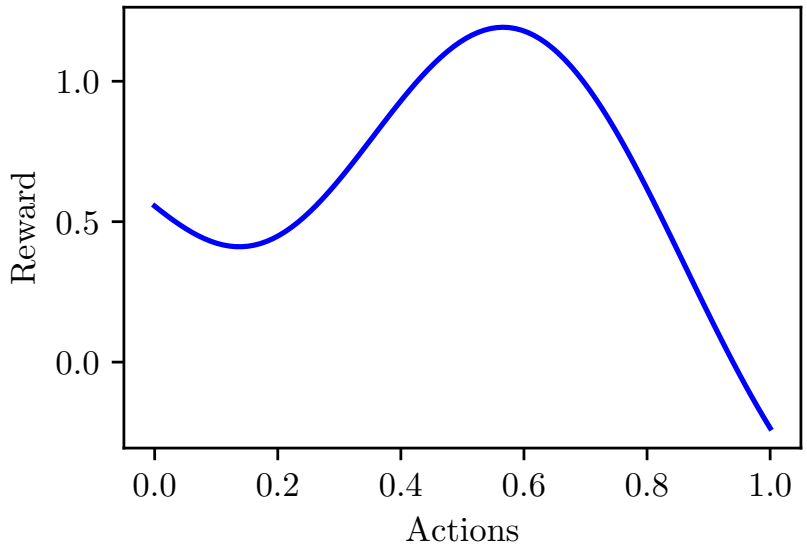
Lecture 17: Bayesian Optimization

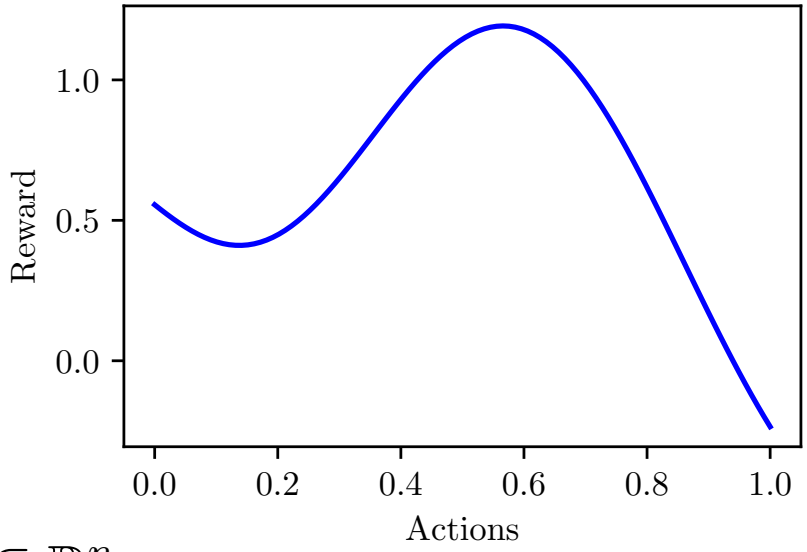
- Pure Exploration
- Bayesian Optimization
- Acquisition Functions
- Practical challenges

Recall: Online function approximation



- Sequentially select locations where to observe the function
- Noisy observations
- Gathering an observation is not *free*

Recall: Stochastic bandit with structured actions



- Action space $\mathcal{X} \subseteq \mathbb{R}^n$
- Reward function $f : \mathcal{X} \mapsto \mathbb{R}$
- For each round t:
  1. Select an action $x_t \in \mathcal{X}$
  2. Observe reward $r_t = f(x_t) + \epsilon_t$ $\quad \leftarrow$ Observation noise ϵ_t

Goal: Maximize $\sum_{t=1}^{T} f(x_t) \rightarrow$ play $x_\star = \arg\max_{x \in \mathcal{X}} f(x)$

Exploration/Exploitation

Minimize regret: $R_T = \sum_{t=1}^{T} (f(x_\star) - f(x_t))$

- *Learning while earning*
- Maximize performance *during* learning
- No dedicated learning phase
- Examples:
  - Treatment optimization
  - Tuning with A/B testing on customers

Pure Exploration

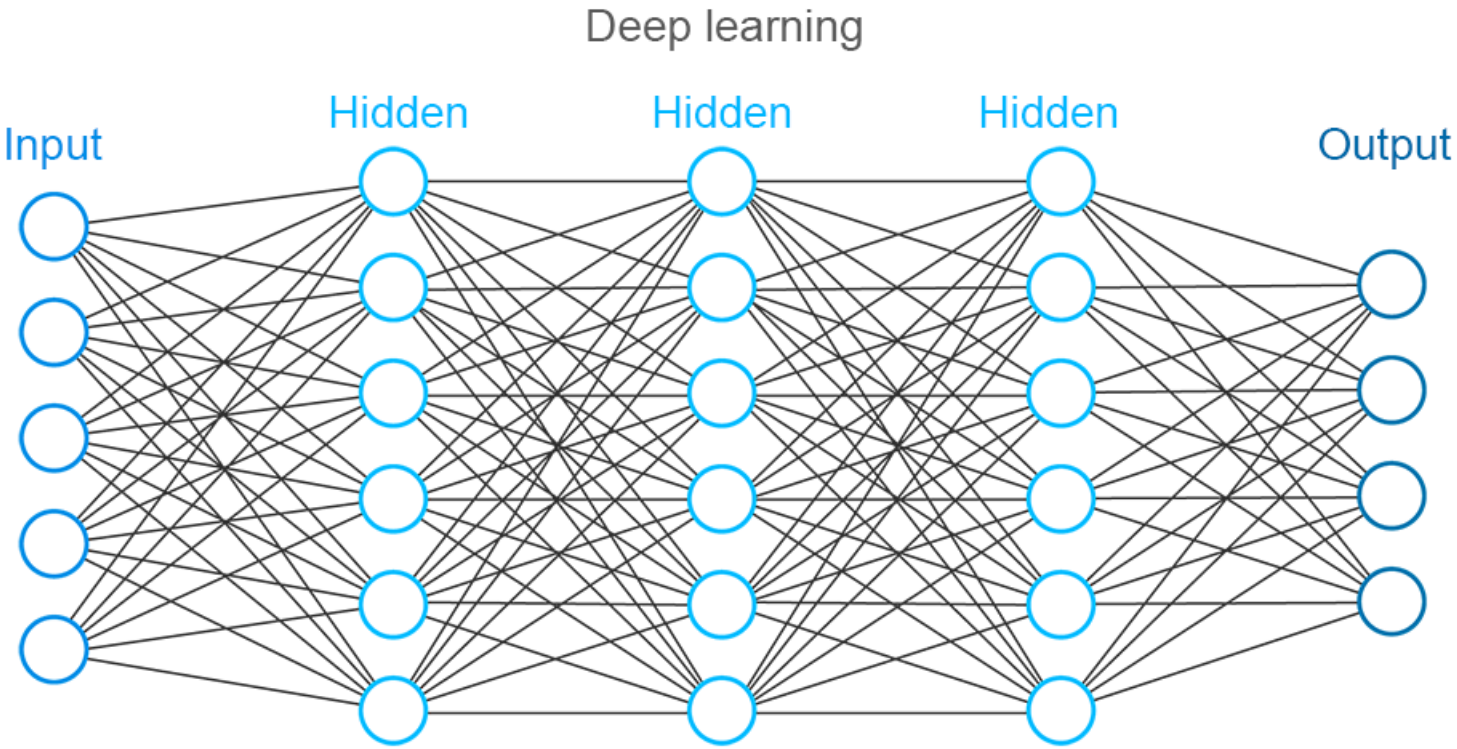
- Finite budget for exploration (evaluations of f are expensive)
- Dedicated exploration phase
- Exploration *does not hurt* during this phase
- Examples:
  - Tuning with A/B testing on a user study
  - Parameter tuning in machine learning algorithms

Example: Tuning GUI with A/B testing

What position of buttons/windows/ads brings more clicks?
More downloads? More purchases?

