Automated Machine Learning
(AutoML)

M. Lindauer F. Hutter

University of Freiburg

AutoML.org

Lecture 4:
Hyperparameter Optimization
Bayesian Optimization

Where are we? The big picture

- Introduction
- Background
  - Design spaces in ML
  - Evaluation and visualization
- → Hyperparameter optimization (HPO)
  - → Bayesian optimization
  - Other black-box techniques
  - Speeding up HPO with multi-fidelity optimization
- Pentecost (Holiday) – no lecture
- Architecture search I + II
- Meta-Learning
- Learning to learn & optimize
- Beyond AutoML: algorithm configuration and control
- Project announcement and closing

Learning Goals

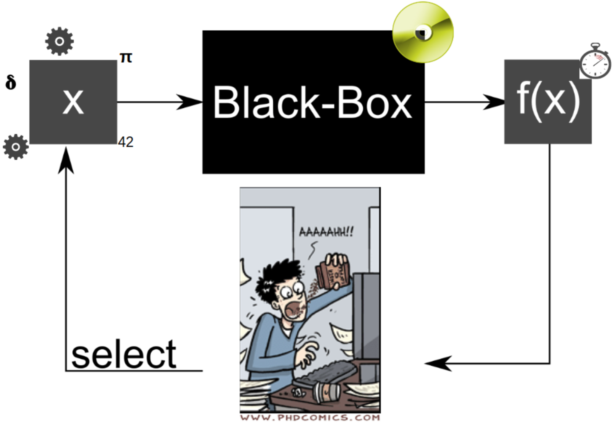
After this lecture, you will be able to . . .

- explain the challenges in hyperparameter optimization
- efficiently optimize black box functions via Bayesian Optimization
- discuss the advantages of different surrogate models
- explain the idea of acquisition functions to trade off exploration and exploitation
- consider important design decisions for BO

Lecture Overview

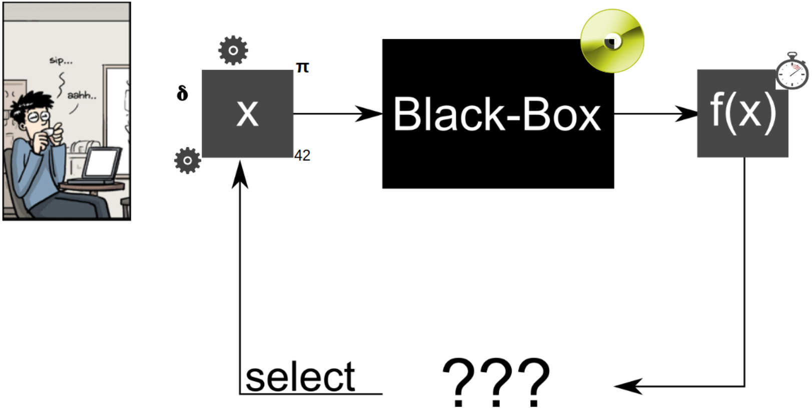
1. Hyperparameter Optimization and Black-Box Optimization
2. Tree-Parzen Estimator
3. Bayesian Optimization
4. Surrogate Models
5. Acquisition Functions
6. Practical Considerations

How to Optimize Black Box Functions?



Only interaction: Query of function at x to obtain $f(x)$

How to Optimize Black Box Functions?



Why Black Box Functions for AutoML?

- Internals of algorithms are often not known (or well understood)
- Extreme: Only way of interaction is running the algorithms

Why Black Box Functions for AutoML?

- Internals of algorithms are often not known (or well understood)
- Extreme: Only way of interaction is running the algorithms

Example: Hyperparameter Optimization (HPO)

Let

- λ be the hyperparameters of an ML algorithm A with domain Λ,
- D_{opt} be a training set which is split into D_{train} and D_{valid}
- $\mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$ denote the loss of A_λ trained on D_{train} and evaluated on D_{valid}.

The *hyper-parameter optimization (HPO)* problem is to find a hyper-parameter configuration that minimizes this loss:

$$\lambda^* \in \arg\min_{\lambda \in \Lambda} \mathcal{L}(A_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$$

Challenges in HPO

What could be challenges in hyperparameter optimization? [2min]

Challenges in HPO

- function evaluations are very expensive
  - training a single ML-pipeline can require minutes (or even hours)

  $\rightsquigarrow$ exhaustive search is not feasible

Challenges in HPO

- function evaluations are very expensive
    - training a single ML-pipeline can require minutes (or even hours)
  $\rightsquigarrow$ exhaustive search is not feasible

- complex, structured search space
    - small continuous parameter spaces already challenging to optimize
    - typically, we talk about large configuration spaces ($\gg 10$ hyper-parameters)
        - many HPO benchmarks only consider a few continuous parameters
    - mixed parameter types
    - conditional structures

Challenges in HPO

- function evaluations are very expensive
  - training a single ML-pipeline can require minutes (or even hours)
  - $\rightsquigarrow$ exhaustive search is not feasible
- complex, structured search space
  - small continuous parameter spaces already challenging to optimize
  - typically, we talk about large configuration spaces ($\gg 10$ hyper-parameters)
    - many HPO benchmarks only consider a few continuous parameters
  - mixed parameter types
  - conditional structures
- resembles a black-box optimization problem
  - Input: hyperparameter configuration
  - Black box: ML pipeline
  - Output: loss
  - Note: no gradient information available

Challenges in HPO

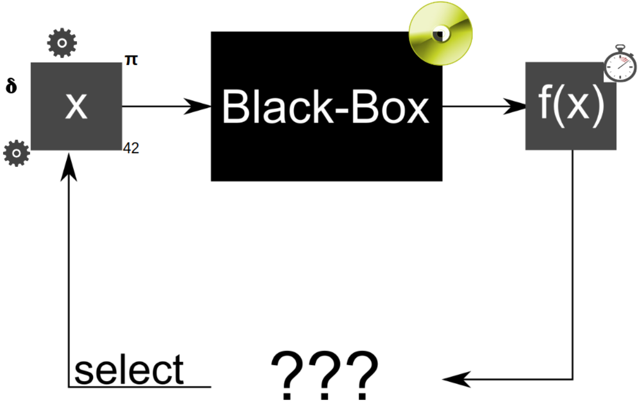
- function evaluations are very expensive
  - training a single ML-pipeline can require minutes (or even hours)

  $\rightsquigarrow$ exhaustive search is not feasible

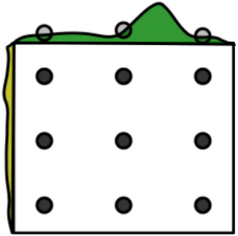
- complex, structured search space
  - small continuous parameter spaces already challenging to optimize
  - typically, we talk about large configuration spaces ($\gg 10$ hyper-parameters)
    - many HPO benchmarks only consider a few continuous parameters
  - mixed parameter types
  - conditional structures

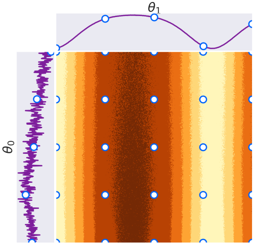
- resembles a black-box optimization problem
  - Input: hyperparameter configuration
  - Black box: ML pipeline
  - Output: loss
  - Note: no gradient information available
- (We discuss grey-box approaches in later sessions)

How to Optimize Black Box Functions?



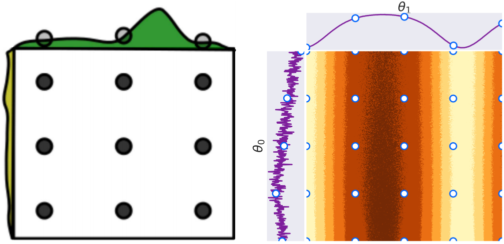
Grid Search [Bergstra et al. '12]





Pros and Cons

What are potential pros and cons of grid search?



