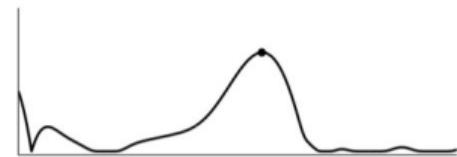
- Downside: computational expense
 - ▶ MCMC is computationally expensive
 - ▶ Acquisition function now has to be calculated for each sample



(a) Posterior samples under varying hyperparameters



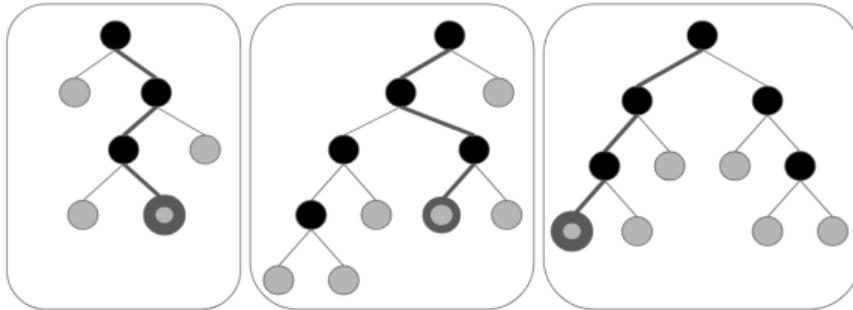
(b) Expected improvement under varying hyperparameters



(c) Integrated expected improvement

Image source: [Snoek et al. 2015]

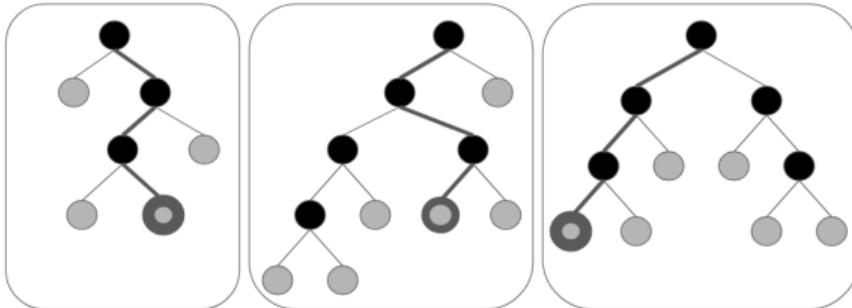
Random Forests (RFs): Reminder & How To Compute Uncertainties



RF Training

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

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



RF Training

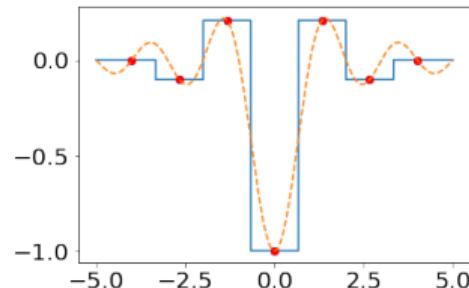
- Fit a set of randomized regression trees
- Randomization via bootstrapping & random selection of split variables / split points
- Each tree yields a possible explanation for the observations

RF Prediction

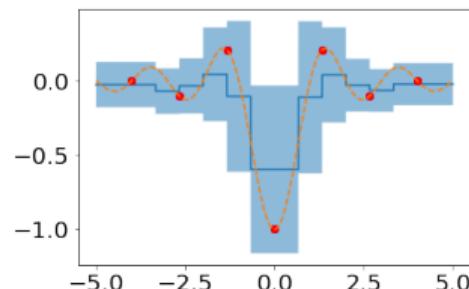
- Predict with each tree
- Aggregate predictions (e.g., average)
- Uncertainty estimate:
empirical variance across tree predictions

Random Forests (RFs): Impact of Basic Model Choices

(a) no bootstrapping,
no random splits

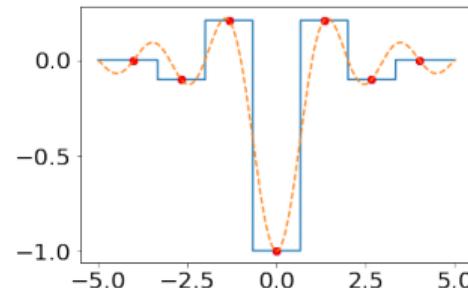


(b) with bootstrapping,
no random splits

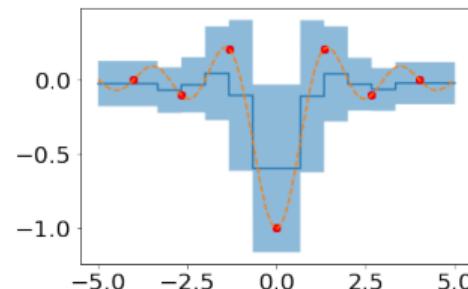


Random Forests (RFs): Impact of Basic Model Choices

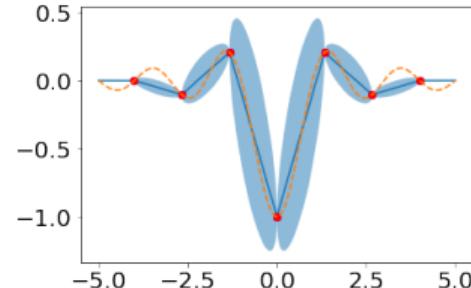
(a) no bootstrapping,
no random splits



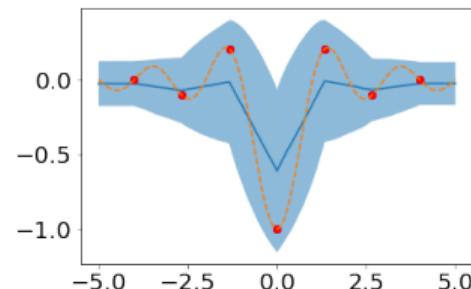
(b) with bootstrapping,
no random splits



(c) no bootstrapping,
with random splits



(d) with bootstrapping,
with random splits



Random Forests (RFs): Overview of Pros and Cons

Advantages

- Cheap to train
- Scales well with #observations n :
 - ▶ Fitting: $O(n \log n)$
 - ▶ Prediction: $O(\log n)$
- Scales well with #dimensions
- Training can be parallelized
- Can easily handle conditional, categorical, continuous and discrete spaces
- Quite robust against its own hyperparameters

Random Forests (RFs): Overview of Pros and Cons

Advantages

- Cheap to train
- Scales well with #observations n :
 - ▶ Fitting: $O(n \log n)$
 - ▶ Prediction: $O(\log n)$
- Scales well with #dimensions
- Training can be parallelized
- Can easily handle conditional, categorical, continuous and discrete spaces
- Quite robust against its own hyperparameters

Disadvantages

- Poor uncertainty estimates
- Poor extrapolation (constant)
- Priors cannot be incorporated easily

Random Forests (RFs): Overview of Pros and Cons

Advantages

- Cheap to train
- Scales well with #observations n :
 - ▶ Fitting: $O(n \log n)$
 - ▶ Prediction: $O(\log n)$
- Scales well with #dimensions
- Training can be parallelized
- Can easily handle conditional, categorical, continuous and discrete spaces
- Quite robust against its own hyperparameters

Disadvantages

- Poor uncertainty estimates
- Poor extrapolation (constant)
- Priors cannot be incorporated easily

These qualities make RFs a **robust** option for Bayesian optimization in **high dimensions**, for **categorical spaces**, or when function evaluations are quite fast

Bayesian Neural Networks: Overview

- Neural networks are more flexible & scalable than Gaussian processes
- But for use in Bayesian optimization, neural networks need to be made probabilistic

Bayesian Neural Networks: Overview

- Neural networks are more flexible & scalable than Gaussian processes
- But for use in Bayesian optimization, neural networks need to be made probabilistic
- Bayesian deep learning aims to deal with all sources of uncertainty

Bayesian Neural Networks: Overview

- Neural networks are more flexible & scalable than Gaussian processes
- But for use in Bayesian optimization, neural networks need to be made probabilistic
- Bayesian deep learning aims to deal with all sources of uncertainty
 - ▶ E.g., we don't have a single weight vector anymore, but a distribution over weights

