Bayesian Optimisation

Mark van der Wilk

Department of Computing
Imperial College London

@markvanderwilk
m.vdwilk@imperial.ac.uk

February 17, 2022

Optimisation of expensive cost functions

In many situations, we want to optimise processes that we do not understand. They may be:

- Take long or costly to evaluate
  (physical experiments, or long-running computer simulations)
- Black-box
  (we have little a-priori knowledge about them)
- Impossible to evaluate gradients of
  (can't back-propagate through reality)

Black-box cost functions

- Design of turbine blades
- Biological molecules
- Optimise chemical processes
- **Selecting Machine Learning hyperparameters**

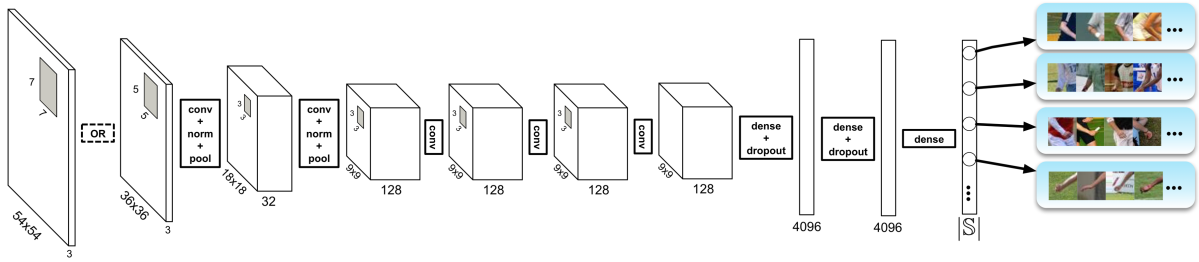
Machine Learning Meta-Challenges

- Machine learning models are getting more and more complicated
  - ⏩ Usually more parameters (e.g., deep neural networks)
- Non-convex and stochastic optimization methods have meta-parameters that are difficult to tune (learning rates, momentum parameters, ...)

⏩ Generally hard to apply modern techniques or reproduce results

Goal: Automate the selection of critical meta-parameters
(see also: Automated Machine Learning (AutoML))

Example: Deep Neural Networks



Huge interest in large neural networks

- When well-tuned, very successful for visual object identification, speech recognition, computational biology, ...
- Huge investments by Google, Facebook, Microsoft, etc.
- Many choices: number of layers, weight regularization, layer size, which nonlinearity, batch size, learning rate schedule, stopping conditions

Example: Online Latent Dirichlet Allocation

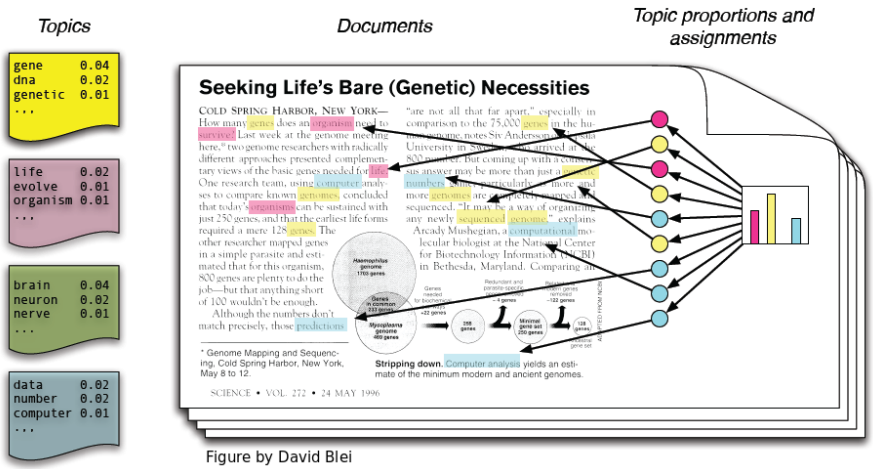
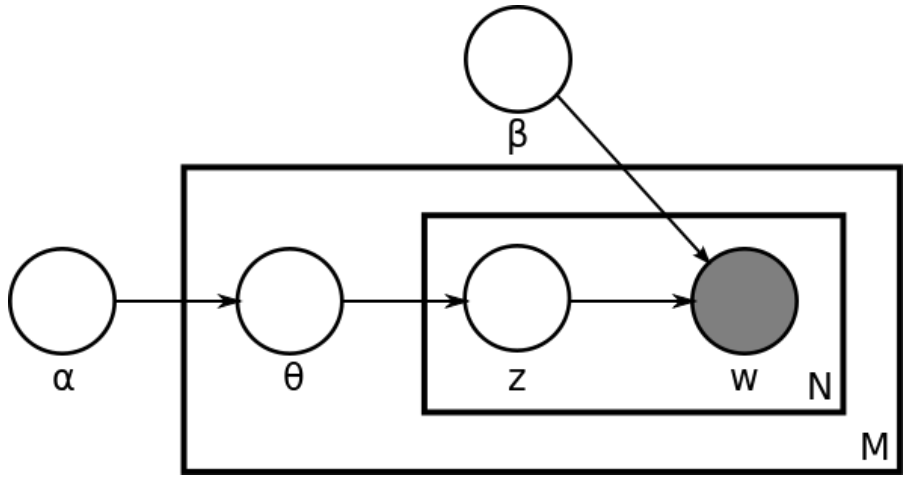
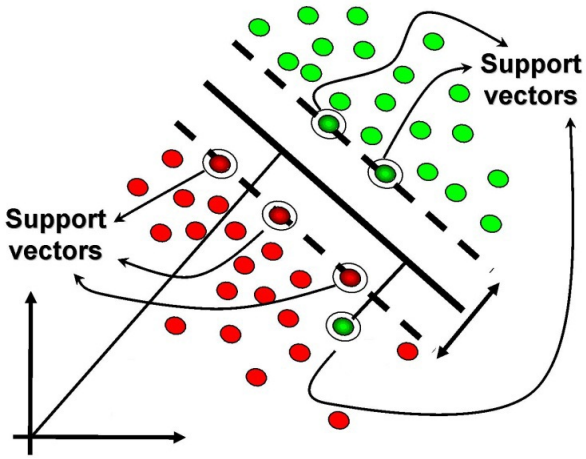