Pros and Cons

What are potential pros and cons of grid search?

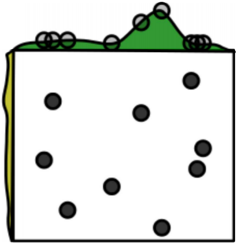
Pros:

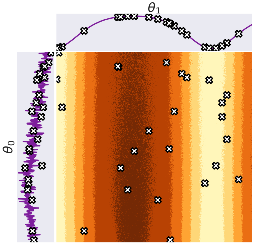
- easy to implement
- easy to parallelize
- exploratory studies

Cons:

- discretization of $\mathbf{\Lambda}$?
- does not scale with #hyperparameters
- inefficient if not all hyperparameters are important

Random Search [Bergstra et al. '12]

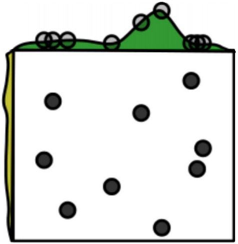


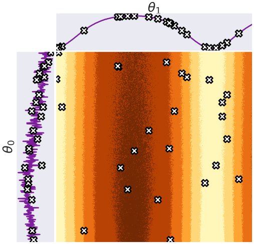


Pros and Cons

What are potential pros and cons of random search?

Random Search [Bergstra et al. '12]





Pros and Cons

What are potential pros and cons of random search?

Pros:

- even easier to implement
- easy to parallelize
- more evaluations along each parameter

Cons:

- does not scale with #hyper-parameters
- purely explorative

Optimization for HPO

- HPO is expensive $\rightsquigarrow$ we also need a greedy component
- Basically all black-box optimization approaches can be used
  - Local search
    - hypothesis: HPO surfaces have many local optima such that local search gets easily trapped in local optima
  - population-based evolution strategies (more next week)
    - hypothesis: often needs many function evaluations to perform well
  - model-based approaches
    - Idea: use an surrogate model to approximate the unknown function to be optimized and do parts of the optimization on these surrogate models
  - Remark: there are also combinations of these approaches (such as model-based ES)

Black-Box Optimization: Remarks

- Expensive black-box optimization is a quite common problem in many disciplines (not only limited to HPO)
- Examples include:
  - General algorithm configuration
  - parameter optimization of expensive simulations
  - automatic design of experiments (e.g., in material science)
  - tuning of robots
  - experimental particle physics
  - and many more ...

Black-Box Optimization: Remarks

- Expensive black-box optimization is a quite common problem in many disciplines (not only limited to HPO)
- Examples include:
  - General algorithm configuration
  - parameter optimization of expensive simulations
  - automatic design of experiments (e.g., in material science)
  - tuning of robots
  - experimental particle physics
  - and many more ...
- You should consider model-based optimization approaches, such as Bayesian Optimization, if ...
  1. you can afford only a few function evaluations
  2. you don't care too much about the overhead induced by the optimization algorithm

Lecture Overview

1. Hyperparameter Optimization and Black-Box Optimization
2. Tree-Parzen Estimator
3. Bayesian Optimization
4. Surrogate Models
5. Acquisition Functions
6. Practical Considerations

Tree-Parzen Estimator [Bergstra et al. 2011]

- Assume that we already observed some configuration and the corresponding loss $D = \{(\lambda_i, y_i))\}_{i=1}^{N}$
  - let's use $y_i = f(\lambda)$ as a short form for $\mathcal{L}(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$

Tree-Parzen Estimator [Bergstra et al. 2011]

- Assume that we already observed some configuration and the corresponding loss $D = \{(\lambda_i, y_i))\}_{i=1}^{N}$
  - let's use $y_i = f(\lambda)$ as a short form for $\mathcal{L}(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$
- We could approximate the good and the bad regions of the configuration space $\mathbf{\Lambda}$

$$p(\lambda|y) = \begin{cases} l(\lambda) \text{ if } y < y^* \\ g(\lambda) \text{ otherwise} \end{cases}$$

where

- y^* is an empirical threshold for a well-performing configuration (e.g., a γ percentile of all observed y in D)

Tree-Parzen Estimator [Bergstra et al. 2011]

- Assume that we already observed some configuration and the corresponding loss $D = \{(\lambda_i, y_i))\}_{i=1}^{N}$
  - let's use $y_i = f(\lambda)$ as a short form for $\mathcal{L}(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$
- We could approximate the good and the bad regions of the configuration space $\mathbf{\Lambda}$

$$p(\lambda|y) = \begin{cases} l(\lambda) \text{ if } y < y^* \\ g(\lambda) \text{ otherwise} \end{cases}$$

where

- y^* is an empirical threshold for a well-performing configuration (e.g., a γ percentile of all observed y in D)
- $l(\lambda)$ models the density of the well-performing region based on D
  - Note that we minimize!

Tree-Parzen Estimator [Bergstra et al. 2011]

- Assume that we already observed some configuration and the corresponding loss $D = \{(\lambda_i, y_i))\}_{i=1}^{N}$
  - let's use $y_i = f(\lambda)$ as a short form for $\mathcal{L}(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$
- We could approximate the good and the bad regions of the configuration space $\mathbf{\Lambda}$

$$p(\lambda|y) = \begin{cases} l(\lambda) \text{ if } y < y^* \\ g(\lambda) \text{ otherwise} \end{cases}$$

where

- y^* is an empirical threshold for a well-performing configuration (e.g., a γ percentile of all observed y in D)
- $l(\lambda)$ models the density of the well-performing region based on D
  - Note that we minimize!
- $g(\lambda)$ models the density of the poorly performing region based on D

Tree-Parzen Estimator [Bergstra et al. 2011]

- Assume that we already observed some configuration and the corresponding loss $D = \{(\lambda_i, y_i))\}_{i=1}^{N}$
  - let's use $y_i = f(\lambda)$ as a short form for $\mathcal{L}(\mathcal{A}_\lambda, \mathcal{D}_{train}, \mathcal{D}_{valid})$
- We could approximate the good and the bad regions of the configuration space $\mathbf{\Lambda}$

$$p(\lambda|y) = \begin{cases} l(\lambda) \text{ if } y < y^* \\ g(\lambda) \text{ otherwise} \end{cases}$$

where

- y^* is an empirical threshold for a well-performing configuration (e.g., a γ percentile of all observed y in D)
- $l(\lambda)$ models the density of the well-performing region based on D
  - Note that we minimize!
- $g(\lambda)$ models the density of the poorly performing region based on D
- g and l can be modeled by kernel density estimator (KDE)

Algorithm 1: Optimization with TPE

Input : Configuration Space $\mathbf{\Lambda}$, black box function f, maximal number of function evaluations m, percentile γ

$D_0 \leftarrow$ initial_design($\mathbf{\Lambda}$);

Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]

Algorithm 2: Optimization with TPE

Input : Configuration Space $\mathbf{\Lambda}$, black box function f, maximal number of function evaluations m, percentile γ

$D_0 \leftarrow$ initial_design($\mathbf{\Lambda}$);
for $n = 1, 2, \ldots m - |D_0|$ **do**
    $D_{\text{good}}, D_{\text{bad}} \leftarrow$ Split D_{n-1} into good and bad observations according to γ percentile of all observed y;

Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]

Algorithm 3: Optimization with TPE

Input : Configuration Space $\mathbf{\Lambda}$, black box function f, maximal number of function evaluations m, percentile γ

$D_0 \leftarrow$ initial_design($\mathbf{\Lambda}$);
for $n = 1, 2, \ldots m - |D_0|$ **do**
  $D_{\text{good}}, D_{\text{bad}} \leftarrow$ Split D_{n-1} into good and bad observations according to γ percentile of all observed y;
  $l(\lambda) \leftarrow$ fit KDE on D_{good};
  $g(\lambda) \leftarrow$ fit KDE on D_{bad};

Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]

Algorithm 4: Optimization with TPE

Input : Configuration Space $\mathbf{\Lambda}$, black box function f, maximal number of function evaluations m, percentile γ

$D_0 \leftarrow$ initial_design($\mathbf{\Lambda}$);
for $n = 1, 2, \ldots m - |D_0|$ **do**
    $D_{\text{good}}, D_{\text{bad}} \leftarrow$ Split D_{n-1} into good and bad observations according to γ percentile of all observed y;
    $l(\lambda) \leftarrow$ fit KDE on D_{good};
    $g(\lambda) \leftarrow$ fit KDE on D_{bad};
    $\Lambda_{\text{cand}} \leftarrow$ draw examples according to l;
    select λ_n by optimizing $\lambda_n \in \arg\max_{\lambda \in \Lambda_{\text{cand}}} l(\lambda)/g(\lambda)$;

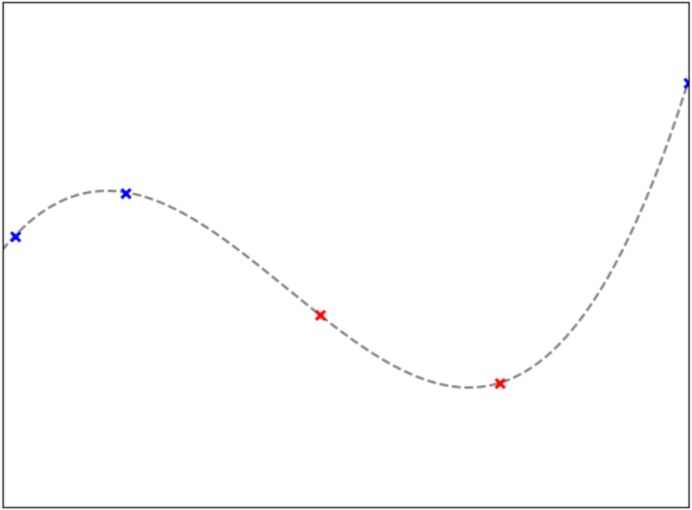
Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]

Algorithm 5: Optimization with TPE

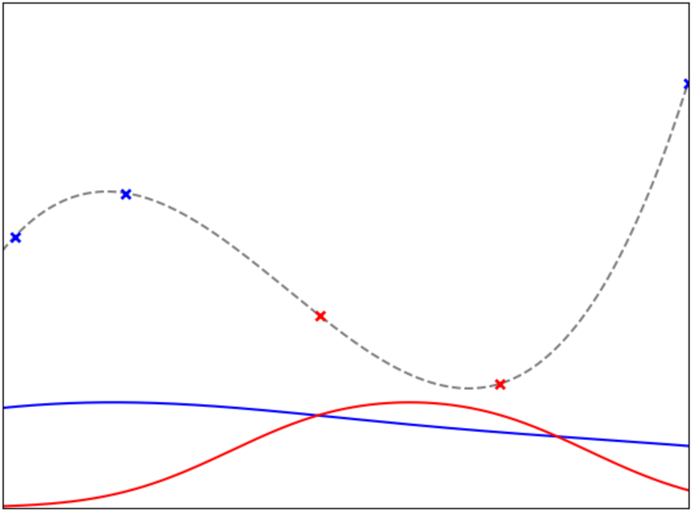
Input : Configuration Space $\mathbf{\Lambda}$, black box function f, maximal number of function evaluations m, percentile γ

$D_0 \leftarrow$ initial_design($\mathbf{\Lambda}$);
for $n = 1, 2, \ldots m - |D_0|$ **do**
  $D_{\text{good}}, D_{\text{bad}} \leftarrow$ Split D_{n-1} into good and bad observations according to γ percentile of all observed y;
  $l(\lambda) \leftarrow$ fit KDE on D_{good};
  $g(\lambda) \leftarrow$ fit KDE on D_{bad};
  $\Lambda_{\text{cand}} \leftarrow$ draw examples according to l;
  select λ_n by optimizing $\lambda_n \in \arg\max_{\lambda \in \Lambda_{\text{cand}}} l(\lambda)/g(\lambda)$;
  Query $y_n := f(\lambda_n)$;
  Add observation to data $D_n := D_{n-1} \cup \{\langle \lambda_n, y_n \rangle\}$;
return *Best observed* λ *according to* D_m

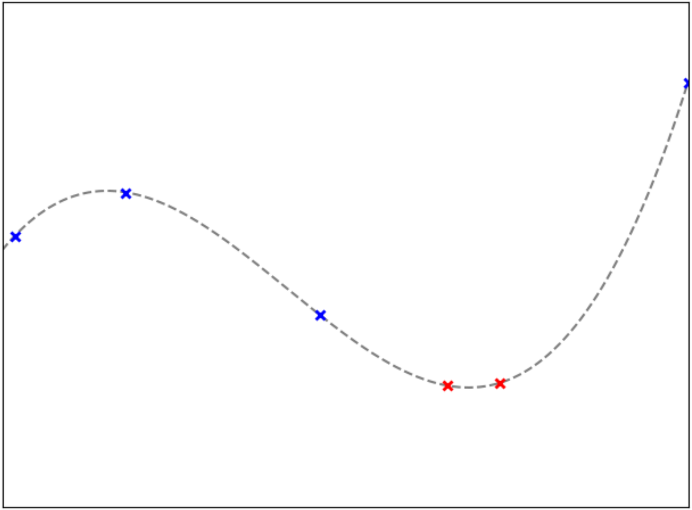
Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]



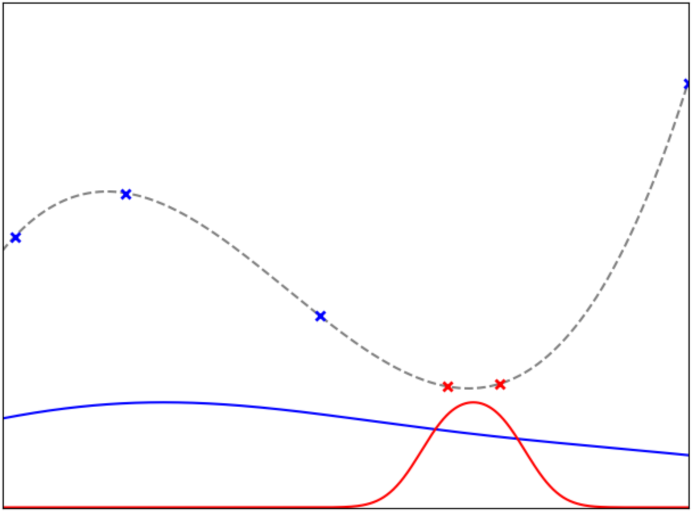
Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]



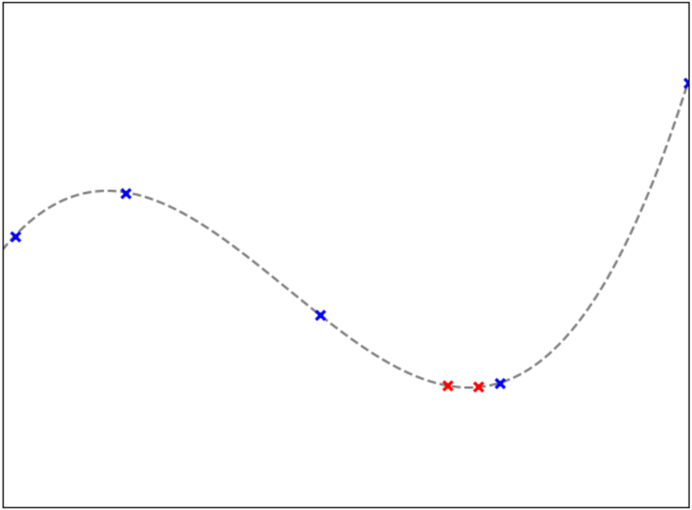
Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]