Real customers vs User study?

Example: Parameter tuning in ML algorithms



From: altexsoft.com

How many hidden layers? How many units per hidden layer? Learning rate? Regularization?

AutoML

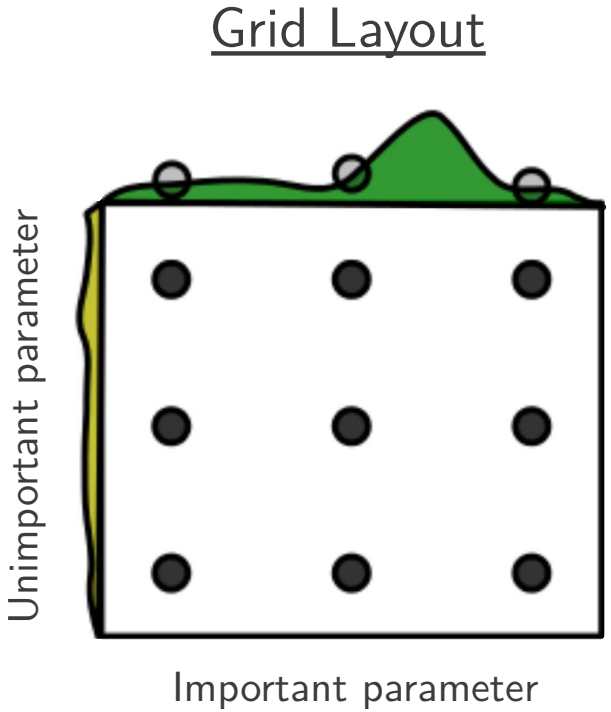
- Automated data preparation/task detection (e.g., binary classification, regression, clustering, ranking)
- Automated feature engineering (e.g. feature selection)
- Automated model selection
- Hyperparameter optimization
- Automated analysis of results

Option 1: Use previous knowledge

- Select parameters as seen in previous papers
- Manual tuning
- Graduate student search

Not really efficient...

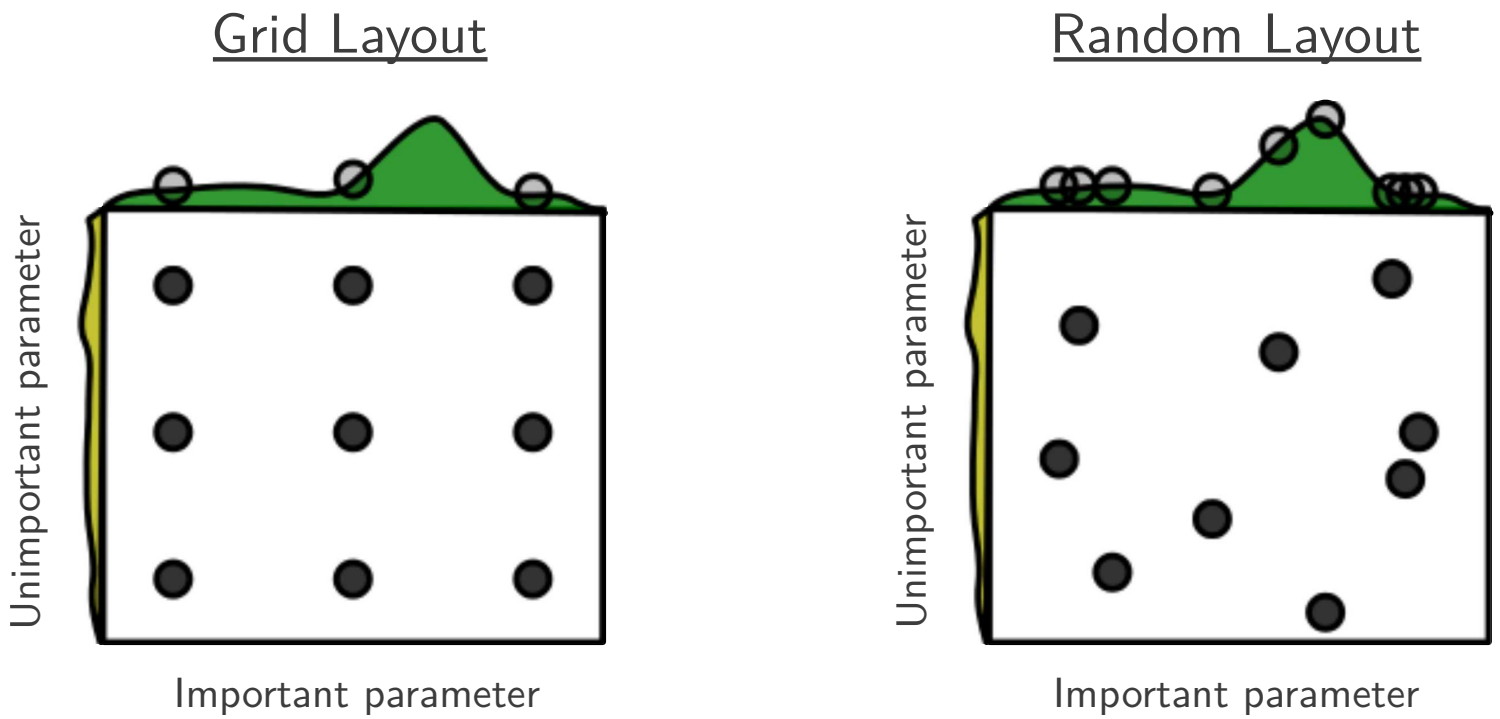
Option 2: Grid search



Bergstra and Bengio, 2012, Random Search for Hyper-Parameter Optimization

Scales exponentially with the number of dimensions
How do you adapt the grid to the function smoothness?