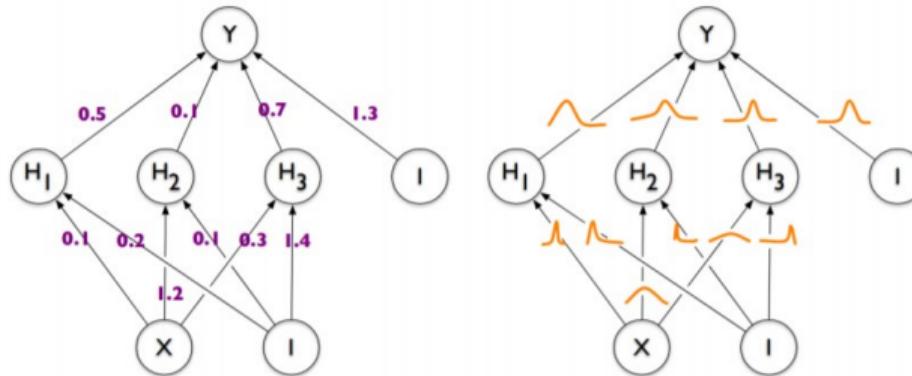


Image source: [Blundell et al. 2015]

Simplest Way of Incorporating Uncertainty in Neural Networks: DNGO

- Fit a standard regression neural network to the data (with a linear output layer)
- Use the representation in the last hidden layer as **basis functions** $\phi(x)$ of the input x
- Use Bayesian linear regression with these basis functions

Simplest Way of Incorporating Uncertainty in Neural Networks: DNGO

- Fit a standard regression neural network to the data (with a linear output layer)
- Use the representation in the last hidden layer as **basis functions** $\phi(x)$ of the input x
- Use **Bayesian linear regression with these basis functions**
 - ▶ The last layer is linear in its parameters θ
 - ▶ Therefore, the Bayesian linear regression formulas work directly
 - ▶ Feasible in closed form, in time $O(Nd^3)$, where N is the number of data points and d is the number of hidden units in the last layer
- Not fully Bayesian yet, but already allows scalable Bayesian optimization [Snoek et al. 2015]

Bayesian Optimization with BNNs: Overview of Existing Approaches

- [Snoek et al. 2015] Scalable Bayesian Optimization Using Deep Neural Networks (DNGO)
- [Springenberg et al. 2016] Bayesian Optimization with Robust Bayesian Neural Networks
- [Hernández-Lobato et al. 2017] Parallel and Distributed Thompson Sampling for Large-scale Accelerated Exploration of Chemical Space

Bayesian Optimization with BNNs: Overview of Existing Approaches

- [Snoek et al. 2015] Scalable Bayesian Optimization Using Deep Neural Networks (DNGO)
- [Springenberg et al. 2016] Bayesian Optimization with Robust Bayesian Neural Networks
- [Hernández-Lobato et al. 2017] Parallel and Distributed Thompson Sampling for Large-scale Accelerated Exploration of Chemical Space
- [Schilling et al. 2015] Hyperparameter Optimization with Factorized Multilayer Perceptrons
- [Perrone et al. 2018] Scalable Hyperparameter Transfer Learning

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. 2017]
- Dropout [Gal et al. 2016] Representing Model Uncertainty in Deep Learning
- 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 Research Project.** All of the surrogate models we saw have pros and cons. It would be interesting to select the best model (and its hyperparameters) dependent on the data at hand.

Outline

- 1 Introduction to Bayesian Optimization
- 2 Computationally Cheap Acquisition Functions
- 3 Computationally Expensive Acquisition Functions
- 4 Surrogate Models
- 5 High-Dimensional Bayesian Optimization
- 6 Extensions to Bayesian Optimization
- 7 The Alternative Optimization Approach of the Tree-Parzen Estimator (TPE)
- 8 Success Stories
- 9 Further Reading

AutoML: Bayesian Optimization for Hyperparameter Optimization

High-Dimensional Bayesian Optimization

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

High-Dimensional Bayesian Optimization: Motivation

- Issue: Standard BO works best on problems of moderate dimensions $d \leq 20$
 - ▶ Standard Gaussian processes do not tend to fit well in high dimensions
 - ▶ Maximizing the acquisition function is also computationally challenging

High-Dimensional Bayesian Optimization: Motivation

- Issue: Standard BO works best on problems of moderate dimensions $d \leq 20$
 - ▶ Standard Gaussian processes do not tend to fit well in high dimensions
 - ▶ Maximizing the acquisition function is also computationally challenging
- Possible solutions we will discuss:
 - ▶ Different models, in particular random forests [Hutter et al. 2011]
 - ▶ Embedding into a low-dimensional space (REMBO) [Wang et al. 2016]
 - ▶ Additive models [Kandasamy et al. 2015]

Low Effective Dimensionality

- Many optimization problems in practice have low effective dimensionality
 - ▶ E.g., HPO for deep neural networks [Bergstra et al. 2012]
 - ▶ E.g., algorithm configuration for combinatorial optimization solvers [Hutter et al. 2014]

Low Effective Dimensionality

- Many optimization problems in practice have **low effective dimensionality**
 - ▶ E.g., HPO for deep neural networks [Bergstra et al. 2012]
 - ▶ E.g., algorithm configuration for combinatorial optimization solvers [Hutter et al. 2014]
- Idea: Exploit low effective dimensionality to cover a lower-dimensional space well

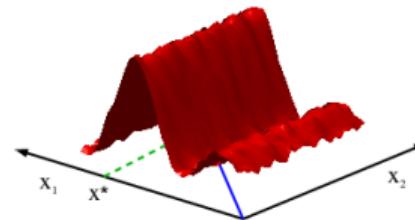
Using Random Forests for High Dimensions / Low Effective Dimensionality

- Random forests are **automatic feature detectors**
 - ▶ They automatically select the important (axis-aligned) inputs
- Random forests have indeed been used effectively on spaces of more than 700 hyperparameters
 - ▶ In terms of computational efficiency, they do not pose a bottleneck
 - ▶ In terms of statistical efficiency, they scale more gracefully to high dimensions than GPs

Random Embeddings for Exploiting Low Effective Dimensionality: Overview

Given a $D = 2$ dimensional black-box function $c(x_1, x_2)$:

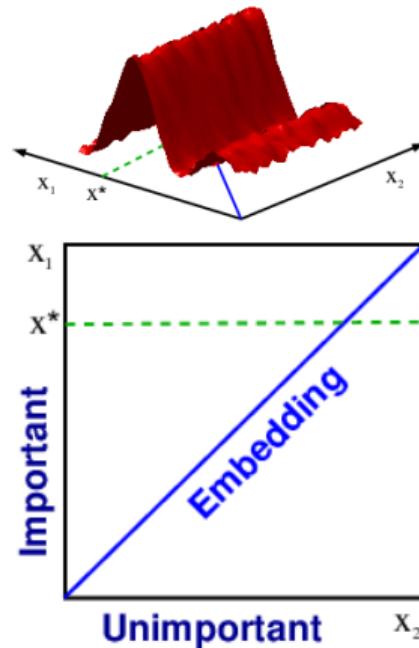
- Assume we know c has only $d = 1$ important dimensions, but we don't know which one it is.



Random Embeddings for Exploiting Low Effective Dimensionality: Overview

Given a $D = 2$ dimensional black-box function $c(x_1, x_2)$:

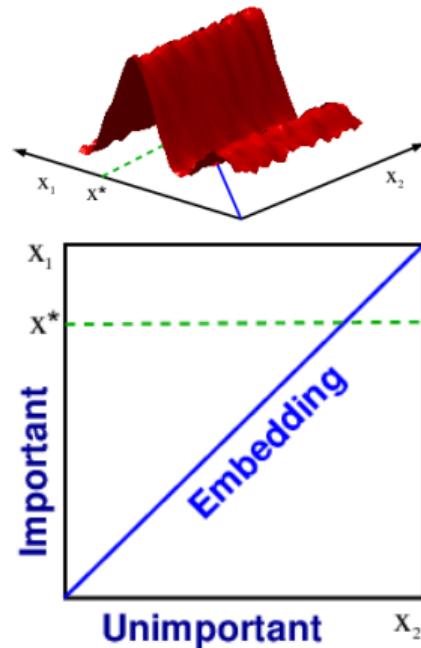
- Assume we know c has only $d = 1$ important dimensions, but we don't know which one it is.
- Subspace $x_1 = x_2$ is guaranteed to include the optimum.