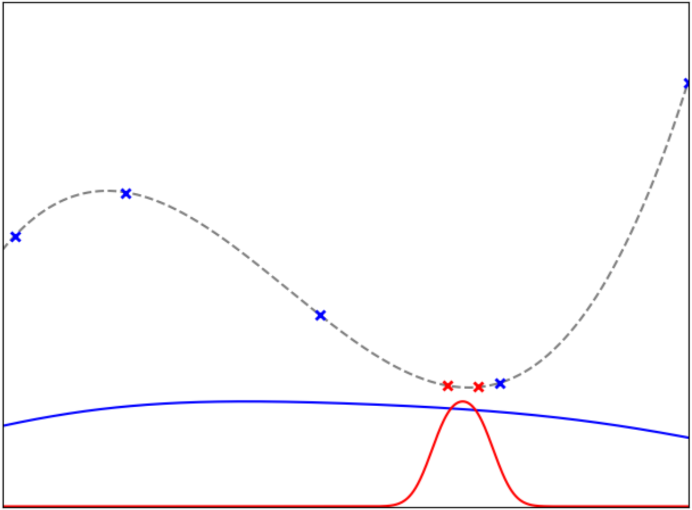
Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]



Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]



Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]



Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]

Remarks:

- TPE models $p(\lambda|y)$
  - we can multiply it with a prior to add expert knowledge

Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]

Remarks:

- TPE models $p(\lambda|y)$
  - we can multiply it with a prior to add expert knowledge
- Performance of TPE depends on:
  1. setting of γ to trade-off exploration and exploitation
  2. band width of the KDEs

Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]

Remarks:

- TPE models $p(\lambda|y)$
  - we can multiply it with a prior to add expert knowledge
- Performance of TPE depends on:
  1. setting of γ to trade-off exploration and exploitation
  2. band width of the KDEs
- optimizing $l(\lambda)/g(\lambda)$ is equivalent to optimizing *expected improvement* as acquisition function in Bayesian Optimization (more in a second)

Optimization with Tree-Parzen Estimator [Bergstra et al. 2011]

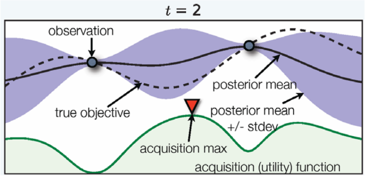
Remarks:

- TPE models $p(\lambda|y)$
  - we can multiply it with a prior to add expert knowledge
- Performance of TPE depends on:
  1. setting of γ to trade-off exploration and exploitation
  2. band width of the KDEs
- optimizing $l(\lambda)/g(\lambda)$ is equivalent to optimizing *expected improvement* as acquisition function in Bayesian Optimization (more in a second)
- successful tool implementing TPE is HyperOpt

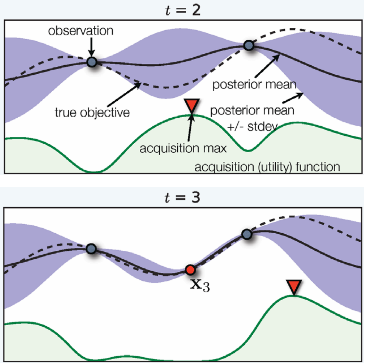
Lecture Overview

1. Hyperparameter Optimization and Black-Box Optimization
2. Tree-Parzen Estimator
3. Bayesian Optimization
4. Surrogate Models
5. Acquisition Functions
6. Practical Considerations

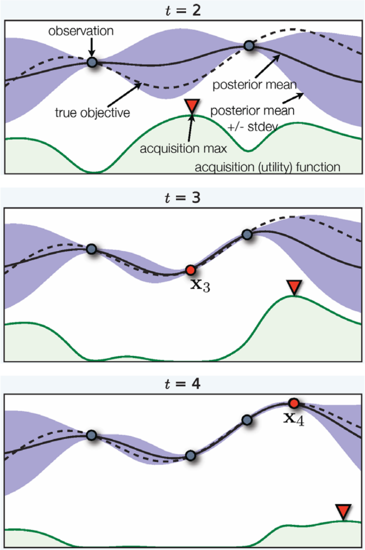
Bayesian Optimization [Mockus 1977, Jones et al. 1998]

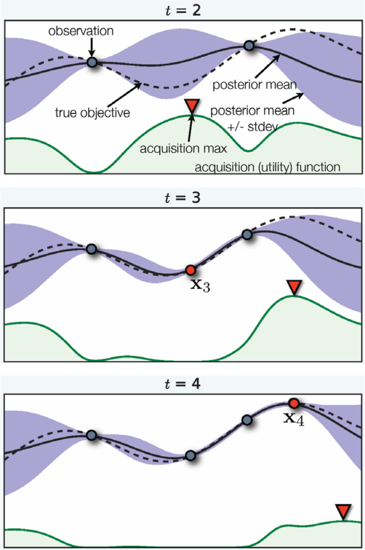


Bayesian Optimization [Mockus 1977, Jones et al. 1998]



Bayesian Optimization [Mockus 1977, Jones et al. 1998]

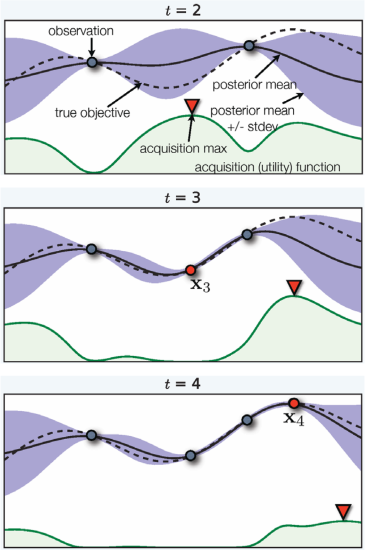




General Approach:

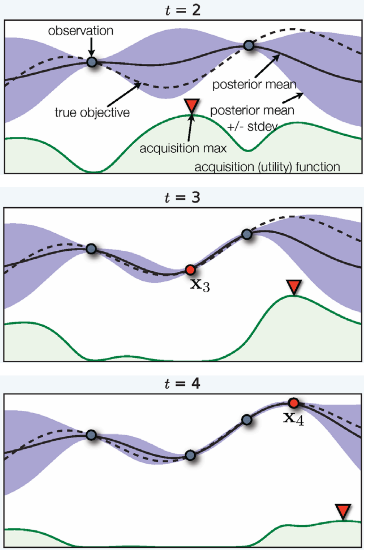
1. Fit model $p(f(\lambda)|\lambda)$ on collected observations $\langle \lambda_t, f(\lambda_t) \rangle$

Bayesian Optimization [Mockus 1977, Jones et al. 1998]



General Approach:

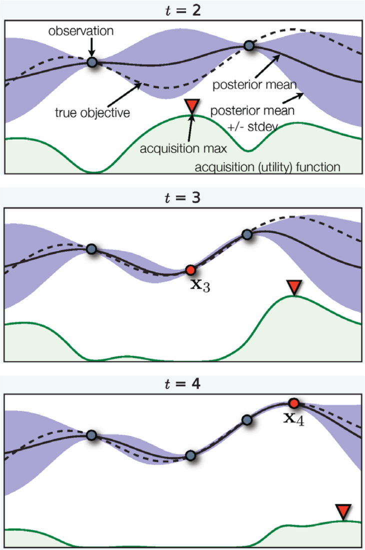
1. Fit model $p(f(\lambda)|\lambda)$ on collected observations $\langle \lambda_t, f(\lambda_t) \rangle$
2. use acquisition function a to trade off exploration and exploitation