Option 2: Random search

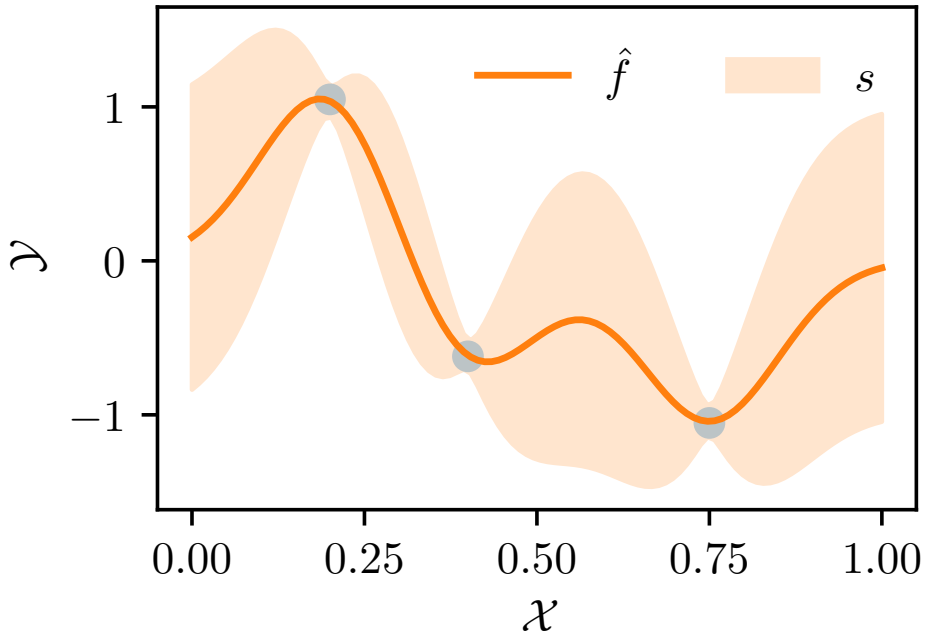


Bergstra and Bengio, 2012, Random Search for Hyper-Parameter Optimization

Better than grid search, but still expensive to guarantee good coverage

Another option: Bayesian optimization

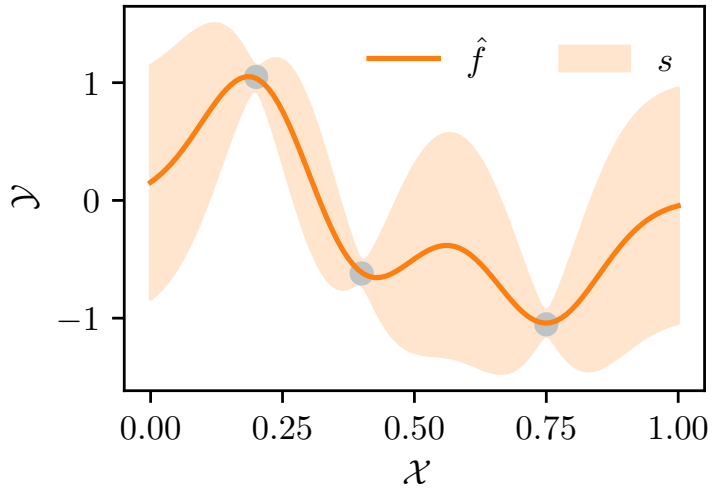
- Sequentially gather observations at different locations in the input space
- Exploit the underlying structure of the input space $\Rightarrow$ Kernel regression
- Bayesian view: Model expectation and uncertainty $\Rightarrow$ Gaussian Processes
- Once the budget is over, recommend the *best* location



Recall: Gaussian Processes

- Distribution over functions
- Tractable Bayesian modeling of functions even with infinite basis funcs.
- Input space (the search space) $\mathcal{X}$
- Kernel function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$

Every finite set of N points $\{x_1, \ldots, x_N\}$ induces a N-dimensional Gaussian distribution, which is the posterior distribution on $\{f(x_1), \ldots, f(x_N)\}$



Recall: Gaussian Processes posterior/prediction

- Consider noisy observations $y = f(x) + \epsilon = \phi(x)^\top \theta_\star + \epsilon$
- With Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$
- With Gaussian prior on parameters $\theta_\star \sim \mathcal{N}_d(0, \mathbf{I}_d)$

$$[\tilde{f}(x)]_{x \in \mathcal{X}} \sim \mathcal{N}_{|\mathcal{X}|}\left(\left[\hat{f}(x)\right]_{x \in \mathcal{X}}, [k_t(x, x')]_{x, x' \in \mathcal{X}} \right)$$

With

$$\hat{f}_t(x) = \mathbf{k}(x)^\top (\mathbf{K} + \sigma^2 \mathbf{I}_t)^{-1} \mathbf{y}$$

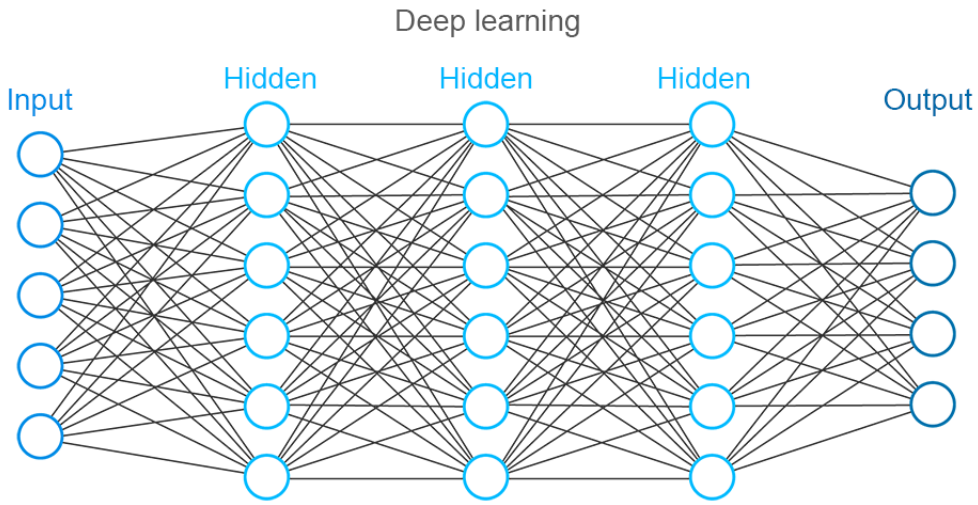
$$k_t(x, x') = k(x, x') - \mathbf{k}(x)^\top \left(\mathbf{K} + \sigma^2 \mathbf{I}_t\right)^{-1} \mathbf{k}(x') \qquad \text{and}$$

$$s_t^2(x) = k_t(x, x)$$

Historical overview

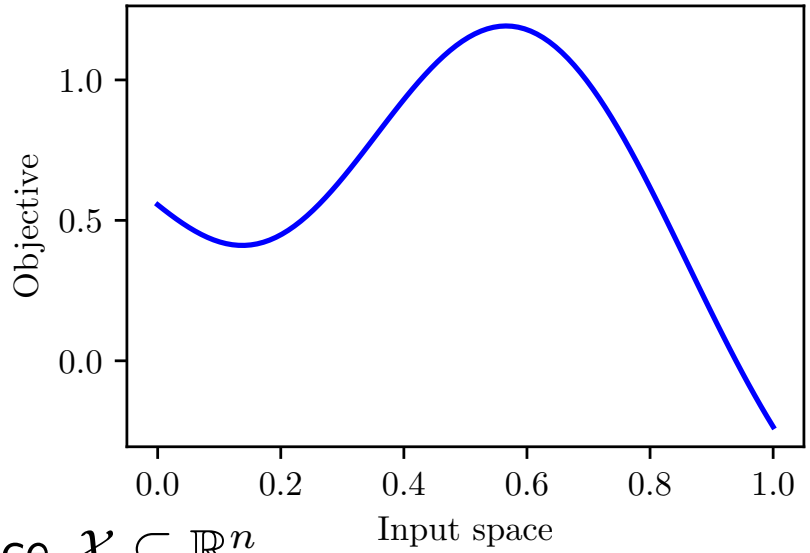
- Optimal design of experiments (Kirstine Smith, 1918)
- Response surface methods (Box and Wilson, 1951)
- Bayesian optimization (Kushner, 1964, then Mockus, 1978)
- Boost of attention in ML starting 2007

$\rightarrow$ It can be used to tuned ML hyperparameters!