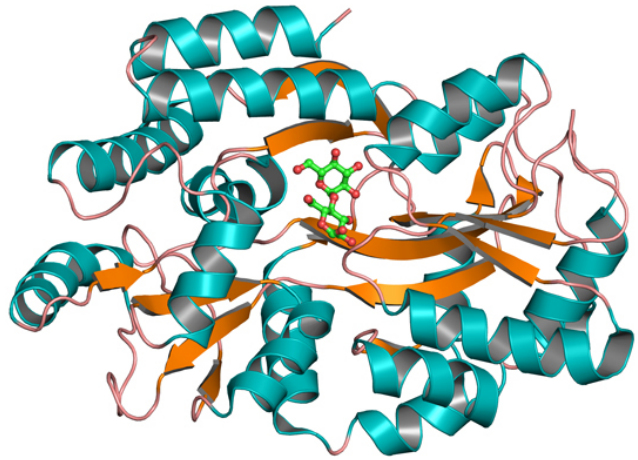
Figure by David Blei



- Hoffman et al. (2010): Approximate inference for large-scale text analysis (topic modeling) with Latent Dirichlet Allocation
- Good empirical results when well tuned
- Hyper-parameters tricky to set: Dirichlet parameters, number of topics, learning rate schedule, batch size, vocabulary size, ...

Example: Classification of DNA Sequences





- Objective: Predict which DNA sequences will bind with which proteins
- Miller et al. (2012): Latent Structural Support Vector Machine
- Hyper-parameters: margin/slack parameter, entropy parameter, convergence criterion

Search for Good Hyper-parameters

- Define an objective function to evaluate the quality of the hyper-parameters
  - Usually, we care about generalization performance
  - Cross validation to measure parameter quality
- Standard search procedures:
  - Manual tuning
  - Grid search
  - Random search (very simple, works surprisingly well)
  - Black magic
- Painful:
  - Evaluating the quality of the objective may be very expensive (e.g., time or money)
    ⏩ Imagine we would need to run a GPU/TPU cluster for 2 weeks
  - Many training cycles
  - Possibly noisy

Alternative Approach: Bayesian Optimization

Setting

Globally optimize a black-box objective that is expensive to evaluate (e.g., cross-validation error for a massive neural network)

We want to be smart about using the evaluations we have

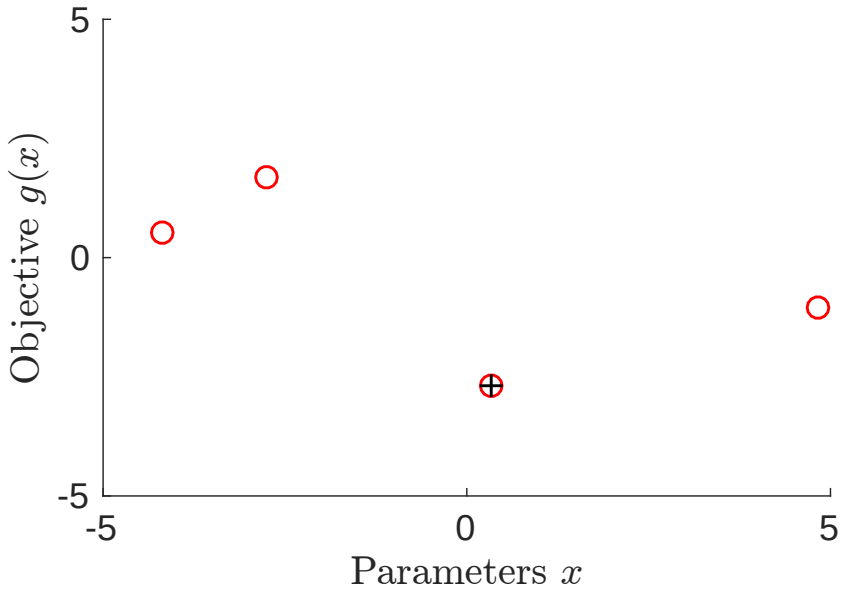
- Build a probabilistic proxy model for the objective using outcomes of past experiments as training data
- We can predict the outcome of the next experiment using the proxy model, which is much cheaper to evaluate
- Optimize cheap proxy function to determine where to evaluate the true objective next
- Standard proxy: Gaussian process

Setting (2)

- Objective: Find global minimum of objective function f:

$$x_* = \arg\min_x f(x)$$

- We can evaluate the objective g pointwise, but do not have an easy functional form or gradients; observations may be noisy
- Evaluating f is costly (e.g., train a massive deep network)



Bayesian optimisation

Let's phrase BayesOpt using decision theory:

$$L(f, \{\mathbf{x}_n\}_{n=1}^N) = Nc + \min_{1 \leqslant n \leqslant N} f(\mathbf{x}_n) \tag{1}$$

(Optimisers are pessimists, they talk about loss instead of utility)

- We decide how many evaluations we want to do N
- We decide where to evaluate $f(\cdot)$
- We incur a cost c for each evaluation
- We reduce loss by finding a lower $f(\mathbf{x})$

Bayesian optimisation – decision theory

For 3 observations:

$$\{\mathbf{x}_1^*, \mathbf{x}_2^*(\mathcal{D}_1), \mathbf{x}_3^*(\mathcal{D}_2)\} = \underset{\{\mathbf{x}_n\}_{n=1}^3}{\operatorname{argmin}} \mathbb{E}_{p(f)}\left[Nc + \min_{1 \leqslant n \leqslant N} f(\mathbf{x}_n)\right] \tag{2}$$

Remember: actions $\mathbf{x}_2$ and $\mathbf{x}_3$ can depend on the data we observe. The data are observations of the GP $\mathcal{D} = \{\mathbf{x}_n, f(\mathbf{x}_n)\}_{n=1}^N$.

$$\mathbf{x}_1^* = -Nc + \underset{\mathbf{x}_1}{\operatorname{argmin}} \mathbb{E}_f\left[\min_{\mathbf{x}_2} \mathbb{E}_{f \mid \mathcal{D}_1}\left[\min_{\mathbf{x}_3} \mathbb{E}_{f \mid \mathcal{D}_2}\left[\min_{1 \leqslant n \leqslant 3}\{f(\mathbf{x}_n)\}\right]\right]\right] \tag{3}$$

Computational difficulties

$$\mathbf{x}_1^* = -Nc + \underset{\mathbf{x}_1}{\operatorname{argmin}} \mathbb{E}_f \left[\min_{\mathbf{x}_2} \mathbb{E}_{f \mid \mathcal{D}_1} \left[\min_{\mathbf{x}_3} \mathbb{E}_{f \mid \mathcal{D}_2} \left[\min_{1 \leqslant n \leqslant 3} \{ f(\mathbf{x}_n) \} \right] \right] \right]$$

This is **very** difficult to evaluate.