General Approach:

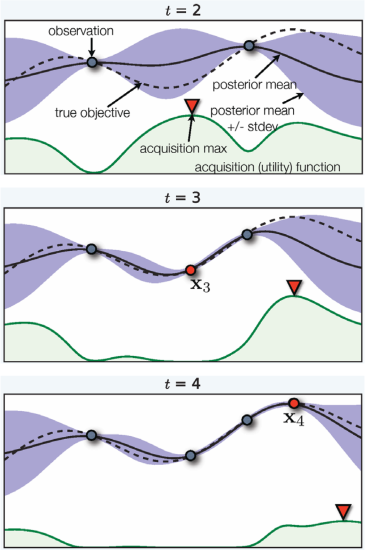
1. Fit model $p(f(\lambda)|\lambda)$ on collected observations $\langle \lambda_t, f(\lambda_t) \rangle$
2. use acquisition function a to trade off exploration and exploitation
3. maximize acquisition function: $x^* \in \arg\min a(x)$

Bayesian Optimization [Mockus 1977, Jones et al. 1998]



General Approach:

1. Fit model $p(f(\lambda)|\lambda)$ on collected observations $\langle \lambda_t, f(\lambda_t) \rangle$
2. use acquisition function a to trade off exploration and exploitation
3. maximize acquisition function: $x^* \in \arg\min a(x)$
4. obtain new observation at x^*



General Approach:

1. Fit model $p(f(\lambda)|\lambda)$ on collected observations $\langle \lambda_t, f(\lambda_t) \rangle$
2. use acquisition function a to trade off exploration and exploitation
3. maximize acquisition function: $x^* \in \arg\min a(x)$
4. obtain new observation at x^*

Moving pieces:

- Which model family to use
- How to use the model to guide optimization
  - Determined by $a(x)$ (Which data point should I *acquire* next?)

Bayesian Optimization: Algorithm

Algorithm 6: Bayesian Optimization (BO)

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

$\mathcal{D}_0 \leftarrow$ initial_design($\mathcal{X}$);
for $n = 1, 2, \ldots m - |D_0|$ **do**
    $\hat{f} : \lambda \mapsto y \leftarrow$ fit predictive model on $\mathcal{D}_{n-1}$;
    select x_n by optimizing $x_n \in \arg\max_{x \in \mathcal{X}} \alpha(x; \mathcal{D}_{n-1}, \hat{f})$;
    Query $y_n := f(x_n)$;
    Add observation to data $D_n := D_{n-1} \cup \{\langle x_n, y_n \rangle\}$;
return *Best* x *according to* D_m *or* $\hat{f}$

Pros and Cons

Pros:

- sample efficient
- can be applied to many black-box functions with expensive function evaluations (not only HPO)

Cons:

- overhead because of model training in each iteration
- hard to efficiently parallelize
- (requires good surrogate model)

Lecture Overview

1. Hyperparameter Optimization and Black-Box Optimization
2. Tree-Parzen Estimator
3. Bayesian Optimization
4. Surrogate Models
5. Acquisition Functions
6. Practical Considerations

Surrogate Model

Required features

- Mandatory:
  - Regression model
  - Uncertainty estimates

Surrogate Model

Required features

- Mandatory:
  - Regression model
  - Uncertainty estimates
- Preferable:
  - accurate predictions
  - cheap-to-train
  - scales with the complexity of the data (number of features and observations)
  - can handle different types of features (categorical and continuous)

Surrogate Model

Required features

- Mandatory:
  - Regression model
  - Uncertainty estimates
- Preferable:
  - accurate predictions
  - cheap-to-train
  - scales with the complexity of the data (number of features and observations)
  - can handle different types of features (categorical and continuous)

- Candidates:
  - Gaussian Processes (quite common)
  - Random Forests (our default choice)
  - Deep Neural Networks (recent trend)

Gaussian Process

- There can be several explanations (i.e., functions) for a set of observations

Gaussian Process

- There can be several explanations (i.e., functions) for a set of observations
- A good model would integrate over all these possible explanations (and weight them by their probability)

Gaussian Process

- There can be several explanations (i.e., functions) for a set of observations
- A good model would integrate over all these possible explanations (and weight them by their probability)
- Gaussian Processes (GPs) do exactly that by using a normal distribution assumption at each point.

Informal Definition

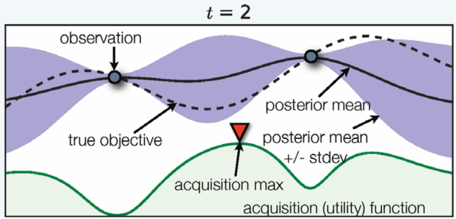
A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution

Gaussian Process

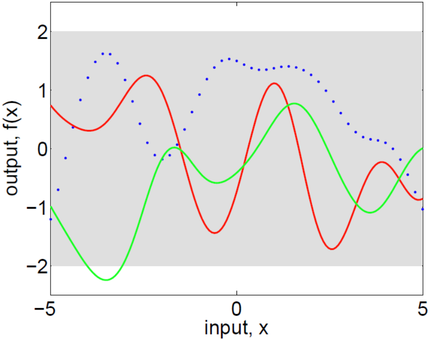
- There can be several explanations (i.e., functions) for a set of observations
- A good model would integrate over all these possible explanations (and weight them by their probability)
- Gaussian Processes (GPs) do exactly that by using a normal distribution assumption at each point.

Informal Definition

A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution



Gaussian Process: Prior

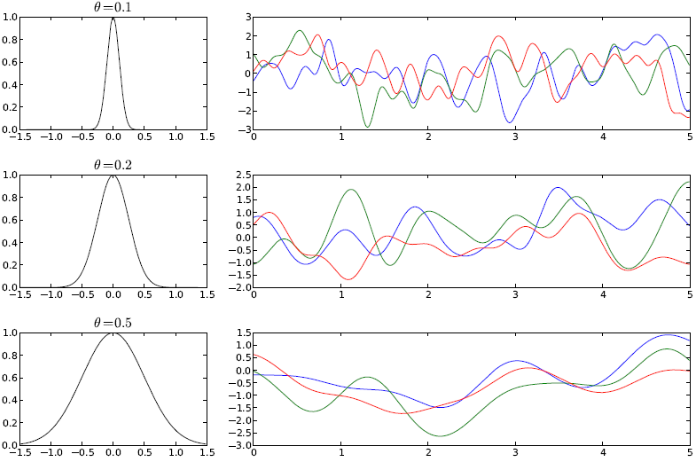


(a), prior

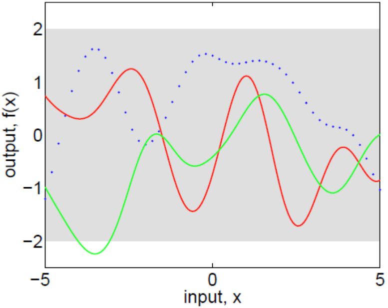
- Samples from the prior are zero-mean, with values drawn from a multivariate Gaussian distribution $\mathcal{N}(0, \mathbf{K})$
- The kernel $\mathbf{K}$ tells us how correlated the function values at two points are

Gaussian Process: RBF Kernel

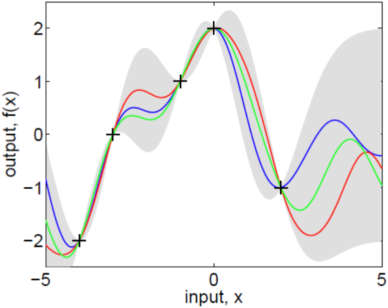
$$k(x_i, x_j) = \exp\left(-\frac{1}{2\theta^2}||x_i - x_j||^2\right)$$