From: altexsoft.com

Budgeted function optimization setting



- ~~Action~~ Input space $\mathcal{X} \subseteq \mathbb{R}^n$
- ~~Reward~~ Objective function $f : \mathcal{X} \mapsto \mathbb{R}$
- For each round $t = 1 \dots T$:
  1. Select an ~~action~~ input location $x_t \in \mathcal{X}$
  2. Observe ~~reward~~ output $y_t = f(x_t) + \epsilon_t$ $\leftarrow$ Observation noise ϵ_t
- After T rounds: recommend solution $\tilde{x}_T$

Goal: Minimize $|f(\tilde{x}_T) - f(x_\star)| \rightarrow$ recommend $x_\star = \arg\max_{x \in \mathcal{X}} f(x)$

Simple regret

Minimize simple regret: $\tilde{R}_T = |f(\tilde{x}_T) - f(x_\star)|$

- You are committing to a single solution $\tilde{x}_T$
- If $\tilde{x}_T$ is bad, you'll do bad *forever*
- Examples:
  - If you select *poor* hyperparameters for your algorithm
  - If you select a *poor* GUI for your website

Given some previously sampled locations $x_1, \ldots, x_t$ with observations $y_1, \ldots, y_t$, at which location x_{t+1} should I get my next sample?
$\Rightarrow$ With the intent of minimizing $\tilde{R}_{t+1}$

Acquisition function

- Input space $\mathcal{X} \subseteq \mathbb{R}^n$
- Objective function $f : \mathcal{X} \mapsto \mathbb{R}$
- For each round $t = 1 \dots T$:
  1. Select input location $x_t \in \mathcal{X}$ that maximizes an acquisition function
  2. Observe output $y_t = f(x_t) + \epsilon_t$ $\quad \leftarrow$ Observation noise ϵ_t
- After T rounds: recommend solution $\tilde{x}_T$

Goal: Minimize $|f(\tilde{x}_T) - f(x_\star)| \rightarrow$ recommend $x_\star = \arg\max_{x \in \mathcal{X}} f(x)$

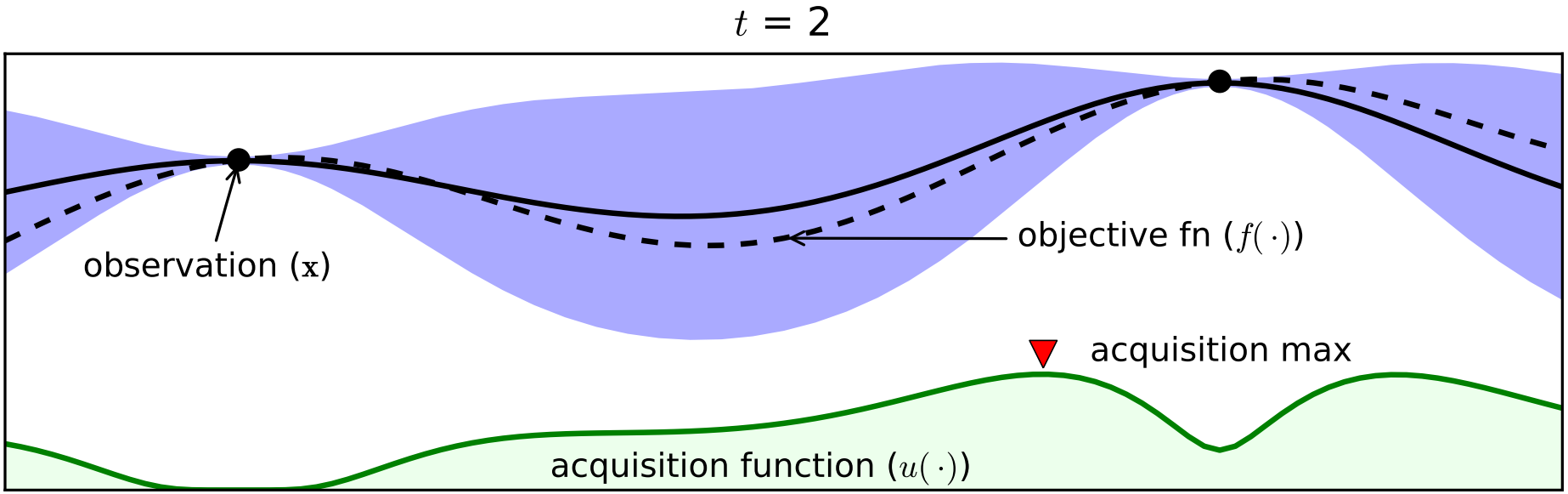
Exploration/Exploitation tradeoff under pure exploration

Acquisition function should be:

- high for points we expect to be better than what we know
- high for points we're uncertain about
- low for points we know

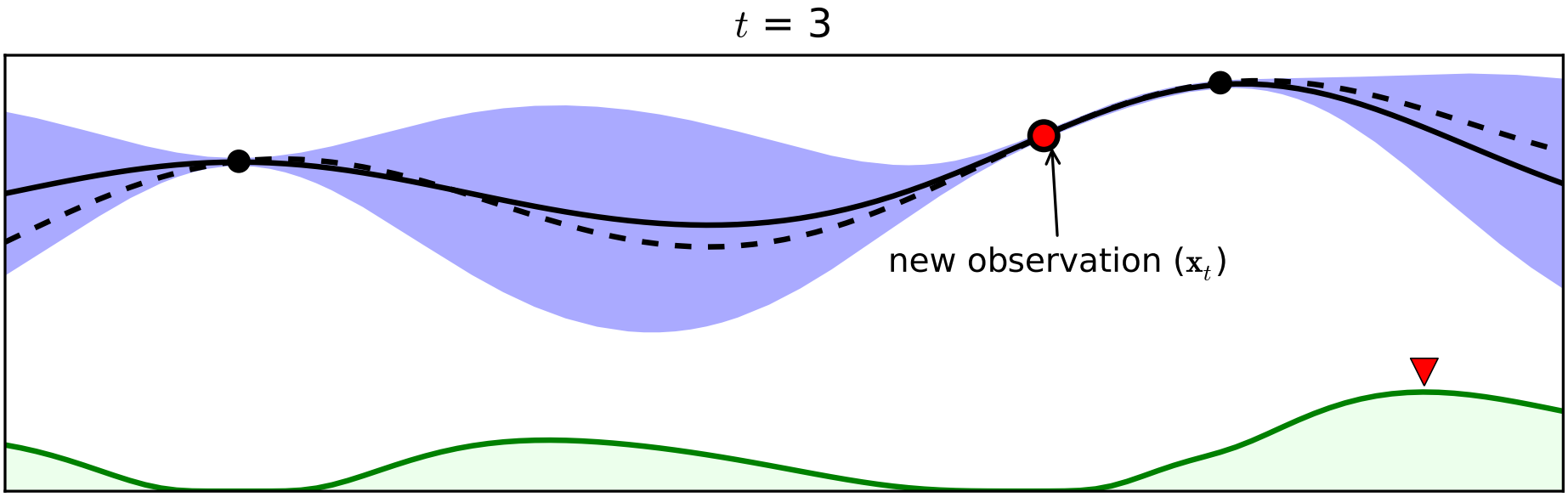
How different is it from the exploration/exploitation tradeoff when minimizing regret?