Random Embeddings for Exploiting Low Effective Dimensionality: Overview

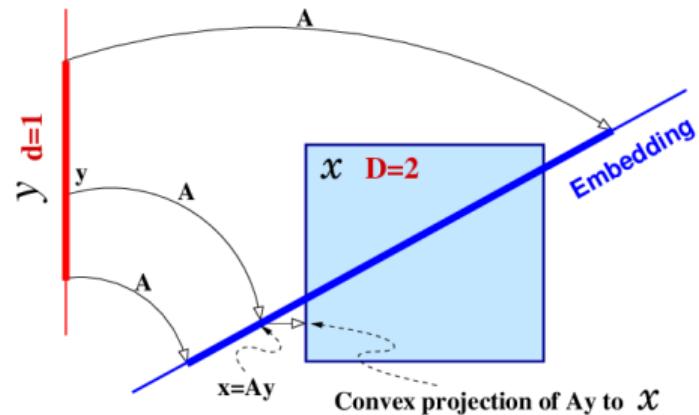
Given a $D = 2$ dimensional black-box function $c(x_1, x_2)$:

- Assume we know c has only $d = 1$ important dimensions, but we don't know which one it is.
- Subspace $x_1 = x_2$ is guaranteed to include the optimum.
- This idea applies to any d -dimensional linear subspace; allows scaling to arbitrary D (e.g., $D = 1$ billion)



Random Embedding Bayesian Optimization (REMBO)

- Generate a random matrix $A \in \mathbb{R}^{D \times d}$
- Use BO to optimize $g(\lambda) = c(Ay)$ instead of high dimensional $c(\lambda)$

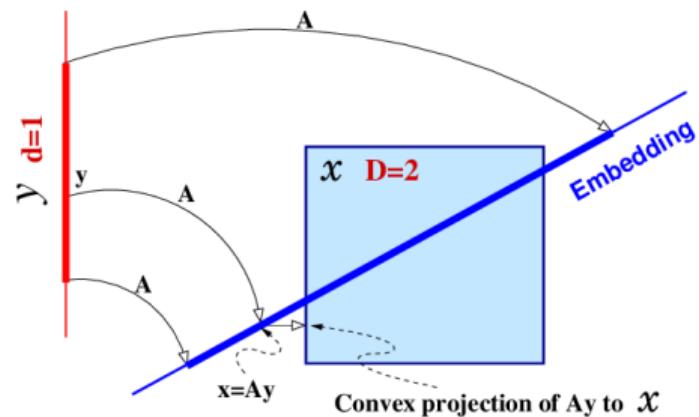


Random Embedding Bayesian Optimization (REMBO)

- Generate a random matrix $A \in \mathbb{R}^{D \times d}$
- Use BO to optimize $g(\lambda) = c(Ay)$ instead of high dimensional $c(\lambda)$

Theorem

If the effective dimensionality of c is at most d , then with probability 1, for any $\lambda \in \mathbb{R}^D$, there exists a $y \in \mathbb{R}^d$ such that $c(\lambda) = c(Ay)$.



High Dimensional Bayesian Optimization: REMBO Pseudocode

REMBO: Bayesian Optimization with Random Embedding

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

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

- 1 Generate a random matrix $A \in \mathbb{R}^{D \times d}$
 - 2 $\mathcal{D}^{(0)} \leftarrow \emptyset$
 - 3 **for** $t = 1$ **to** T **do**
 - 4 $\hat{c}^{(t)} \leftarrow$ fit predictive model on $\mathcal{D}^{(t-1)}$
 - 5 $\mathbf{y} \leftarrow \mathbf{y} \in \arg \max_{\mathbf{y} \in \mathcal{Y}} u(\mathbf{y} | \mathcal{D}^{(t-1)}, \hat{c}^{(t)})$
 - 6 Query $c(A\mathbf{y})$;
 - 7 $\mathcal{D}^{(t)} \leftarrow \mathcal{D}^{(t-1)} \cup \{\langle A\mathbf{y}, c(A\mathbf{y}) \rangle\}$
-

High Dimensional Bayesian Optimization

Random Embedding Bayesian Optimization - Summary

Advantages

- Exploits low effective dimensionality
- Allows scaling to arbitrarily high extrinsic dimensions
- Applies to both continuous and categorical variables
- Trivial modification of BO algorithm
- Coordinate independent (invariant under rotations)

High Dimensional Bayesian Optimization

Random Embedding Bayesian Optimization - Summary

Advantages

- Exploits low effective dimensionality
- Allows scaling to arbitrarily high extrinsic dimensions
- Applies to both continuous and categorical variables
- Trivial modification of BO algorithm
- Coordinate independent (invariant under rotations)

Disadvantages

- Sensitive to the definition of the bounded low dimensional constrained space \mathcal{Y}
- Assumes truly unimportant dimensions

High Dimensional Bayesian Optimization via Additive Models

- Recall:
 - ▶ Standard GPs do not tend to fit well in high dimensions
 - ▶ Maximizing the acquisition function is also computationally challenging

High Dimensional Bayesian Optimization via Additive Models

- Recall:
 - ▶ Standard GPs do not tend to fit well in high dimensions
 - ▶ Maximizing the acquisition function is also computationally challenging
- Idea:
 - ▶ Assume additive structure of the objective function [Kandasamy et al. 2015]:

$$f(\boldsymbol{\lambda}) = f^{(1)}(\boldsymbol{\lambda}^{(1)}) + f^{(2)}(\boldsymbol{\lambda}^{(2)}) + \dots + f^{(M)}(\boldsymbol{\lambda}^{(M)})$$

- ▶ Model each $f^{(i)}$ by an individual GP

High Dimensional Bayesian Optimization via Additive Models

- Recall:
 - ▶ Standard GPs do not tend to fit well in high dimensions
 - ▶ Maximizing the acquisition function is also computationally challenging
- Idea:
 - ▶ Assume additive structure of the objective function [Kandasamy et al. 2015]:

$$f(\boldsymbol{\lambda}) = f^{(1)}(\boldsymbol{\lambda}^{(1)}) + f^{(2)}(\boldsymbol{\lambda}^{(2)}) + \dots + f^{(M)}(\boldsymbol{\lambda}^{(M)})$$

- ▶ Model each $f^{(i)}$ by an individual GP
- ▶ If the decomposition does not overlap:
can maximize acquisition function separately for each of the $f^{(i)}$

High Dimensional Bayesian Optimization via Additive Models

- Recall:
 - ▶ Standard GPs do not tend to fit well in high dimensions
 - ▶ Maximizing the acquisition function is also computationally challenging
- Idea:
 - ▶ Assume additive structure of the objective function [Kandasamy et al. 2015]:
$$f(\boldsymbol{\lambda}) = f^{(1)}(\boldsymbol{\lambda}^{(1)}) + f^{(2)}(\boldsymbol{\lambda}^{(2)}) + \dots + f^{(M)}(\boldsymbol{\lambda}^{(M)})$$
 - ▶ Model each $f^{(i)}$ by an individual GP
 - ▶ If the decomposition does not overlap:
can maximize acquisition function separately for each of the $f^{(i)}$
 - ▶ Best results for known decomposition, but also possible to learn decomposition from the data

High Dimensional Bayesian Optimization via Additive Models

Advantages

- Exploits low effective dimensionality
- Scales GPs to high-dimensional parameter spaces
- Regret is linearly dependent on the dimension D when c is additive

High Dimensional Bayesian Optimization via Additive Models

Advantages

- Exploits low effective dimensionality
- Scales GPs to high-dimensional parameter spaces
- Regret is linearly dependent on the dimension D when c is additive

Disadvantages

- Sensitive to the number of additive components
- Restricted to an axis-aligned representation
- Relies on assumption of additivity

Questions to Answer for Yourself / Discuss with Friends

- Repetition. What is the main assumption behind REMBO?
- Repetition. What is the main assumption behind additive modelling?
- Discussion. Are these assumptions likely satisfied for tuning deep neural networks?
- Discussion. How do random forests help deal with high dimensions and low effective dimensionality? Can they also model additive structure?