- Difficult optimisations: closed-form optimum cannot be found
- Expectations of non-closed-form expressions
- Optimisations of expectations of non-closed-form expressions

Alternative approaches:

- Can try to approximate these quantities (fun to try for small-scale examples)
- Heuristic approaches

How to design a heuristic

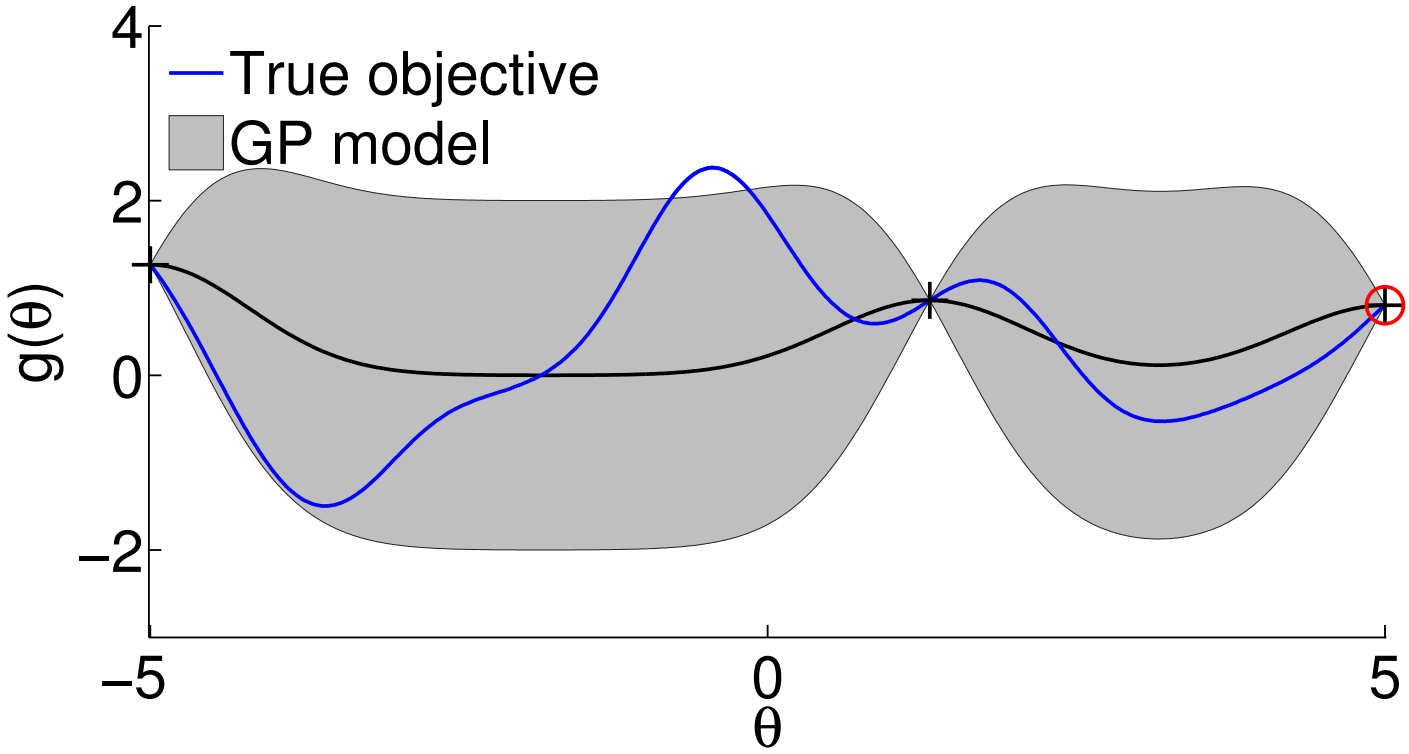
In BayesOpt, heuristic is called the **acquisition function**. Instead of maximising the utility (or minimising loss), we maximise the acquisition function.

- Computational constraint: Only use posterior of GP given **current** observations to choose current action
- Desired behaviour: Some sort of a trade-off between **exploration** and **exploitation**.
  - Want to try inputs that are different (exploration)
  - Want to focus on areas that are likely to have high values (exploitation)

Inadequate acquisition functions

Two inadequate acquisition functions:

- Minimise posterior mean (pure exploitation)
- Maximise posterior variance (pure exploration)



Common acquisition functions

- For all $\boldsymbol{x} \in \mathbb{R}^D$ the GP posterior gives a predictive mean $\mu(\boldsymbol{x})$ variance $\sigma^2(\boldsymbol{x})$ of $g(\boldsymbol{x})$
- Define

$$\gamma(\boldsymbol{x}) = \frac{f(\boldsymbol{x}_{\text{best}}) - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}$$