Acquisition function in motion



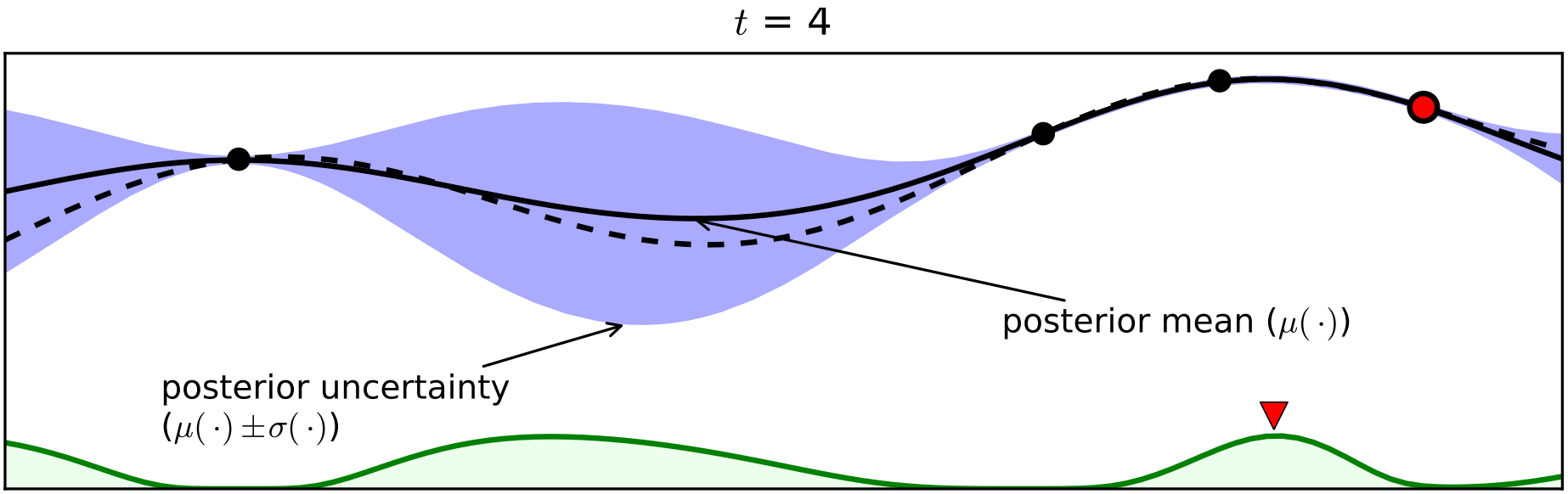
Brochu et al., 2010, A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning

Acquisition function in motion



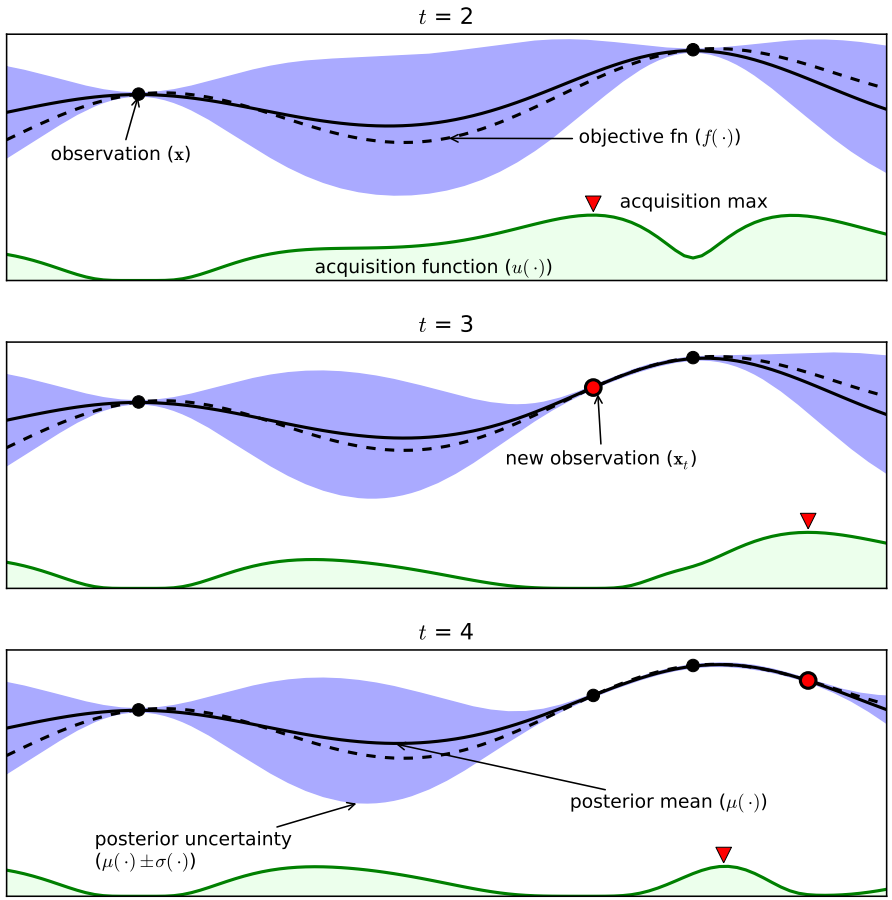
Brochu et al., 2010, A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning

Acquisition function in motion



Brochu et al., 2010, A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning

Acquisition function in motion - Overview



Brochu et al., 2010, A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning

Probability of Improvement (PI)

- Consider that we are searching for $x_\star = \arg\max_{x \in \mathcal{X}} f(x)$
- Noise-free setting
- Let $f^+_{t-1} = \arg\max_{x \in \{x_1, \ldots, x_{t-1}\}} f(x)$
- Probability of improvement at location x: $\mathrm{PI}(x) = \Pr[f(x) \geq f^+_{t-1}]$

Does not consider the magnitude of improvement

Expected Improvement (EI)

- Consider that we are searching for $x_\star = \arg\max_{x \in \mathcal{X}} f(x)$
- Noise-free setting
- Let $f_{t-1}^+ = \arg\max_{x \in \{x_1, \ldots, x_{t-1}\}} f(x)$
- Improvement at location x: $I_t(x) = \max\left(f(x) - f_{t-1}^+, 0\right)$
- Expected improvement:

$$\text{Select} \quad x_t = \arg\max_{x \in \mathcal{X}} \mathbb{E}[I_t(x) | x_1, y_1, \ldots, x_{t-1}, y_{t-1}]$$

Computing EI

- Recall: Noise-free setting
- Likelihood of improvement I on a normal posterior distribution parameterized by $\hat{f}(x)$ and $s^2(x)$:

$$\frac{1}{\sqrt{2\pi}s_{t-1}(x)} \exp\left(-\frac{(\hat{f}_{t-1}(x) - f^+_{t-1} - I)^2}{2s^2_{t-1}(x)}\right)$$

$$\begin{aligned}\mathbb{E}[I] &= \int_0^\infty I \frac{1}{\sqrt{2\pi}s(x)} \exp\left(-\frac{(\hat{f}(x) - f^+ - I)^2}{2s^2(x)}\right) dI \\ &= s(x)\left[\frac{\hat{f}(x) - f^+}{s(x)}\Phi\left(\frac{\hat{f}(x) - f^+}{s(x)}\right) + \phi\left(\frac{\hat{f}(x) - f^+}{s(x)}\right)\right]\end{aligned}$$

$\rightarrow \Phi/\phi$ are the CDF/PDF of a standard normal distribution

Computing EI in practice

- Observations are noisy $\rightarrow$ we don't have f^+_{t-1} directly
- Use $\hat{f}^+_{t-1} = \arg\max_{x \in \{x_1,\ldots,x_{t-1}\}} \hat{f}_{t-1}(x)$
- Compensate uncertainty:

$$Z_t(x) = \begin{cases} \frac{\hat{f}(x) - \hat{f}^+ - \xi}{s(x)} & \text{if } s(x) > 0 \\ 0 & \text{if } s(x) = 0 \end{cases}$$

$$\mathbb{E}[I_t(x)] = \begin{cases} \left(\hat{f}(x) - \hat{f}^+ - \xi\right) \Phi\left(Z_t(x)\right) + s(x)\phi\left(Z_t(x)\right) & \text{if } s(x) > 0 \\ 0 & \text{if } s(x) = 0 \end{cases}$$