Outline

- 1 Introduction to Bayesian Optimization
- 2 Computationally Cheap Acquisition Functions
- 3 Computationally Expensive Acquisition Functions
- 4 Surrogate Models
- 5 High-Dimensional Bayesian Optimization
- 6 Extensions to Bayesian Optimization
- 7 The Alternative Optimization Approach of the Tree-Parzen Estimator (TPE)
- 8 Success Stories
- 9 Further Reading

AutoML: Bayesian Optimization for Hyperparameter Optimization

Extensions to Bayesian Optimization

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Some More Extensions We Will Discuss

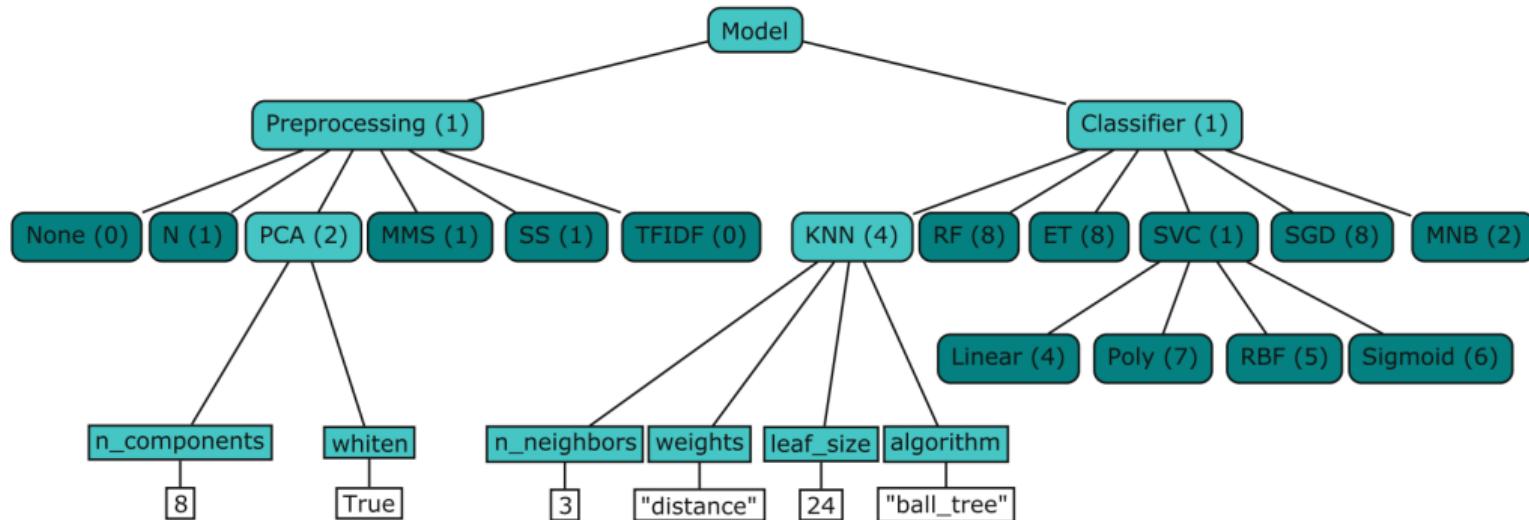
Standard Bayesian optimization problems

- Continuous, smooth functions
- Sequential optimization
- Noise-free evaluations

Extensions

- Structured search spaces: categorical & conditional hyperparameters
- Parallel evaluations
- Noisy evaluations
- Optimization with constraints

Structured Search Spaces: Categorical & Conditional Hyperparameters

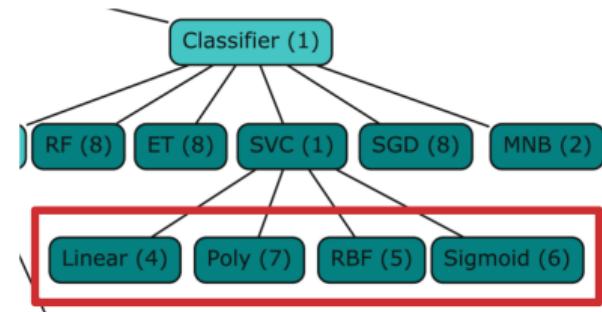


Example of a structured search space (Source: Figure 5.1 of the [AutoML book])

Structured Search Spaces: Categorical Hyperparameters

Properties of categorical hyperparameters:

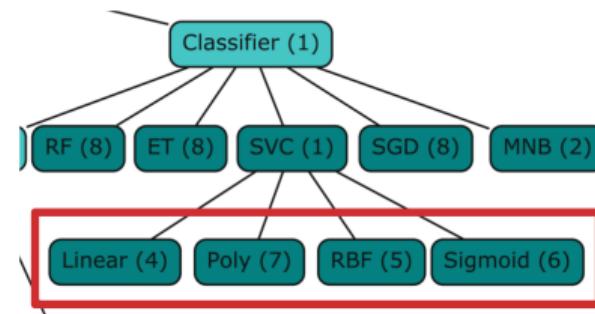
- Finite, discrete set of values
- No natural order between values
- Potentially different distances between values



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]:

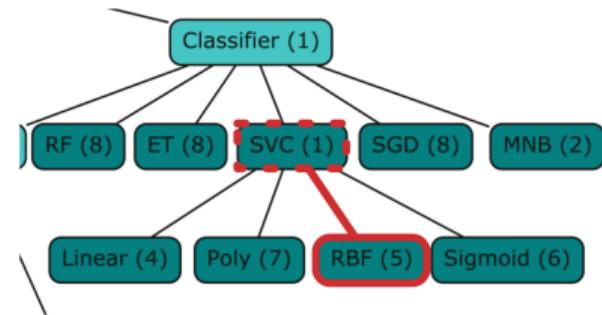
$$\kappa_{\theta}(\boldsymbol{\lambda}_i, \boldsymbol{\lambda}_j) = \exp \sum_{l=1}^d (-\theta \cdot \delta(\lambda_{i,l} \neq \lambda_{j,l}))$$

- Neural networks can learn entity embeddings for categorical inputs [Guc et al. 2016]

Structured Search Spaces: Conditional Hyperparameters

Conditional hyperparameters:

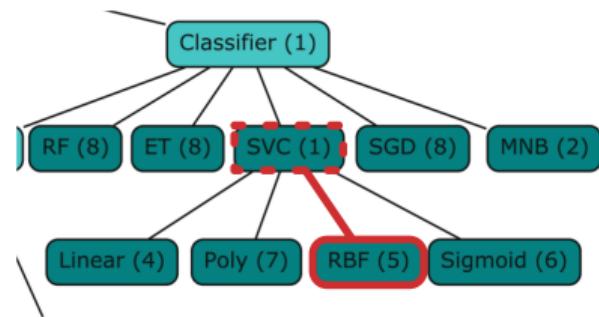
- Are **only relevant if** certain other hyperparameters take on certain values
- Should be ignored by the model **if not active**



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**



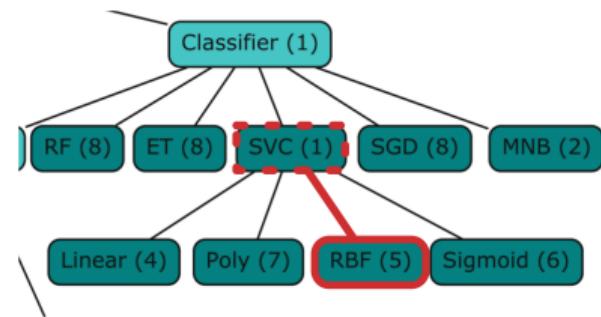
Modelling conditional hyperparameters:

- 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 et al. 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 et al. 2013] [Lévesque et al. 2017] [Jenatton et al. 2017]

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

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. 2007], [Wu et al. 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. 2007], [Wu et al. 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

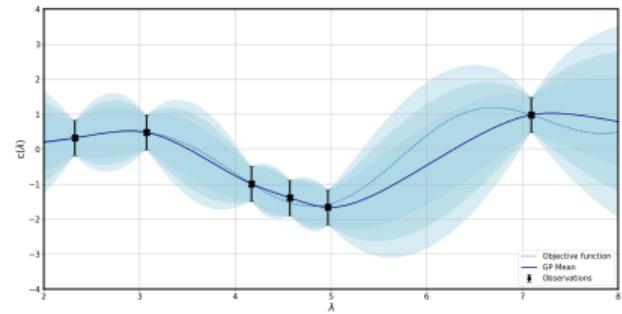
- In practice, typically, not all function evaluations take the same amount of time
 - ▶ Thus, we need to select **some new points** while we're still waiting for **pending evaluations** at other points
- Simple solution: **hallucinate observations for pending evaluations**, and use otherwise standard 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