Gaussian Process: Posterior



(a), prior



(b), posterior

$$\begin{aligned} P(f_{t+1}|D_t, \mathbf{x}_{t+1}) &= \mathcal{N}(\mu_t(\mathbf{x}_{t+1}), \sigma_t^2(\mathbf{x}_{t+1})) \text{ where} \\ \mu_t(\mathbf{x}_{t+1}) &= \mathbf{k}^T \mathbf{K}^{-1} f_{1:t} \\ \sigma_t^2(\mathbf{x}_{t+1}) &= k(\mathbf{x}_{t+1}, \mathbf{x}_{t+1}) \mathbf{k}^T \mathbf{K}^{-1} \mathbf{k} \end{aligned}$$

Gaussian Process: Remarks

- GPs are very good models for small-dimensional, continuous functions
- Advantage: we can encode in the kernel expert knowledge about the design space

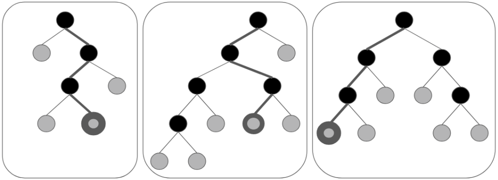
Gaussian Process: Remarks

- GPs are very good models for small-dimensional, continuous functions
- Advantage: we can encode in the kernel expert knowledge about the design space
- Disadvantage: we have to define a good kernel for each application (if we don't optimize small-dimensional, continuous functions)
  - e.g., special kernels for categorical hyperparameters and conditional dependencies

Gaussian Process: Remarks

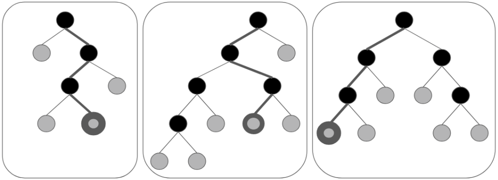
- GPs are very good models for small-dimensional, continuous functions
- Advantage: we can encode in the kernel expert knowledge about the design space
- Disadvantage: we have to define a good kernel for each application (if we don't optimize small-dimensional, continuous functions)
    - e.g., special kernels for categorical hyperparameters and conditional dependencies
- GPs have a cubic scaling with the number of observations (because of inverting the kernel)
    - to address this issue, there are sparse GPs [Snelson and Ghahramani. 2005]

Random Forests as Surrogate Models



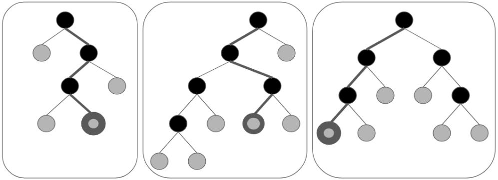
- Train:
  - n decision (or regression) trees
  - subsampled training data for each tree (with bootstrapping)
  - (subsampled feature set for each split)

Random Forests as Surrogate Models



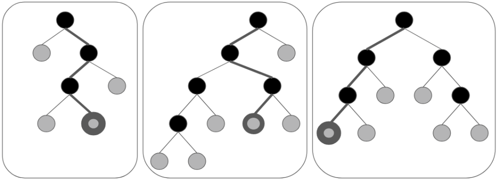
- Train:
  - n decision (or regression) trees
  - subsampled training data for each tree (with bootstrapping)
  - (subsampled feature set for each split)
- $\rightsquigarrow$ each tree gives us a possible explanation for the observations

Random Forests as Surrogate Models



- Train:
  - n decision (or regression) trees
  - subsampled training data for each tree (with bootstrapping)
  - (subsampled feature set for each split)
  - $\rightsquigarrow$ each tree gives us a possible explanation for the observations
- Predict
  - Obtain prediction of each tree
  - Aggregate predictions (e.g., average)

Random Forests as Surrogate Models

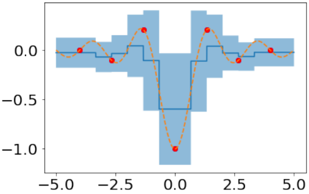


- Train:
  - n decision (or regression) trees
  - subsampled training data for each tree (with bootstrapping)
  - (subsampled feature set for each split)

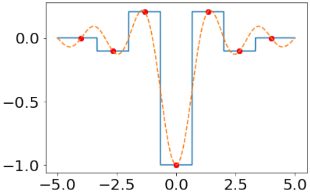
  $\rightsquigarrow$ each tree gives us a possible explanation for the observations
- Predict
  - Obtain prediction of each tree
  - Aggregate predictions (e.g., average)
  - Uncertainty of predictions: stdev across tree predictions

Random Forest's Hyperparameters

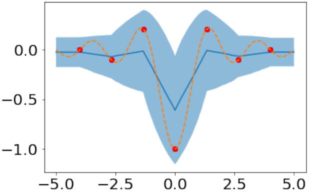
w bootstrapping and
w/o random splits



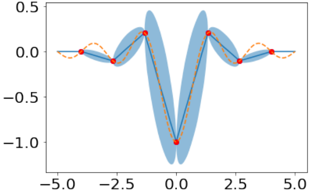
w/o bootstrapping and
w/o random splits



w bootstrapping and
w/ random splits



w/o bootstrapping and
w/ random splits



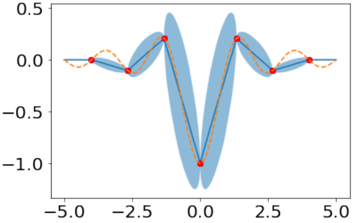
Advantages and Disadvantages of Random Forests

Pros:

- Cheap to train
- Scales well with #observations
  - Worst-case complexity for T tress with n data points of dimensionality p: $\mathcal{O}(T \cdot p \cdot n^2 \log n)$
- training can be parallelized
- Can handle continuous and categorical features
  - most RF implementations can handle only continuous features

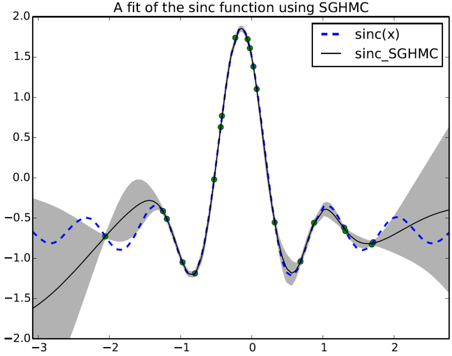
Cons:

- Poor uncertainty estimates
- No extrapolation
  - last seen value for extrapolation (constant)
  - no prior



Deep Neural Networks as Surrogate Models

- DNNs are known to have good predictions given big data
- In Bayesian Optimization (BO), we have (often) little data
- DNNs are not known for very good uncertainty estimates
- Nevertheless, we can use DNN for BO



Source: [Springerberger et al. 2016]

DNGO [Snoek et al. 2015]

- Main idea: combine DNN with a Bayesian linear regression

DNGO [Snoek et al. 2015]

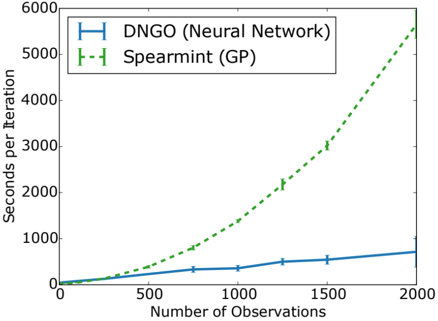
- Main idea: combine DNN with a Bayesian linear regression
- 1st step: Train a DNN with linear regression as last layer to get an embedding $\phi(x)$