- In practice, $\xi = 0.01$ (scaled by noise variance if necessary) works well

EI example

```
import numpy
from scipy.stats import norm

def expected_improvement(X, X_sample, gpr, xi):
    f_hat, std_hat = gpr.predict(X, return_std=True)
    f_hat_sample = gpr.predict(X_sample)

    s_hat = s_hat.reshape(-1, X_sample.shape[1])

    f_opt = numpy.max(f_hat_sample)

    improvement = f_hat - f_opt - xi
    Z = improvement / s_hat
    ei = improvement * norm.cdf(Z) + s_hat * norm.pdf(Z)
    ei[s_hat == 0.0] = 0.0
    return ei
```

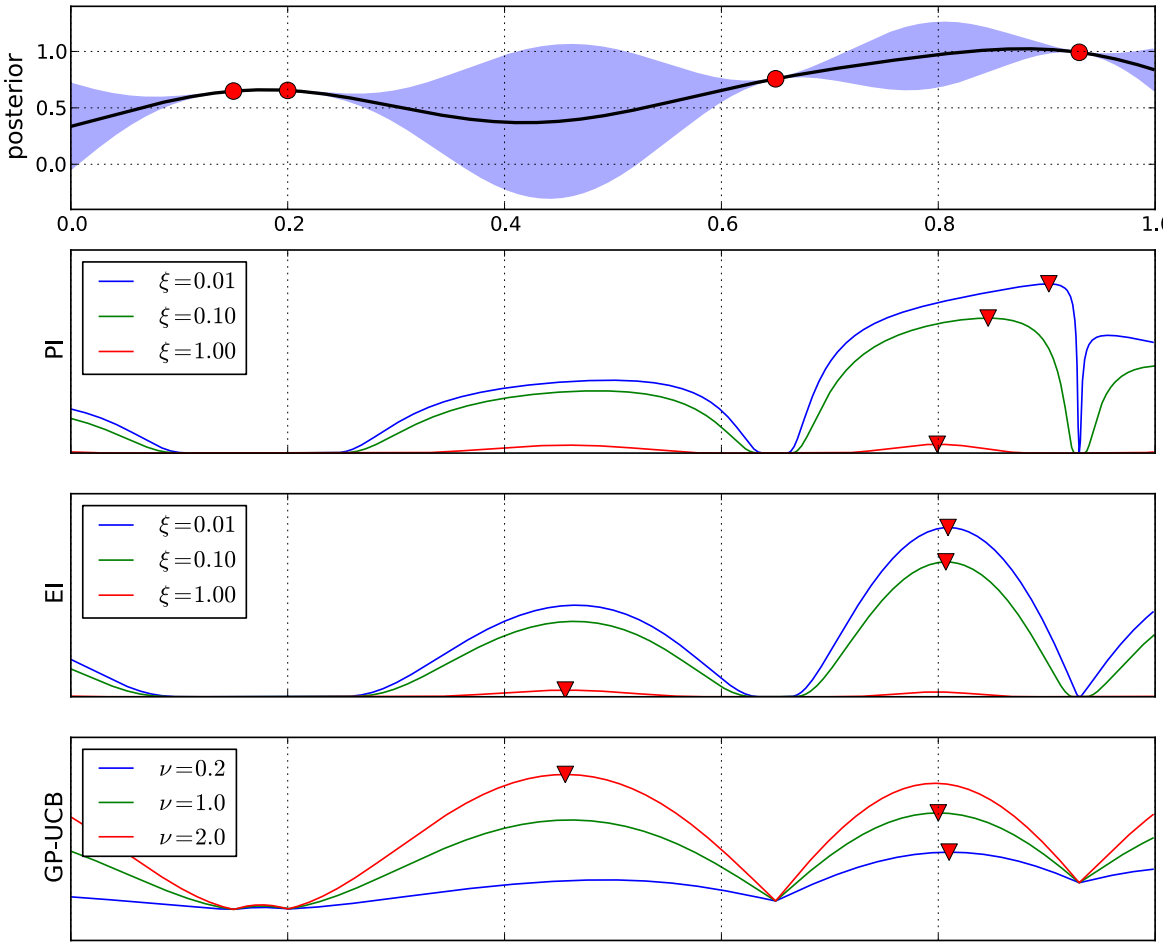
Kernel/GP-UCB

$$\mathrm{UCB}_t(x) = \hat{f}_{t-1}(x) + \sqrt{\nu\tau_t}s_{t-1}(x)$$

- Select $x_t = \arg\max_{x\in\mathcal{X}} \mathrm{UCB}_t(x)$
- Designed to optimize an exploration/exploitation tradeoff
- Designed to minimize regret with τ_t of order $\mathcal{O}(t)$

Acquisition function in Bayesian optimization vs policy in bandits?

Acquisition function comparison



Brochu et al., 2010, A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning

Summary

- Disinguish regret from simple regret (pure exploration)
- Bayesian optimization has many applications, including in ML!
- In practice it is not clear what acquisition function to use
- Bayesian optimization raises an exploration/exploitation challenge with a different aim from bandit optimization
- What if improvement is costly?
  Maximize expected improvement per second
  Learn the time function

Any challenges remaining?

Choosing the covariance function

- Kernel function $k : \mathcal{X} \times \mathcal{X} \mapsto \mathbb{R}$ is also called covariance function
- Space of functions covered by the GP depend on k (RKHS)
- Squared exponential (Gaussian) kernel:

$$\text{Gaussian:} \quad k_{\text{SE}}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2} r^2(\mathbf{x}, \mathbf{x}')\right)$$

$$\text{with} \quad r^2(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} (x_i - x_i')^2 / \rho_i^2$$

- Isotropic if $\rho_i = \rho$ for every dimension i, otherwise anisotropic

Functions defined by Gaussian kernel are often *too smooth*

Matérn covariance

- Explicit the smoothness: ν
- Γ: Gamma function
- K_ν: Bessel function of the second kind
- Matérn kernel:

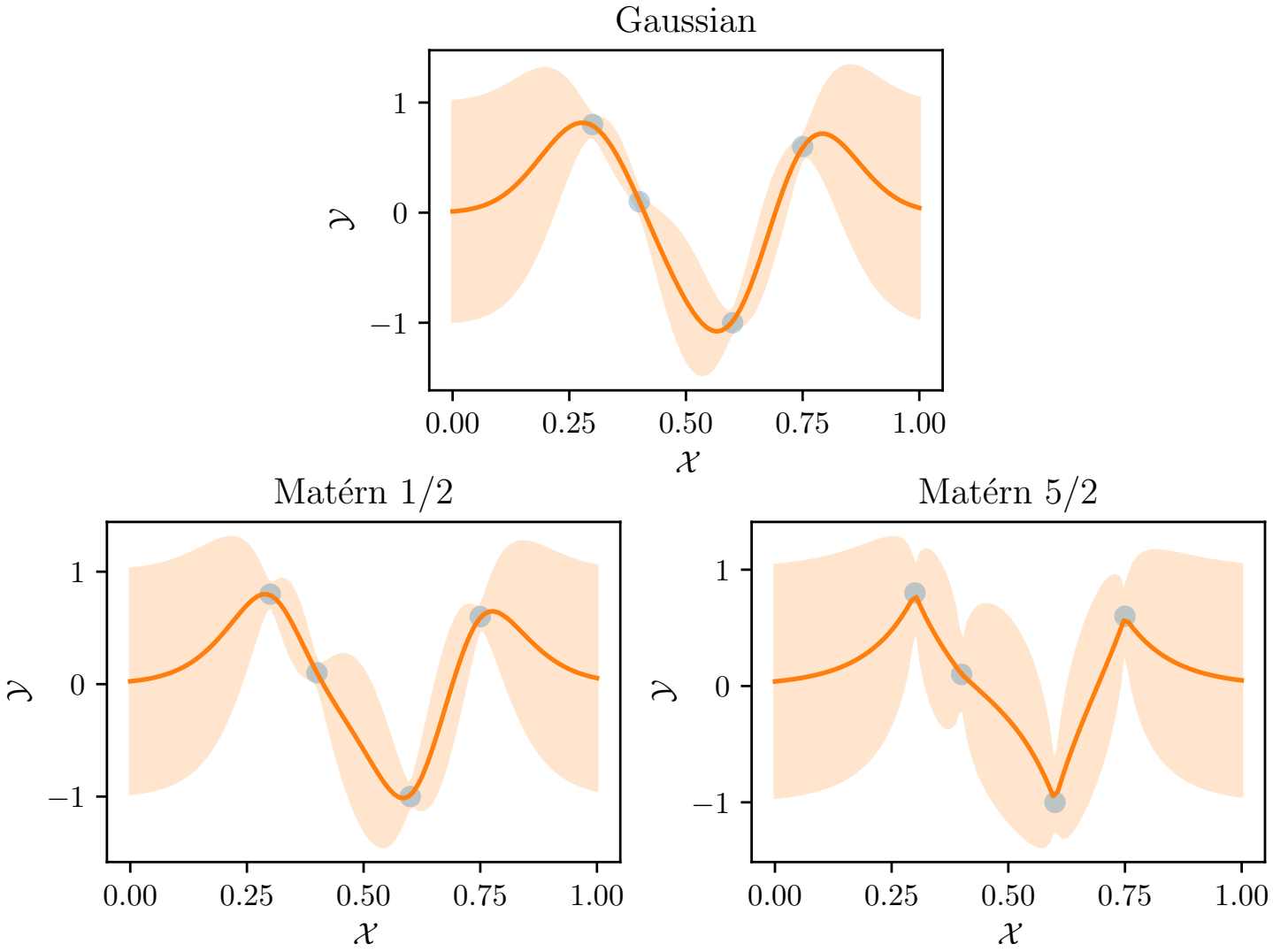
$$k_\nu(\mathbf{x}, \mathbf{x}') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu r^2(\mathbf{x}, \mathbf{x}')} \right)^\nu K_\nu \left(\sqrt{2\nu r^2(\mathbf{x}, \mathbf{x}')} \right)$$

$$k_{5/2}(\mathbf{x}, \mathbf{x}') = \left(1 + \sqrt{5r^2(\mathbf{x}, \mathbf{x}')} + \frac{5}{3} r^2(\mathbf{x}, \mathbf{x}') \right) \exp\left(-\sqrt{5r^2(\mathbf{x}, \mathbf{x}')} \right)$$

$$\text{with} \quad r^2(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{n} (x_i - x_i')^2 / \rho_i^2$$

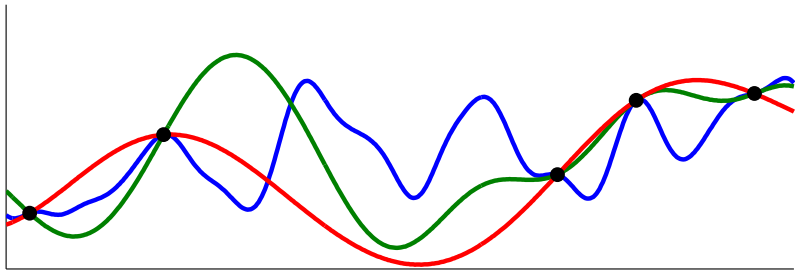
- When $\nu \to \infty$, Matérn kernel converges to Gaussian kernel

Covariance comparison: Posterior

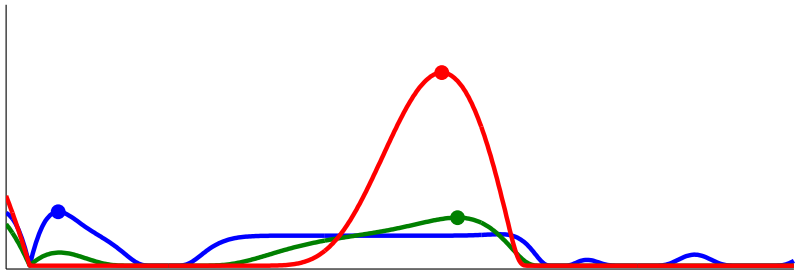


Covariance hyperparameters

- The GP has its own hyperparameters to tune!
- GP hypers influence the posterior $\rightarrow$ the acquisition function



(a) Posterior samples under varying hyperparameters



(b) Expected improvement under varying hyperparameters

Snoek et al., 2012, Practical Bayesian Optimization of Machine Learning Algorithms

Selecting covariance hyperparameters

- Optimize the marginal likelihood under the Gaussian process:

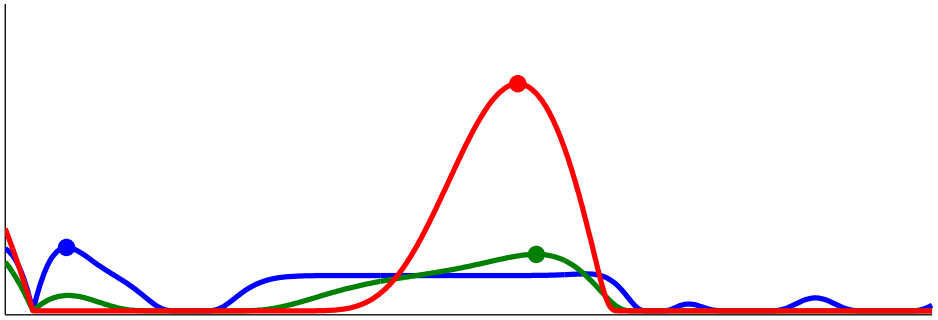
$$\log \Pr[\mathbf{y}_{1..t}|\mathbf{x}_{1..t}, \mu_0, \sigma, \boldsymbol{\rho}] = -\frac{1}{2}(\mathbf{y}_{1..t} - \mu_0)^\top (\mathbf{K}_{\boldsymbol{\rho}} + \sigma \mathbf{I}_t)^{-1}(\mathbf{y}_{1..t} - \mu_0) - \frac{1}{2}\log|\mathbf{K}_{\boldsymbol{\rho}} + \sigma \mathbf{I}_t| - \frac{t}{2}\log 2\pi$$

- Fully-Bayesian: integrated acquisition function:

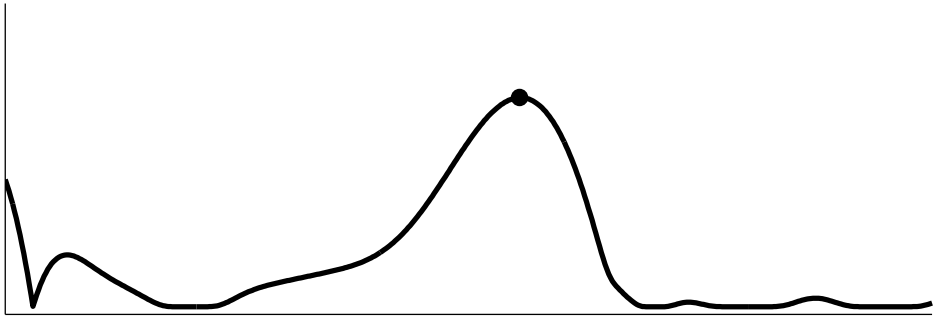
$$\int a(\mathbf{x}; \mathbf{x}_{1..t}, \mathbf{y}_{1..t}, \mu_0, \sigma, \boldsymbol{\rho}) p(\mu_0, \sigma, \boldsymbol{\rho}|\mathbf{x}_{1..t}, \mathbf{y}_{1..t}) d\mu_0 d\sigma d\boldsymbol{\rho}$$