- **Probability of Improvement (Kushner 1964):**

$$\alpha_{\text{PI}}(\boldsymbol{x}) = \Phi(\gamma(\boldsymbol{x}))$$

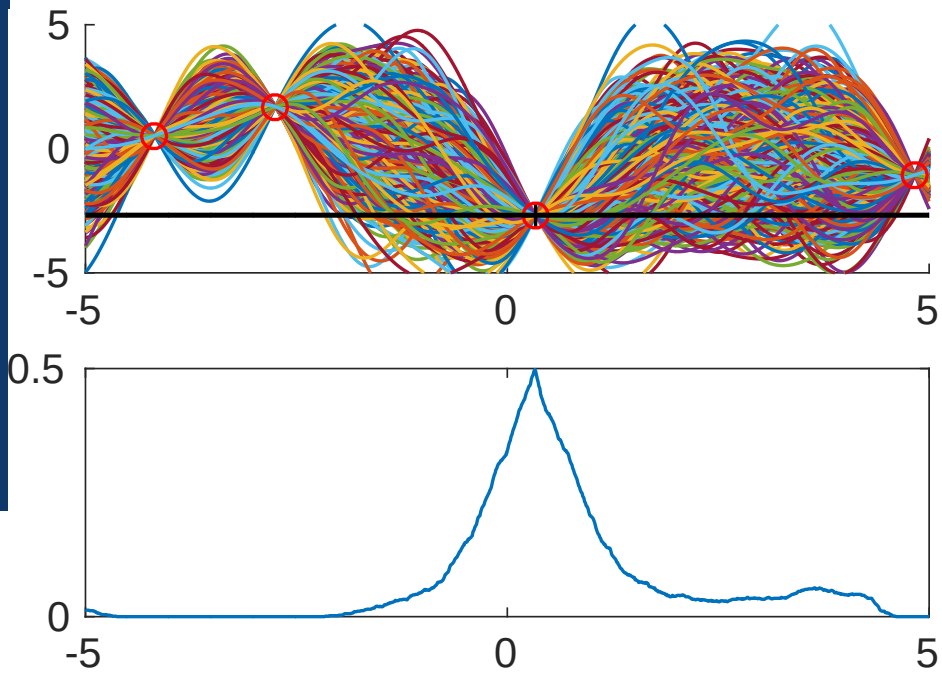
- **Expected Improvement (Mockus 1978):**

$$\alpha_{\text{EI}}(\boldsymbol{x}) = \sigma(\boldsymbol{x})\big(\gamma(\boldsymbol{x})\Phi(\gamma(\boldsymbol{x})) + \mathcal{N}\big(\gamma(\boldsymbol{x}) \,|\, 0, 1\big)\big)$$

- **GP Lower Confidence Bound (Srinivas et al., 2010):**

$$\alpha_{\text{LCB}}(\boldsymbol{x}) = -(\mu(\boldsymbol{x}) - \kappa\sigma(\boldsymbol{x}))\,, \quad \kappa > 0$$

Probability of Improvement (1)

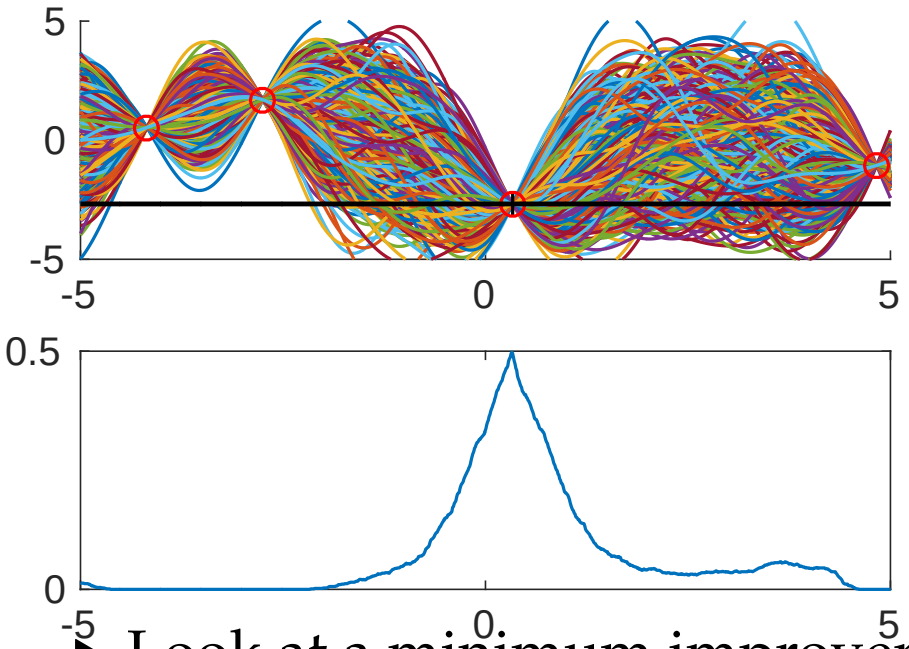


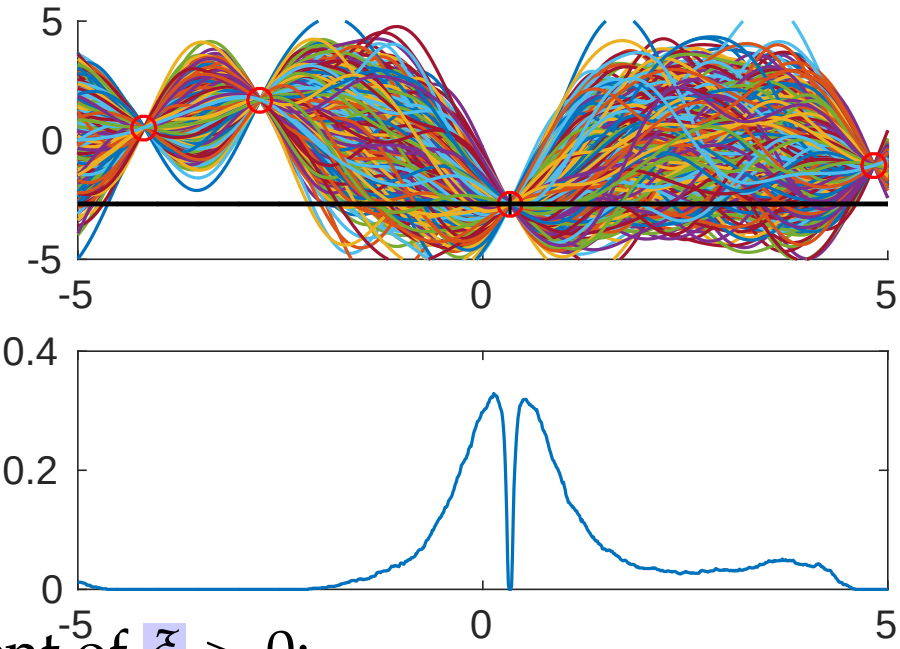
- Idea: Determine the probability that x_* leads to a better function value than the currently best one $f(x_{\text{best}})$
- Sampling-based setting: Sample N functions f_i; at every input x compute a Monte-Carlo estimate

$$\alpha_{\text{PI}}(x) = p(f(x) < f(x_{\text{best}})) \approx \frac{1}{N} \sum_{i=1}^{N} \delta\big(f_i(x) < f(x_{\text{best}})\big)$$