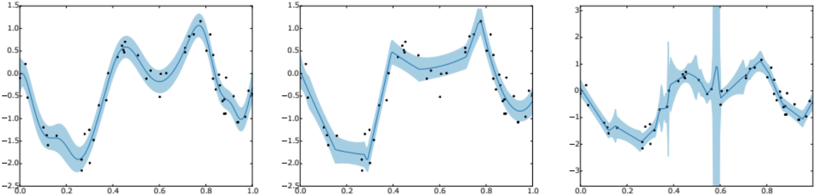
DNGO [Snoek et al. 2015]

- Main idea: combine DNN with a Bayesian linear regression
- 1st step: Train a DNN with linear regression as last layer to get an embedding $\phi(x)$
- Replace last layer by Bayesian linear regressor
  - Bayesian linear regression has two hyperparameters α, β
  - Snoek et al. used slice sampling to integrate these out

DNGO [Snoek et al. 2015]

- Main idea: combine DNN with a Bayesian linear regression
- 1st step: Train a DNN with linear regression as last layer to get an embedding $\phi(x)$
- Replace last layer by Bayesian linear regressor
  - Bayesian linear regression has two hyperparameters α, β
  - Snoek et al. used slice sampling to integrate these out
- DNGO scales much better than GPs:





(a) TanH Units (b) ReLU + TanH Units (c) ReLU Units

- the activation function is quite important to get good uncertainty estimates
- other hyperparameters (such as the architecture of the DNN) are also crucial
  - Snoek et al. used BO to optimize these hyperparameters (as a meta-meta problem)
  - however, the impact of this meta-meta tuning is unknown

Further approaches for BO with DNNs

- [Springenberger et al. 2016] proposed Bayesian Optimization with Hamiltonian Monte Carlo Artificial Neural Networks (BOHAMIANN)
  - uses stochastic gradient Hamiltonian Monte Carlo (SGHMC) for a truly Bayesian approach

Further approaches for BO with DNNs

- [Springenberger et al. 2016] proposed Bayesian Optimization with Hamiltonian Monte Carlo Artificial Neural Networks (BOHAMIANN)
  - uses stochastic gradient Hamiltonian Monte Carlo (SGHMC) for a truly Bayesian approach

- [Lu et al. 2018] proposed to use structured variationally auto-encoders
  1. learn an embedding into a small-dimensional latent space
  2. the latent representation is fed into a GP

Lecture Overview

1. Hyperparameter Optimization and Black-Box Optimization
2. Tree-Parzen Estimator
3. Bayesian Optimization
4. Surrogate Models
5. Acquisition Functions
6. Practical Considerations

The Role of the Acquisition Function

- Given: a model $\hat{f} : \mathbf{\Lambda} \to \mathbb{R}$ that predicts the quality $\mu(\lambda)$ for each configuration λ and its standard deviation $\sigma(\lambda)$ ($\leadsto$ uncertainty)
  - Assume w.l.o.g. that we want to *maximize* f

The Role of the Acquisition Function

- Given: a model $\hat{f} : \mathbf{\Lambda} \to \mathbb{R}$ that predicts the quality $\mu(\lambda)$ for each configuration λ and its standard deviation $\sigma(\lambda)$ ($\rightsquigarrow$ uncertainty)
  - Assume w.l.o.g. that we want to *maximize* f
- Which configuration should we select next? Need to trade off:
  - Exploitation
    (sampling where the predicted mean $\mu(\lambda)$ is small)
  - Exploration
    (sampling where we're uncertain about f; i.e., $\sigma(\lambda)$ is high)

The Role of the Acquisition Function

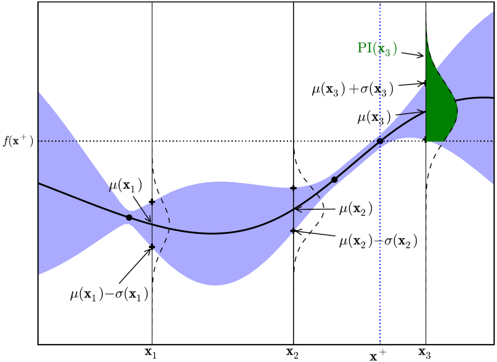
- Given: a model $\hat{f} : \mathbf{\Lambda} \to \mathbb{R}$ that predicts the quality $\mu(\lambda)$ for each configuration λ and its standard deviation $\sigma(\lambda)$ ($\rightsquigarrow$ uncertainty)
  - Assume w.l.o.g. that we want to *maximize* f
- Which configuration should we select next? Need to trade off:
  - Exploitation
    (sampling where the predicted mean $\mu(\lambda)$ is small)
  - Exploration
    (sampling where we're uncertain about f; i.e., $\sigma(\lambda)$ is high)
- Various acquisition functions achieve this trade-off

Probability of Improvement

- Let $f(\theta^+)$ denote the best (here: max) function value known so far.

$$PI(\lambda) \quad = \quad P(f(\lambda) \geq f(\theta^+))) = \Phi\left(\frac{\mu(\theta) - f(\theta^+)}{\sigma(\theta)}\right)$$

- Here, Φ is the cumulative distribution function of the standard normal distribution. (There are $\mathcal{O}(1)$ lookup tables for this.)



Expected Improvement

- Like probability of improvement, but also takes into account the magnitude of the improvement.
- Define the improvement at a point λ as:

$$I(\lambda) = \max(f(\lambda) - f(\lambda^+), 0)$$

Expected Improvement

- Like probability of improvement, but also takes into account the magnitude of the improvement.
- Define the improvement at a point λ as:

$$I(\lambda) = \max(f(\lambda) - f(\lambda^+), 0)$$

- Then, we can compute the expectation of this improvement across the predictive distribution

$$\mathbb{E}[I(x)] = \int_{-\infty}^{\infty} \max(f(\lambda) - f(\lambda^+), 0) \cdot \mathcal{N}(f(\lambda); \mu(\lambda), \sigma^2(\lambda)) df(\lambda)$$

Expected Improvement

- Like probability of improvement, but also takes into account the magnitude of the improvement.
- Define the improvement at a point λ as:

$$I(\lambda) = \max(f(\lambda) - f(\lambda^+), 0)$$

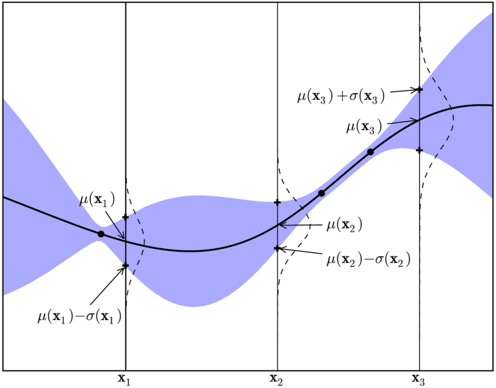
- Then, we can compute the expectation of this improvement across the predictive distribution

$$\mathbb{E}[I(x)] = \int_{-\infty}^{\infty} \max(f(\lambda) - f(\lambda^+), 0) \cdot \mathcal{N}(f(\lambda); \mu(\lambda), \sigma^2(\lambda)) df(\lambda)$$

- This turns out to have a closed form solution:

$$\mathbb{E}[I(x)] = (\mu(\lambda) - f^+)\Phi\left(\frac{\mu(\lambda) - f(\lambda^+)}{\sigma(\lambda)}\right) + \sigma(\lambda)\phi\left(\frac{\mu(\lambda) - f(\lambda^+)}{\sigma(\lambda)}\right)$$