$\rightarrow$ Can be approximated with Monte Carlo

Integrated EI



(b) Expected improvement under varying hyperparameters

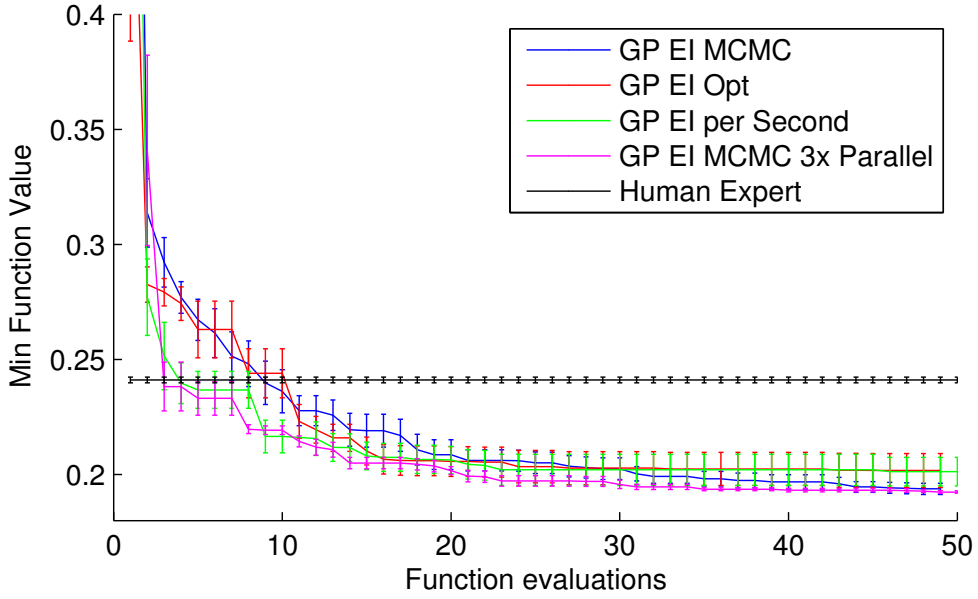


(c) Integrated expected improvement

Snoek et al., 2012, Practical Bayesian Optimization of Machine Learning Algorithms

Examples of results

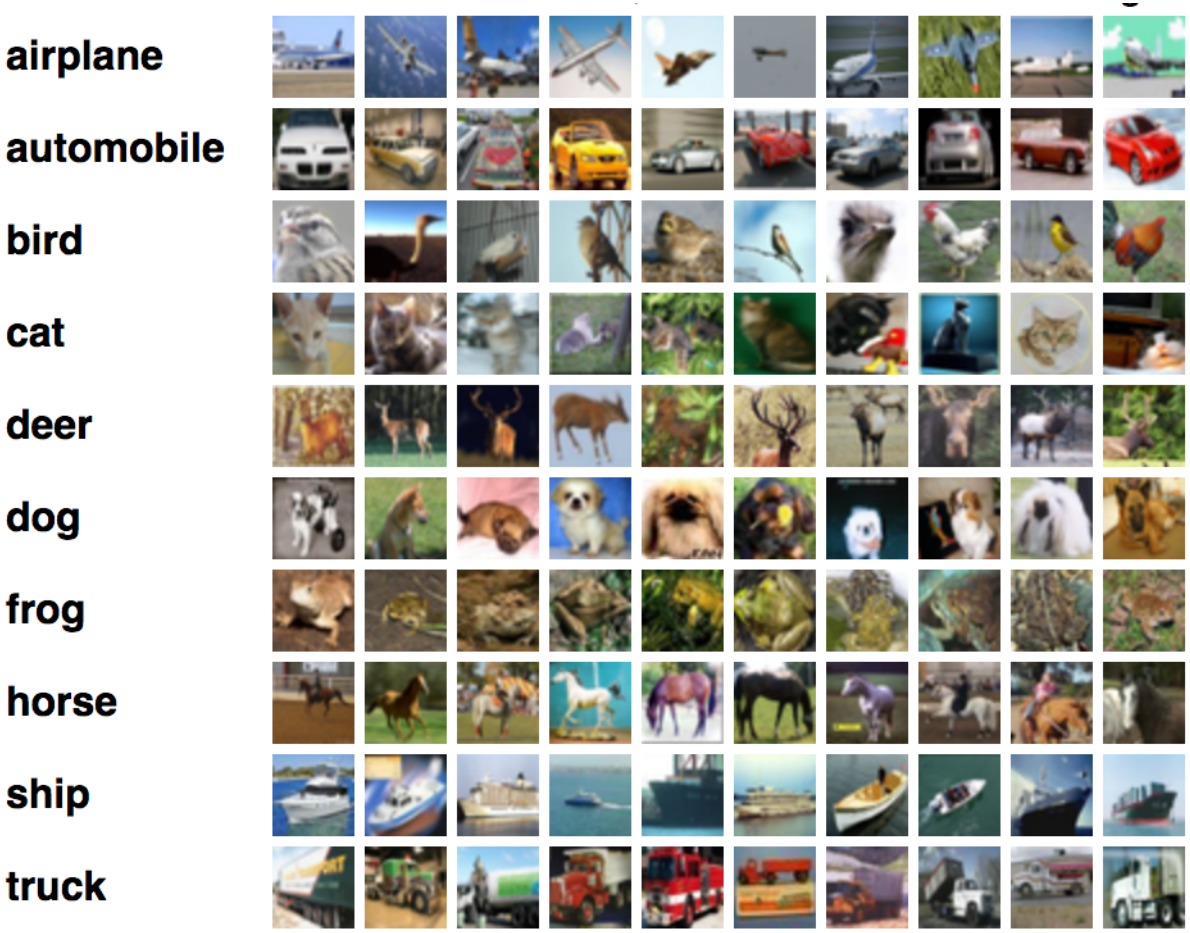
- Optimizing hyperparameters (layer-specific learning rates, weight decay, and a few other parameters) for a CNN on CIFAR-10
- Each function evaluation takes ~ 1 hour
- Human expert = Alex Krizhevsky (creator of AlexNet)



Snoek et al., 2012, Practical Bayesian Optimization of Machine Learning Algorithms

CIFAR-10

60,000 32×32 color images in 10 different classes



Additional challenges

- How to optimize the acquisition function?
- Parallel evaluations
  - Enforce diversity
  - Parallelised Bayesian Optimisation via Thompson Sampling (Kandasamy et al., 2018)
- Scaling up? Bayesian Deep Learning
  - Prior on neural network weights
  - Given training data, compute posterior on weights

  $\rightarrow$ Obtain posterior distribution on target functions

Some ressources

- Black-box Bayesian optimization using Spearmint:
  `https://github.com/JasperSnoek/spearmint`
- Practical Bayesian optimization of machine learning algorithms
  (Snoek, Larochelle, and Adams. 2012)
  `http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms`
- Scalable Bayesian optimization using deep neural networks
  (Snoek et al. 2015)
  `http://www.jmlr.org/proceedings/papers/v37/snoek15.pdf`