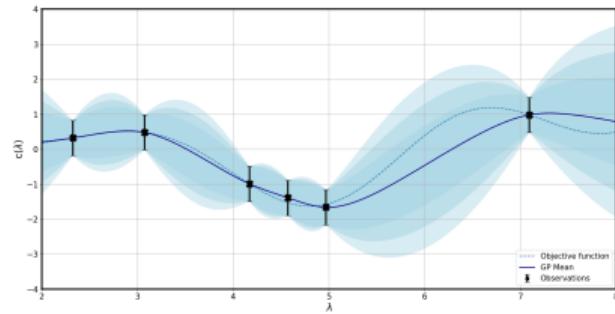
Noisy Evaluations

- The probabilistic model natively supports Gaussian noise
 - ▶ A good hyperprior for the noise variance might be needed to robustly determine the right size of the noise
 - ▶ Noise that is not Gaussian (and not Student t [Santner et al, 2014]) would require approximations



Noisy Evaluations

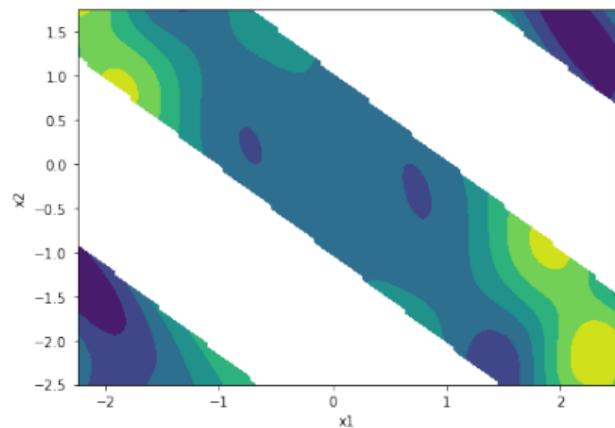
- The probabilistic model natively supports Gaussian noise
 - ▶ A good hyperprior for the noise variance might be needed to robustly determine the right size of the noise
 - ▶ Noise that is not Gaussian (and not Student t [Santner et al, 2014]) would require approximations
- The acquisition function might need adaptation
 - ▶ LBC, TS, ES, and KG, are not affected
 - ▶ PI and EI are based on the cost c_{inc} of the incumbent; it is unclear how to compute this
 - ★ Uncertainty about which point is the current incumbent
 - ★ Uncertainty about the costs $c(\lambda_i)$.
 - ★ **Noisy Expected Improvement** [Letham et al. 2019] extends regular EI by integrating over the predictive posterior of the model using Monte Carlo



Bayesian Optimization with Constraints

Three 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 and Gramacy 2010]
- ③ Unknown constraints: there's an additional, but unknown constraint function, for example the memory used, which can be observed and modeled



Hidden constraints. Image source:
[GPFlowOpt Tutorial, Apache 2 License]

Most general solution: *Expected Constrained Improvement* [Lee and Gramacy 2010]:

$$ECI(\lambda) = EI(\lambda)h(\lambda), \quad (1)$$

where $h(\lambda)$ is the probability that λ is a valid configuration.

Further literature in [Frazier 2018] and [Feurer and Hutter 2019].

Further extensions

Even more extensions

Bayesian optimization has been extended to numerous scenarios:

- Multi-task, Multi-fidelity and Meta-learning → separate lecture
- Multi-objective Bayesian optimization → separate lecture
- Bayesian optimization with safety guarantees [Sui et al. 2015]
- Directly optimizing for ensemble performance [Lévesque et al. 2016]
- Combination with local search methods [Taddy et al. 2009] [Eriksson et al. 2019]
- Optimization of arbitrary spaces that can be described by a kernel (e.g., neural network architectures [Kandasamy et al. 2018] or molecules [Griffiths et al. 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?

Outline

- 1 Introduction to Bayesian Optimization
- 2 Computationally Cheap Acquisition Functions
- 3 Computationally Expensive Acquisition Functions
- 4 Surrogate Models
- 5 High-Dimensional Bayesian Optimization
- 6 Extensions to Bayesian Optimization
- 7 The Alternative Optimization Approach of the Tree-Parzen Estimator (TPE)
- 8 Success Stories
- 9 Further Reading

AutoML: Bayesian Optimization for Hyperparameter Optimization

The Alternative Optimization Approach of the Tree-Parzen Estimator (TPE)

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Overview of TPE [Bergstra et al. 2011]

- Standard Bayesian optimization models the probability $p(y | \lambda)$ of observations y given configurations λ
- Instead, TPE fits kernel density estimators (KDEs) $l(\lambda | y \leq \gamma)$ and $g(\lambda | y \leq \gamma)$
 - ▶ These KDEs are for “good configurations” (leading to objective function values below a threshold γ) and “bad configurations”
 - ▶ By default, γ is set to the 15% quantile of the observations

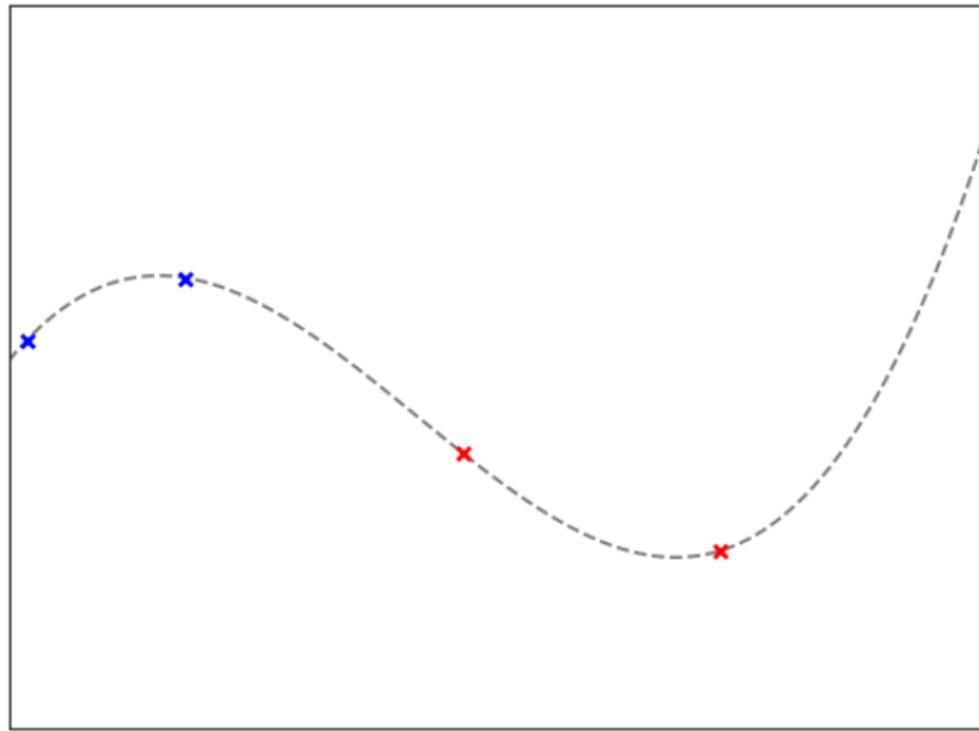
Overview of TPE [Bergstra et al. 2011]