⏩ Can lead to continued exploitation in an ϵ-region around x_{best}.
⏩ Introduce a "slack variable" ξ for more aggressive exploration

Probability of Improvement (2)





- Look at a minimum improvement of $\xi > 0$:

$$\alpha_{\mathrm{PI}}(\boldsymbol{x}) = p(f(\boldsymbol{x}) < f(\boldsymbol{x}_{\mathrm{best}}) - \xi) \approx \frac{1}{N}\sum_{i=1}^{N} \delta\left(f_i(\boldsymbol{x}) < f(\boldsymbol{x}_{\mathrm{best}}) - \xi\right)$$

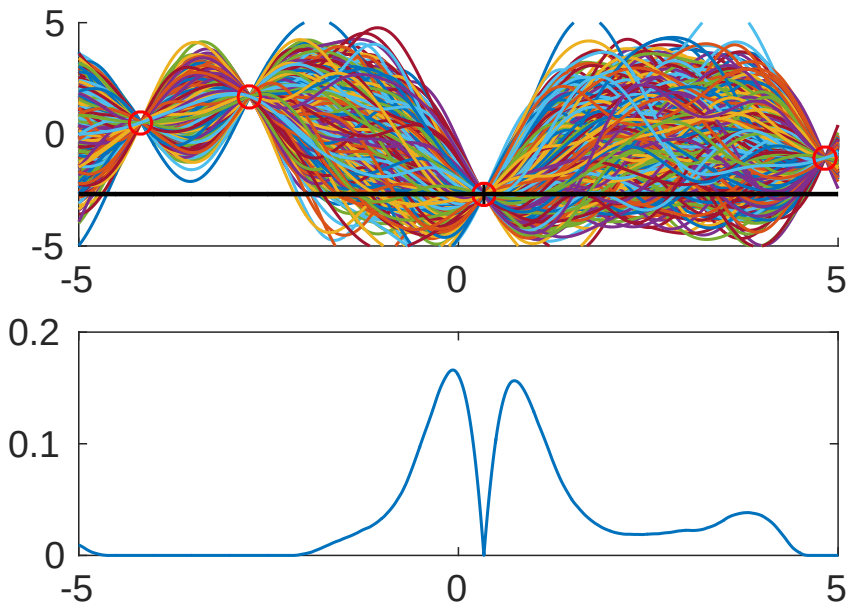
- If $f \sim GP$ and $p(f(\boldsymbol{x})) = \mathcal{N}\left(\mu(\boldsymbol{x}), \sigma(\boldsymbol{x})\right)$:

$$\alpha_{\mathrm{PI}}(\boldsymbol{x}) = \Phi(\gamma(\boldsymbol{x}, \xi))\,, \qquad \gamma(\boldsymbol{x}, \xi) = \frac{f(\boldsymbol{x}_{\mathrm{best}}) - \xi - \mu(\boldsymbol{x})}{\sigma(\boldsymbol{x})}$$

Expected Improvement

- Idea: Quantify the amount of improvement
- Sampling-based scenario, where $f_i \sim p(f)$:

$$\alpha_{\text{EI}}(\boldsymbol{x}) = \mathbb{E}[\max\{0, f(\boldsymbol{x}_{\text{best}}) - f(\boldsymbol{x})\}]$$
$$\approx \frac{1}{N}\sum_{i=1}^{N} \max\{0, f(\boldsymbol{x}_{\text{best}}) - f_i(\boldsymbol{x})\}$$

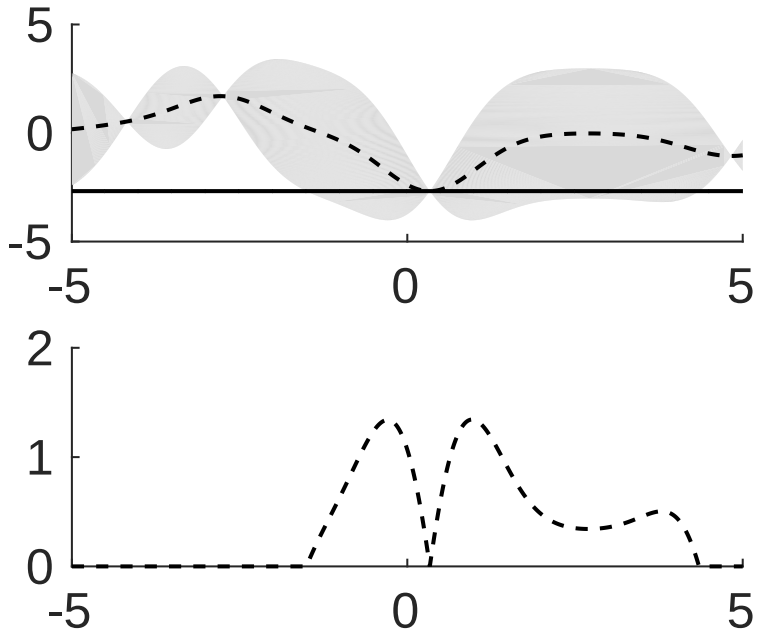


- If $f \sim GP$, we have a closed-form expression:

$$\alpha_{\text{EI}}(\boldsymbol{x}) = \sigma(\boldsymbol{x})\big(\gamma(\boldsymbol{x})\Phi(\gamma(\boldsymbol{x})) + \mathcal{N}\big(\gamma(\boldsymbol{x}) \,|\, 0, 1\big)\big)$$

- Slack-variable approach also possible (similar to PI)

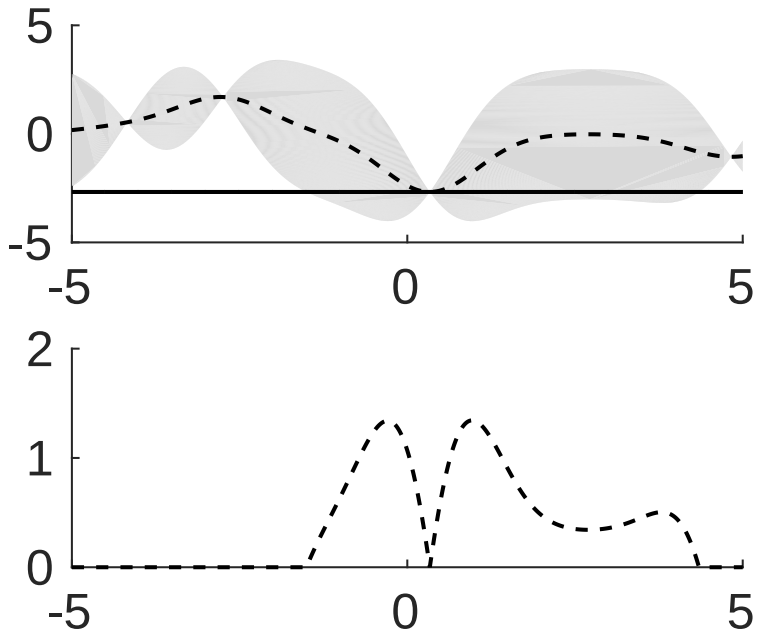
GP-Lower Confidence Bound (1)