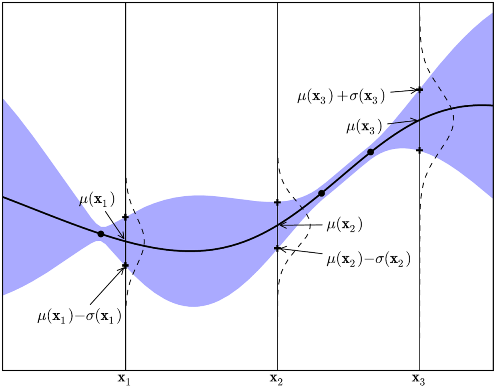
- $\mathrm{UCB}(\lambda) = \mu(\lambda) + \kappa\sigma(\lambda)$, with exploration parameter κ



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

Upper Confidence Bound

- $\text{UCB}(\lambda) = \mu(\lambda) + \kappa\sigma(\lambda)$, with exploration parameter κ



- Which point would we pick next with UCB and $\kappa = 1$?
- $\text{GP-UCB}(\lambda) = \mu(\lambda) + \sqrt{\beta_t}\sigma(\lambda)$, with β_t increasing over time [Srinivas et al. 2009]

Entropy Search [Hennig and Schuler 2012]

- Idea: Learn the probability that x is the minimum

$$p_{\mathsf{min}}(x) = p[x \in \arg\min f(x)]$$

- try to minimize the entropy of the estimated p_{min} distribution
- Parts of the approach
  1. Estimate $p(f)$ e.g., using a GP (as before)
  2. Approximate p_{min} by representer points and monte-carle simulations
  3. The acquisition function basically measures how the entropy of p_{min} is reduced if we would add a new observations
     - relates to gaining more information where the optimum is

Entropy Search [Hennig and Schuler 2012]

- Idea: Learn the probability that x is the minimum

$$p_{\text{min}}(x) = p[x \in \arg\min f(x)]$$

- try to minimize the entropy of the estimated p_{min} distribution
- Parts of the approach
  1. Estimate $p(f)$ e.g., using a GP (as before)
  2. Approximate p_{min} by representer points and monte-carle simulations
  3. The acquisition function basically measures how the entropy of p_{min} is reduced if we would add a new observations
     - relates to gaining more information where the optimum is

$\rightsquigarrow$ This approach does not necessarily samples at the peak of p_{min}; You have to trust your model to obtain the expected best x

Lecture Overview

1. Hyperparameter Optimization and Black-Box Optimization
2. Tree-Parzen Estimator
3. Bayesian Optimization
4. Surrogate Models
5. Acquisition Functions
6. Practical Considerations

Preprocessing Configuration

Convert λ for ML Model

- continuous and integer parameter can be directly passed
  - scaled to $[0, 1]$

Preprocessing Configuration

Convert λ for ML Model

- continuous and integer parameter can be directly passed
  - scaled to $[0, 1]$
- categorical parameters should be encoded if necessary
  - e. g., random forest can handle categorical parameter natively (Not all implementations are able to do it!)
  - use one-hot encoding if necessary:
    - add new variable for each possible value v_i
    - set one of these variable to one (depending on the configuration)

Preprocessing Configuration

Convert λ for ML Model

- continuous and integer parameter can be directly passed
  - scaled to $[0, 1]$
- categorical parameters should be encoded if necessary
  - e. g., random forest can handle categorical parameter natively (Not all implementations are able to do it!)
  - use one-hot encoding if necessary:
    - add new variable for each possible value v_i
    - set one of these variable to one (depending on the configuration)
- ordinal parameters can be converted to integers or also one-hot encoded

Preprocessing Configuration (cont'd)

Conditional Hyperparameters

- Configuration spaces often have a hierarchical structure
- e. g., hyperparameter A is only active if heuristic H is used

Preprocessing Configuration (cont'd)

Conditional Hyperparameters

- Configuration spaces often have a hierarchical structure
- e. g., hyperparameter A is only active if heuristic H is used

$\rightsquigarrow$ List of hyperparameter values has missing entries because of inactive hyperparameters

Preprocessing Configuration (cont'd)

Conditional Hyperparameters

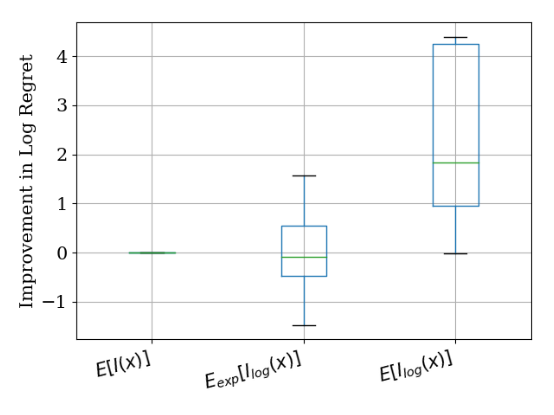
- Configuration spaces often have a hierarchical structure
- e. g., hyperparameter A is only active if heuristic H is used

$\rightsquigarrow$ List of hyperparameter values has missing entries because of inactive hyperparameters

- Fixes:
  - Impute missing data, e. g., by default setting
  - Mark these inactive hyperparameters and let the model deal with it e. g., a random forest could only split on active parameters

Important Design Dimensions in BO

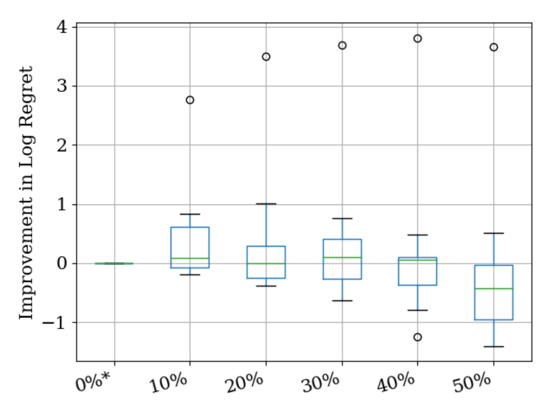
Transformation of y-values



- log-transformed values to fit the model and the acquisition function improves performance
- less emphasize on large outlier values
- focus more on small improvements and less on exploration in unexplored spaces

Important Design Dimensions in BO

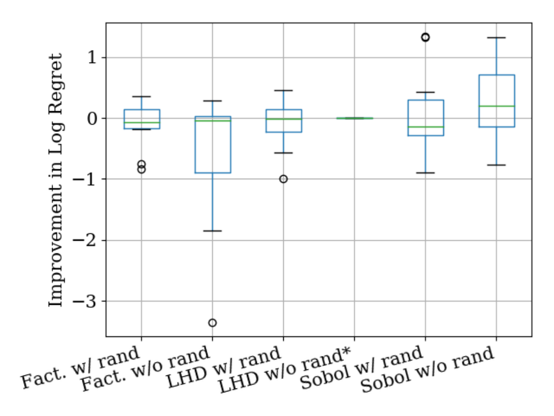
Interleaving Random Points



- RFs don't extrapolate well
- Interpolation between two observations with similar function values leads to constant uncertainty estimates
- BO with RFs can easily get stuck in local optima
- interleave randomly sampled points to escape local optima

Important Design Dimensions in BO

Initial Design



- Alternative to exploration via randomly sampled points
- explore the space before the actual BO takes place
- Pro: will improve the model in early iterations
- Con: will invest a considerable number of function evaluations without taking the already gathered knowledge into account

Learning Goals

After this lecture, you are able to ...

- explain the challenges in hyperparameter optimization
- efficiently optimize black box functions via Bayesian Optimization
- discuss the advantages of different surrogate models
- explain the idea of acquisition functions to trade off exploration and exploitation
- consider important design decisions for BO

Literature [These are links]

- [Jones et al. 1998. Efficient Global Optimization of Expensive Black-Box Functions]
- [Srinivas et al. 2009. Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design]
- [Hutter et al. 2011. Sequential Model-Based Optimization for General Algorithm Configuration]
- [Shahriari et al. 2017. Taking the Human Out of the Loop: A Review of Bayesian Optimization]