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

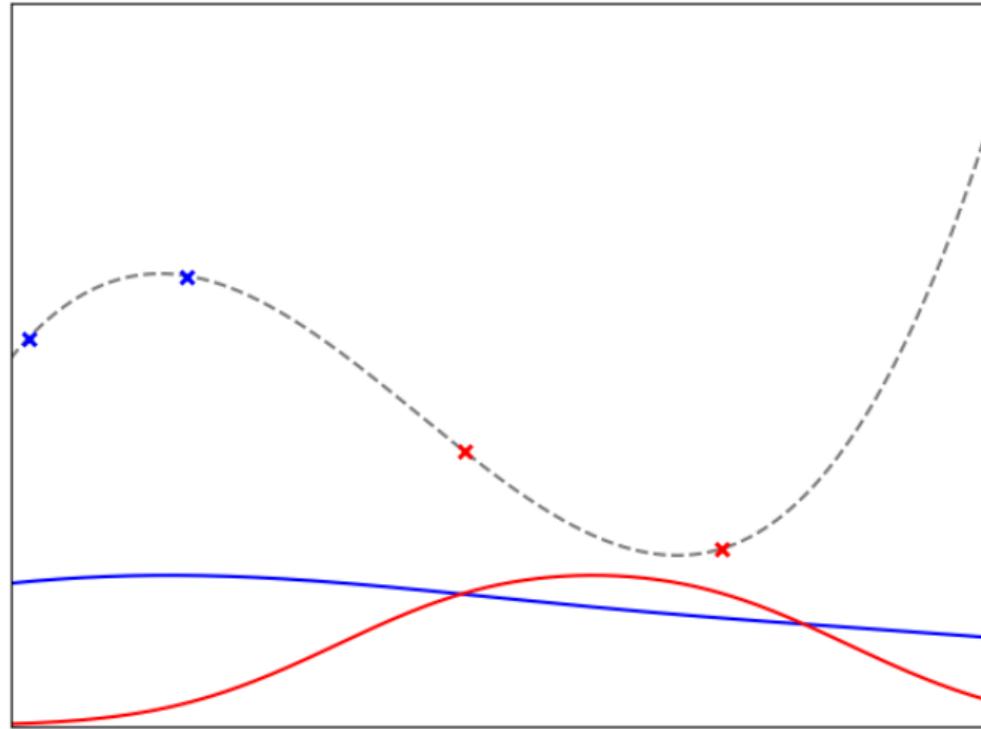
Overview of TPE [Bergstra et al. 2011]

- Standard Bayesian optimization models the probability $p(y | \lambda)$ of observations y given configurations λ
- Instead, TPE fits kernel density estimators (KDEs) $l(\lambda | y \leq \gamma)$ and $g(\lambda | y \leq \gamma)$
 - ▶ These KDEs are for “good configurations” (leading to objective function values below a threshold γ) and “bad configurations”
 - ▶ By default, γ is set to the 15% quantile of the observations
- Optimizing $l(\lambda)/g(\lambda)$ is equivalent to optimizing standard expected improvement in Bayesian optimization [Bergstra et al. 2011]
- Why is the technique called TPE?
 - ▶ The used KDEs are Parzen estimators
 - ▶ TPE can handle tree-structured search spaces