- Use the predictive mean $\mu(\boldsymbol{x})$ and variance $\sigma^2(\boldsymbol{x})$ of the GP prediction directly for targeted exploration by means of the acquisition function

$$\alpha_{\text{LCB}}(\boldsymbol{x}_t) = -\left(\mu(\boldsymbol{x}_t) - \sqrt{\kappa}\sigma(\boldsymbol{x}_t)\right)$$

GP-Lower Confidence Bound (2)



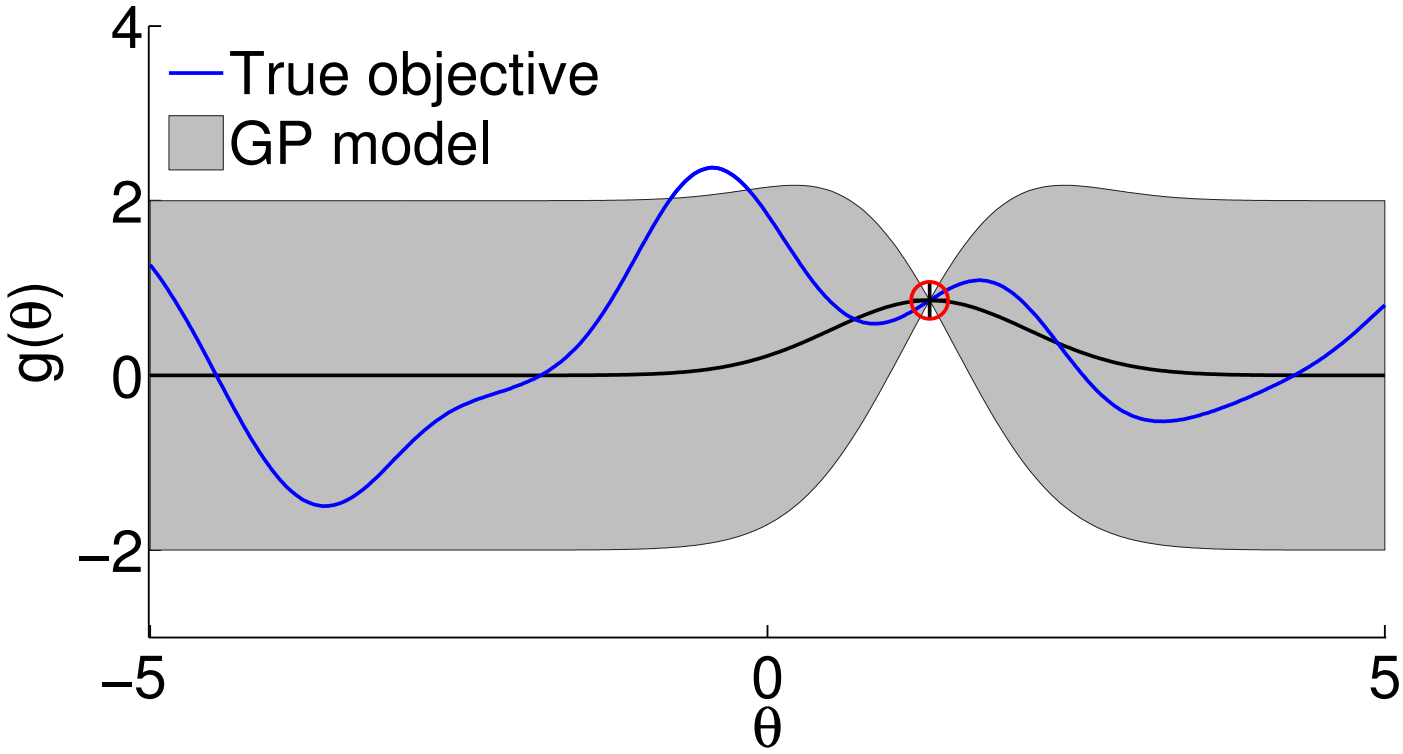
- More generally, we can get regret bounds for iteration-dependent κ (Srinivas et al., 2010)

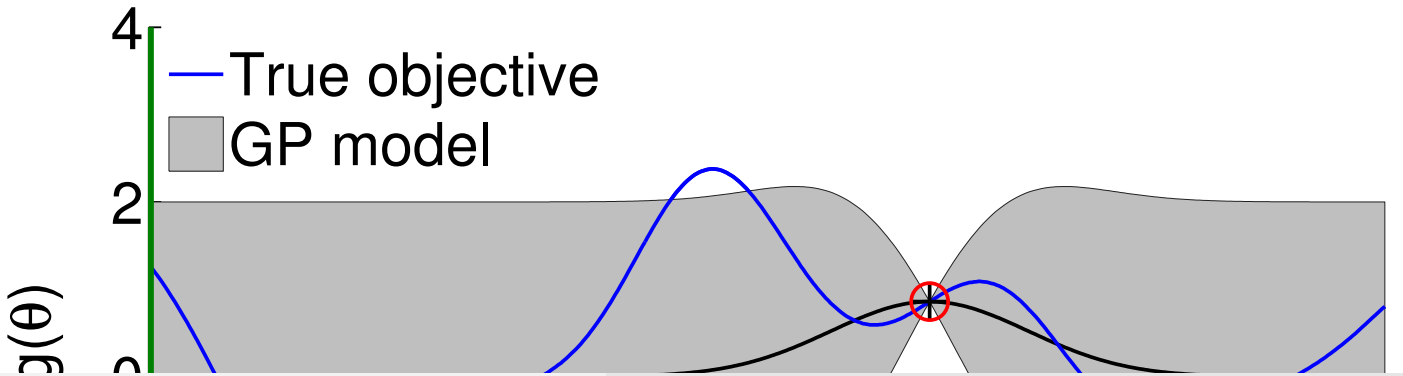
$$\alpha_{\text{LCB}}(\boldsymbol{x}_t) = -\left(\mu(\boldsymbol{x}_t) - \sqrt{\kappa_t}\sigma(\boldsymbol{x}_t)\right)$$

where $\kappa_t \in \mathcal{O}(\log t)$ grows with the iteration t

⏩ Continue exploration

Bayesian Optimization: Illustration





Optimising the Acquisition Function

Notice: Acquisition function is multi-modal

- Optimizing the acquisition function requires us to run a global optimizer inside Bayesian optimization
- What have we gained?
- Evaluating the acquisition function is cheap compared to evaluating the true objective
  - ⏩ We can afford evaluating it many times
  - ⏩ Apply the usual tricks of many random restarts

Key Steps (Pseudo-Code)

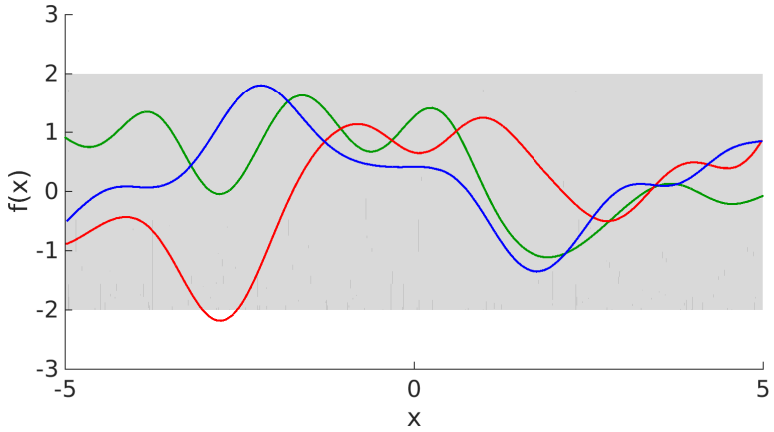
1: **Init:** Data set $\mathcal{D}_0 = \{\boldsymbol{X}_0, \boldsymbol{y}_0\}$ (can be empty)
2: **for** iterations $t = 1, 2, \ldots$ **do**
3: Update GP using data $\mathcal{D}_{t-1}$
4: Select $\boldsymbol{x}_t = \arg\max_{\boldsymbol{x}} \alpha(\boldsymbol{x})$ by optimising acquisition function
5: Query true objective g at $\boldsymbol{x}_t$
6: Augment data set $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(\boldsymbol{x}_t, y_t)\}$
7: **end for**
8: **Return** best input in data set: $\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} y(\boldsymbol{x})$

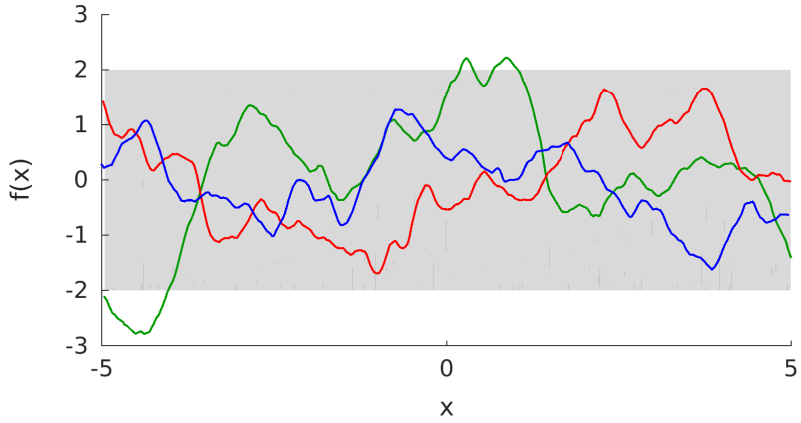
Optimising acquisition function is itself an iterative process, done with a numerical gradient-based optimiser (e.g. BFGS).

Limitations

- Getting the function model (e.g., covariance function) wrong can be catastrophic
  - Model will not correctly predict where function is high
  - Over/under estimation of uncertainty will lead to over/under exploration
- Limited scalability in the number of dimensions and/or evaluations of the true objective function
  - Local kernels too uncertain in high dimensions
  - Gaussian processes expensive for large datasets (although BayesOpt generally doesn't have the problem of large datasets...)

Poor Model Choice





- Covariance function selection is crucial for good performance
  ⏩ Choose a sufficiently flexible and adaptive kernel, e.g., Matérn (but not the squared exponential (Gaussian))
- Nice side-effect of Matérn: Exploration is more encouraged than with the Gaussian kernel

Choosing Covariance Functions

Application:

- Structured SVM for Protein Motif Finding (Miller et al., 2012)
- Optimize hyper-parameters of SSVM using BO (Snoek et al., 2012)