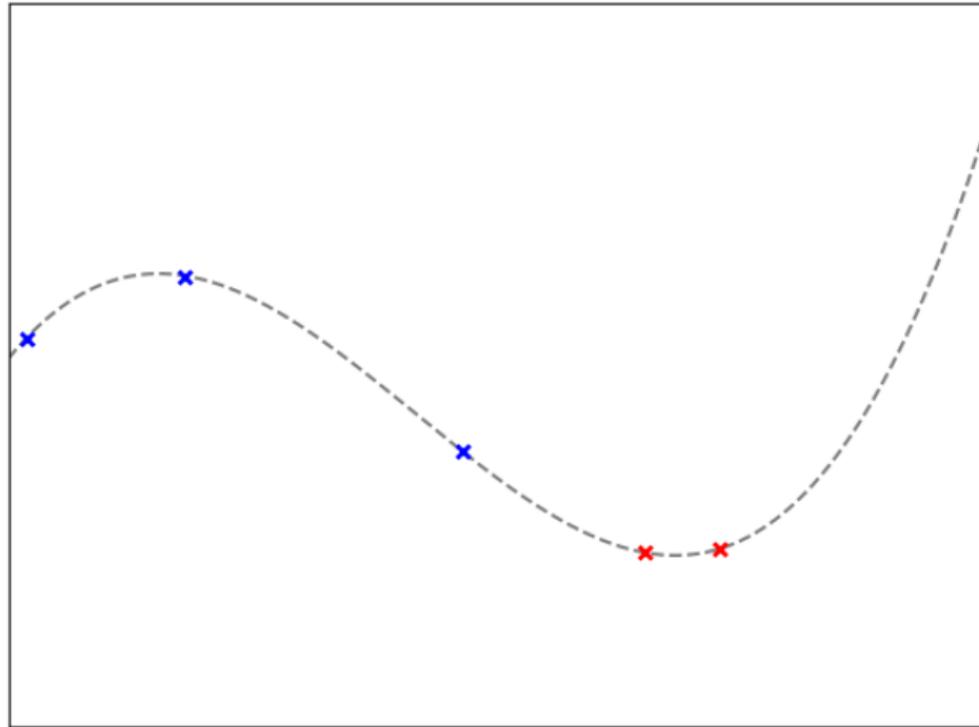
TPE Example



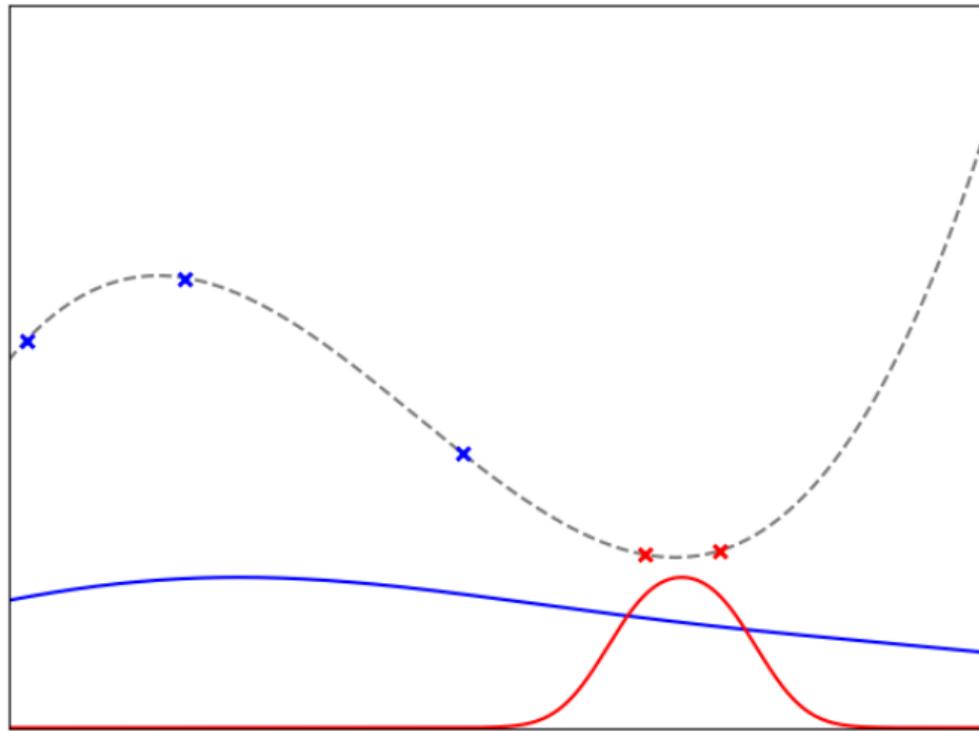
TPE Example



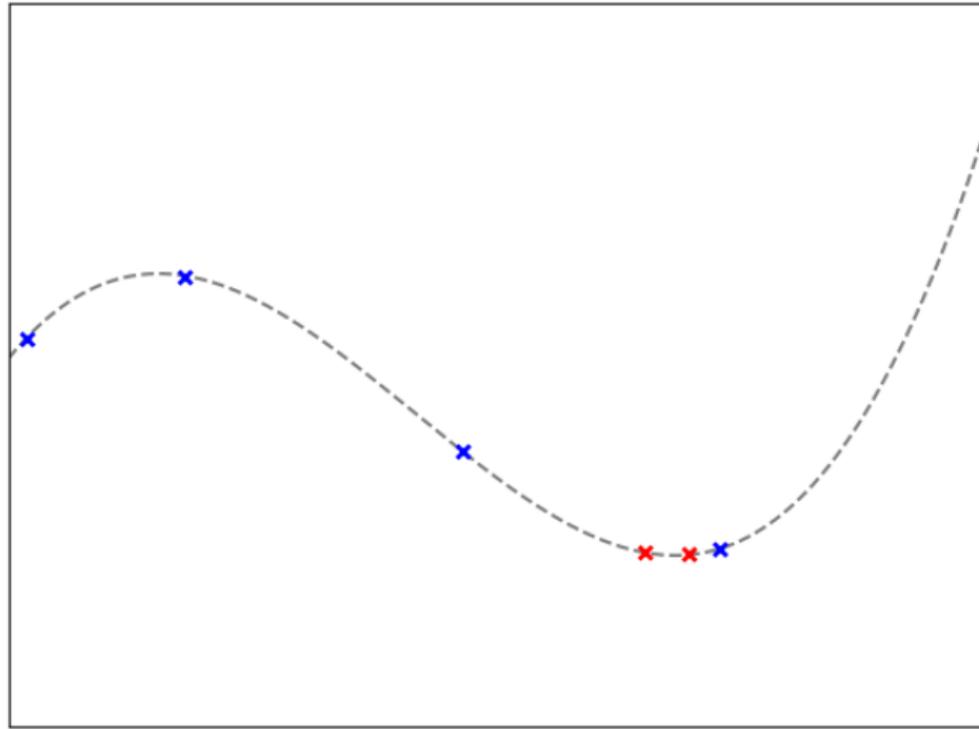
TPE Example



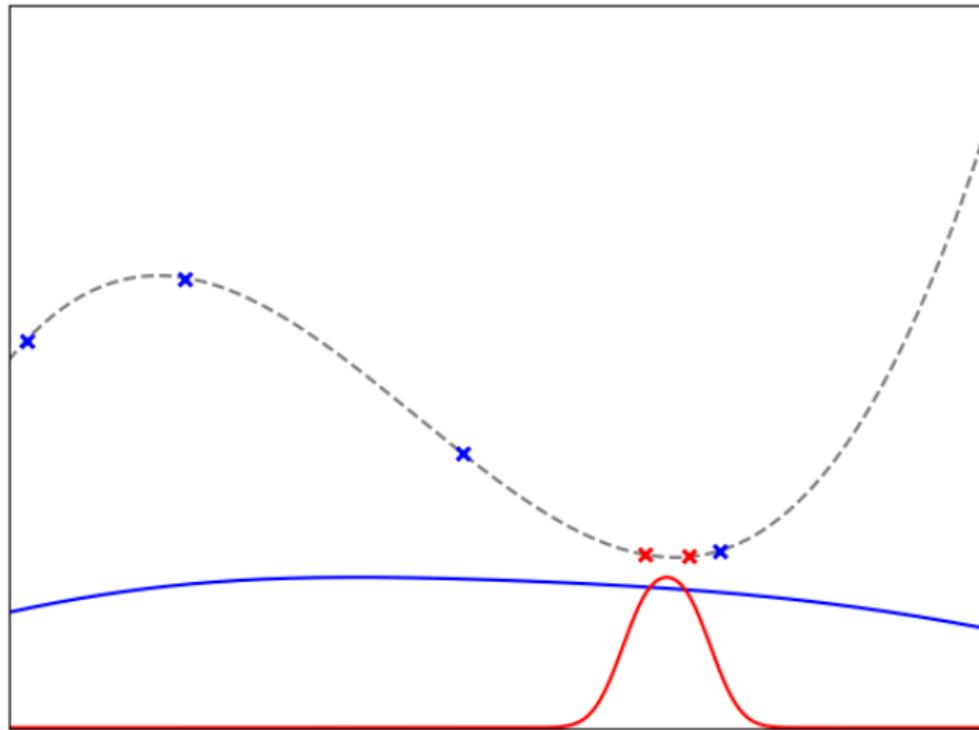
TPE Example



TPE Example



TPE Example



TPE Pseudocode

TPE loop

Require: Search space Λ , cost function c , percentile γ , maximal number of function evaluations T

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

```
1  $\mathcal{D}^{(0)} \leftarrow \emptyset$ 
2 for  $t = 1$  to  $T$  do
3    $\mathcal{D}_{\text{good}}, \mathcal{D}_{\text{bad}} \leftarrow$  split  $\mathcal{D}^{(t-1)}$  according to quantile  $\gamma$ 
4    $l(\lambda), g(\lambda) \leftarrow$  fit KDE on  $\mathcal{D}_{\text{good}}, \mathcal{D}_{\text{bad}}$  respectively
5    $\Lambda_{\text{cand}} \leftarrow$  draw samples from  $l$ ;
6   Select next query point:  $\lambda^{(t)} \in \arg \max_{\lambda \in \Lambda_{\text{cand}}} l(\lambda)/g(\lambda)$ 
7   Query  $c(\lambda^{(t)})$ 
8    $\mathcal{D}^{(t)} \leftarrow \mathcal{D}^{(t-1)} \cup \{\langle \lambda^{(t)}, c(\lambda^{(t)}) \rangle\}$ 
9 end
```

Further Details

Remarks:

- TPE models $p(\boldsymbol{\lambda}|c(\boldsymbol{\lambda}))$
 - ▶ we can multiply it with a prior to add expert knowledge

Further Details

Remarks:

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

Further Details

Remarks:

- TPE models $p(\boldsymbol{\lambda}|c(\boldsymbol{\lambda}))$
 - ▶ we can multiply it with a prior to add expert knowledge
- Performance of TPE depends on:
 - ▶ setting of γ to trade-off exploration and exploitation
 - ▶ bandwidth of the KDEs
- A successful tool implementing TPE is Hyperopt [Bergstra et al.]

Summary

Advantages

- Computationally efficient: $O(Nd)$
- Parallelizable
- Robust
- Can handle complex search spaces with priors

Summary

Advantages

- Computationally efficient: $O(Nd)$
- Parallelizable
- Robust
- Can handle complex search spaces with priors

Disadvantages

- Less sample-efficient than GPs

Questions to Answer for Yourself / Discuss with Friends

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

Outline

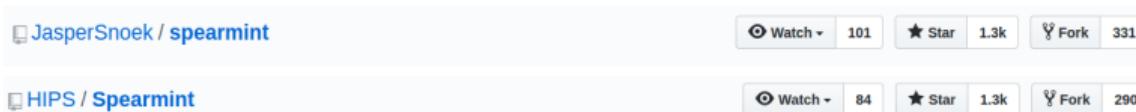
- 1 Introduction to Bayesian Optimization
- 2 Computationally Cheap Acquisition Functions
- 3 Computationally Expensive Acquisition Functions
- 4 Surrogate Models
- 5 High-Dimensional Bayesian Optimization
- 6 Extensions to Bayesian Optimization
- 7 The Alternative Optimization Approach of the Tree-Parzen Estimator (TPE)
- 8 Success Stories**
- 9 Further Reading

AutoML: Bayesian Optimization for Hyperparameter Optimization Success Stories

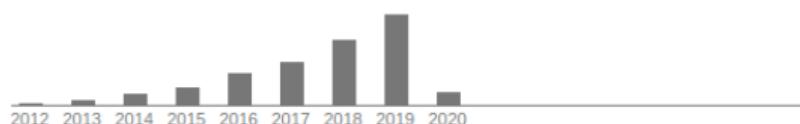
Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Spearmint [Snoek et al. 2012]

- First successful open source Bayesian optimization implementation
- Implements standard Bayesian optimization with MCMC integration of the acquisition function, asynchronous parallelism, input warping and constraints
- Startup based on Spearmint got acquired by Twitter in 2015
- Still heavily used and cited and available at <https://github.com/HIPS/spearmint>:



Cited by 3073



[Practical bayesian optimization of machine learning algorithms](#)

J Snoek, H Larochelle, RP Adams - Advances in neural information processing systems, 2012

[Cited by 3073](#) [Related articles](#) [All 26 versions](#)

Hyperopt [Bergstra et al. 2011, Bergstra et al., 2013, Bergstra et al., 2013, Bergstra et al., 2015]

- Hyperopt is another successful open source Bayesian optimization package
- Implements the TPE algorithm and supports asynchronous parallel evaluations
- Maintained since 2013
- Available at <https://github.com/hyperopt/hyperopt>

 [hyperopt / hyperopt](https://github.com/hyperopt/hyperopt)

 Watch ▾ 118

 Star 4.4k

 Fork 747

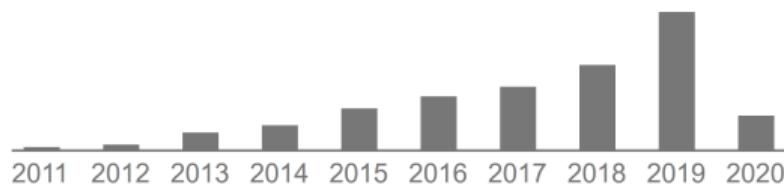
- Standard BO tool based on random forests (RFs), reflecting the strengths of RFs in terms of **scalability & flexibility**:
 - ▶ High dimensionality (low effective dimensionality)
 - ▶ Computational efficiency (\rightarrow low overhead)
 - ▶ Supports continuous/categorical/conditional parameters
 - ▶ Supports non-standard noise (non-Gaussian, heteroscedastic)
 - ▶ Usability off the shelf (robustness towards model's own hyperparameters)

- Standard BO tool based on random forests (RFs), reflecting the strengths of RFs in terms of **scalability & flexibility**:
 - ▶ High dimensionality (low effective dimensionality)
 - ▶ Computational efficiency (\rightarrow low overhead)
 - ▶ Supports continuous/categorical/conditional parameters
 - ▶ Supports non-standard noise (non-Gaussian, heteroscedastic)
 - ▶ Usability off the shelf (robustness towards model's own hyperparameters)
- SMAC also handles a more general problem: $\arg \min_{\lambda \in \Lambda} \sum_{i=1}^N c(\lambda, i)$

SMAC [Hutter et al. 2011]

- Standard BO tool based on random forests (RFs), reflecting the strengths of RFs in terms of **scalability & flexibility**:
 - ▶ High dimensionality (low effective dimensionality)
 - ▶ Computational efficiency (\rightarrow low overhead)
 - ▶ Supports continuous/categorical/conditional parameters
 - ▶ Supports non-standard noise (non-Gaussian, heteroscedastic)
 - ▶ Usability off the shelf (robustness towards model's own hyperparameters)
- SMAC also handles a more general problem: $\arg \min_{\lambda \in \Lambda} \sum_{i=1}^N c(\lambda, i)$
- Maintained since 2011, now available in version 3: <https://github.com/automl/SMAC3>

Cited by 1318



Sequential model-based optimization for general algorithm configuration
F Hutter, HH Hoos, K Leyton-Brown - International conference on
learning and intelligent ..., 2011

Tuning AlphaGo [Chen et al. 2018]

- “During the development of AlphaGo, its many hyperparameters were tuned with Bayesian optimization multiple times.”
- “This automatic tuning process resulted in substantial improvements in playing strength. For example, prior to the match with Lee Sedol, we tuned the latest AlphaGo agent and this improved its win-rate from 50% to 66.5% in self-play games. This tuned version was deployed in the final match.”
- Of course, since we tuned AlphaGo many times during its development cycle, the compounded contribution was even higher than this percentage.

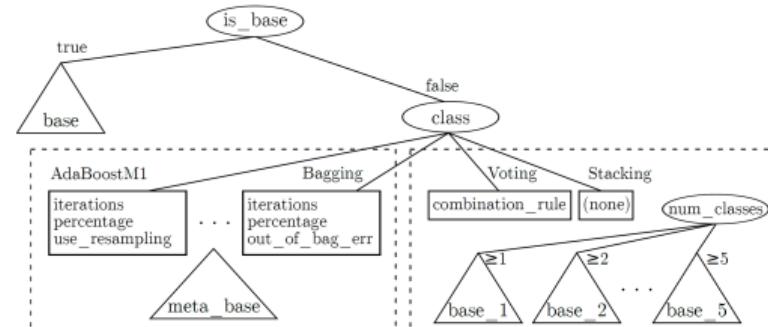
Company usage

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

- First general AutoML system, carrying out **Combined Algorithm Selection and Hyperparameter optimization (CASH)**, jointly optimizing
 - ▶ Choice of algorithm (out of 26 classifiers)
 - ▶ The algorithm's hyperparameters (up to 10)
 - ▶ Choice of preprocessing method and its hyperparameters
 - ▶ Choice of ensemble & meta methods

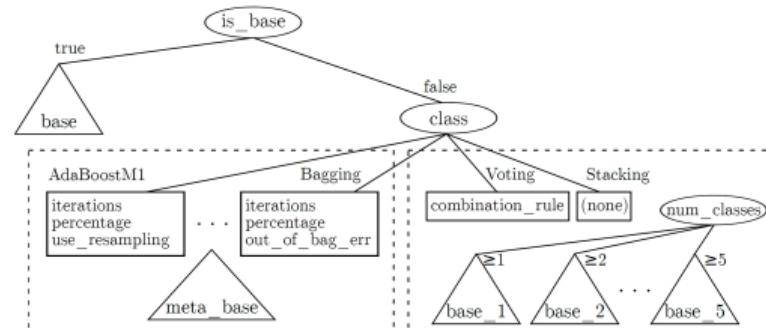
Auto-WEKA [Thornton et al, 2013, Kotthoff et al, 2017, Kotthoff et al. 2019]

- First general AutoML system, carrying out **Combined Algorithm Selection and Hyperparameter optimization (CASH)**, jointly optimizing
 - ▶ Choice of algorithm (out of 26 classifiers)
 - ▶ The algorithm's hyperparameters (up to 10)
 - ▶ Choice of preprocessing method and its hyperparameters
 - ▶ Choice of ensemble & meta methods
- Parameterized WEKA [Frank et al, 2016]: **768 hyperparameters**, 4 leves of conditionality



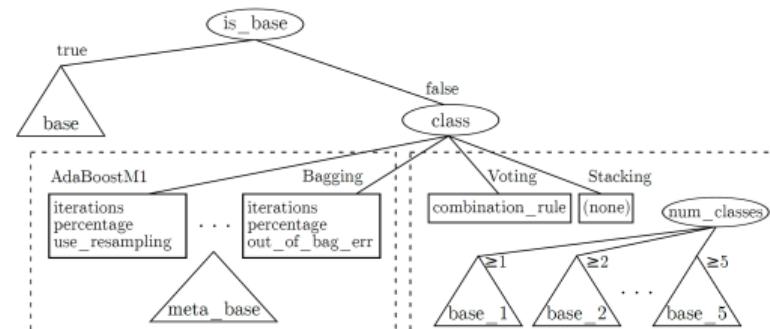
Auto-WEKA [Thornton et al, 2013, Kotthoff et al, 2017, Kotthoff et al. 2019]

- First general AutoML system, carrying out **Combined Algorithm Selection and Hyperparameter optimization (CASH)**, jointly optimizing
 - ▶ Choice of algorithm (out of 26 classifiers)
 - ▶ The algorithm's hyperparameters (up to 10)
 - ▶ Choice of preprocessing method and its hyperparameters
 - ▶ Choice of ensemble & meta methods
- Parameterized WEKA [Frank et al, 2016]: **768 hyperparameters**, 4 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 **Combined Algorithm Selection and Hyperparameter optimization (CASH)**, jointly optimizing
 - ▶ Choice of algorithm (out of 26 classifiers)
 - ▶ The algorithm's hyperparameters (up to 10)
 - ▶ Choice of preprocessing method and its hyperparameters
 - ▶ Choice of ensemble & meta methods
- Parameterized WEKA [Frank et al, 2016]: **768 hyperparameters**, 4 levels of conditionality
- Optimized 10-fold cross-validation via SMAC [Hutter et al, 2011]
- Results:
 - ▶ Better than an oracle of the 26 base classifiers with default hyperparameters
 - ▶ 100× faster than grid search over base classifiers, and still better in 14/21 cases
 - ▶ Better than the only other applicable method TPE in 19/21 cases
- Impact for practitioners: Auto-WEKA plugin was downloaded tens of thousands of times



Questions to Answer for Yourself / Discuss with Friends

- Repetition. List several success stories of Bayesian optimization
- Repetition. List several prominent tools for Bayesian optimization
- Discussion. Recall the algorithm selection problem; how does CASH relate to this (after all, it also has “algorithm selection” as part of its name)? (Hint: they are quite different.)

Outline

- 1 Introduction to Bayesian Optimization
- 2 Computationally Cheap Acquisition Functions
- 3 Computationally Expensive Acquisition Functions
- 4 Surrogate Models
- 5 High-Dimensional Bayesian Optimization
- 6 Extensions to Bayesian Optimization
- 7 The Alternative Optimization Approach of the Tree-Parzen Estimator (TPE)
- 8 Success Stories
- 9 Further Reading

AutoML: Bayesian Optimization for Hyperparameter Optimization

Further Reading

Bernd Bischl Frank Hutter Lars Kotthoff
Marius Lindauer Joaquin Vanschoren

Further Reading

Tutorials on Bayesian Optimization

- A tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning [Brochu et al. 2010]
- Taking the Human out of the Loop: A Review of Bayesian Optimization [Shahriari et al. 2016]
- A Tutorial on Bayesian Optimization [Frazier 2018]

Survey on hyperparameter optimization: [Feurer and Hutter 2019]