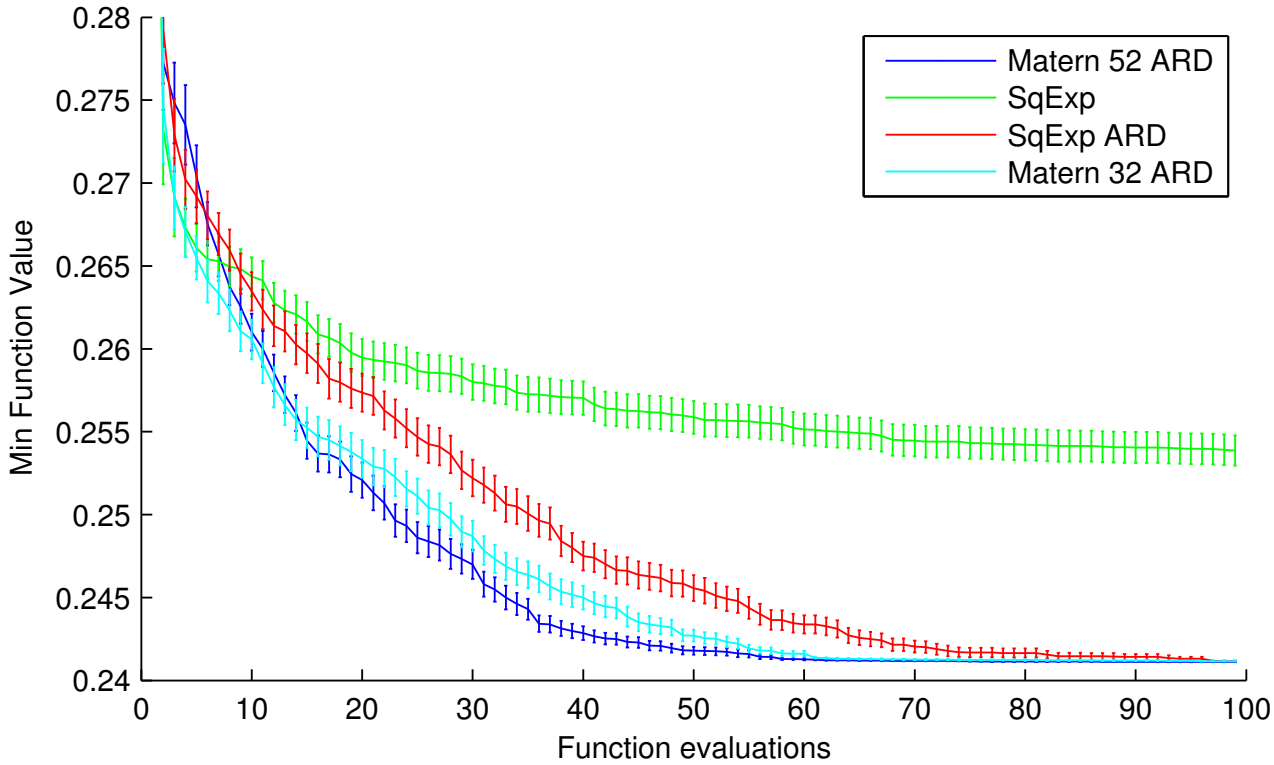


Figure: Figure from Snoek et al. (2012)

Gaussian Process Hyper-Parameters

- Empirical Bayes (maximize the marginal likelihood) can fail horribly, especially in the early stages of Bayesian optimization when we have only a few data points
- Solution: Integrate out the GP hyper-parameters $\boldsymbol{\theta}$ by Markov Chain Monte Carlo (MCMC) sampling (e.g., slice sampling)
- Look at integrated acquisition function

$$\alpha(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{\theta}}[\alpha(\boldsymbol{x}, \boldsymbol{\theta})] = \int \alpha(\boldsymbol{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$$

$$\approx \frac{1}{K} \sum_{k=1}^{K} \alpha(\boldsymbol{x}, \boldsymbol{\theta}^{(k)}), \quad \boldsymbol{\theta}^{(k)} \sim \underbrace{p(\boldsymbol{\theta} | \boldsymbol{X}_n, \boldsymbol{y}_n)}_{\text{hyper-parameter posterior}}$$

Integrating out GP Hyper-parameters

- Online LDA (Hoffman et al., 2010) for topic modeling
- Two critical hyper-parameters that control the learning rate learned by BO (Snoek et al., 2012)

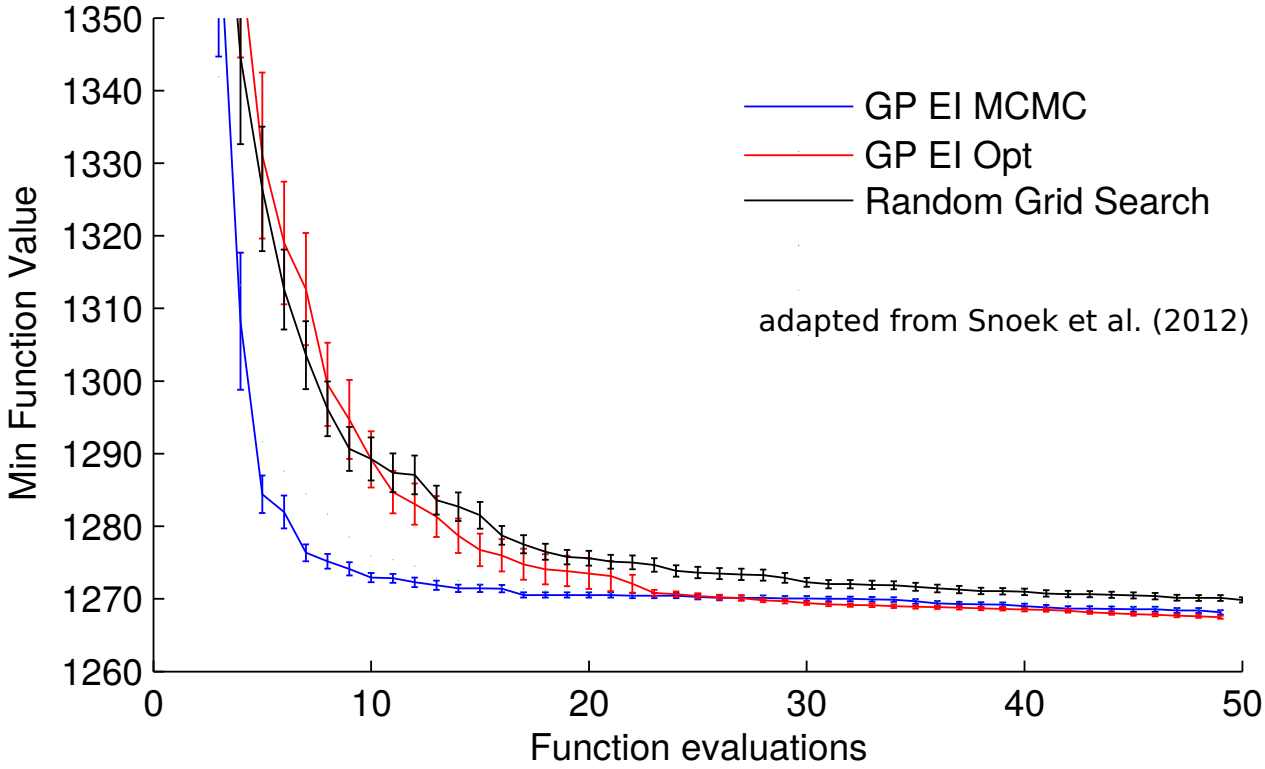
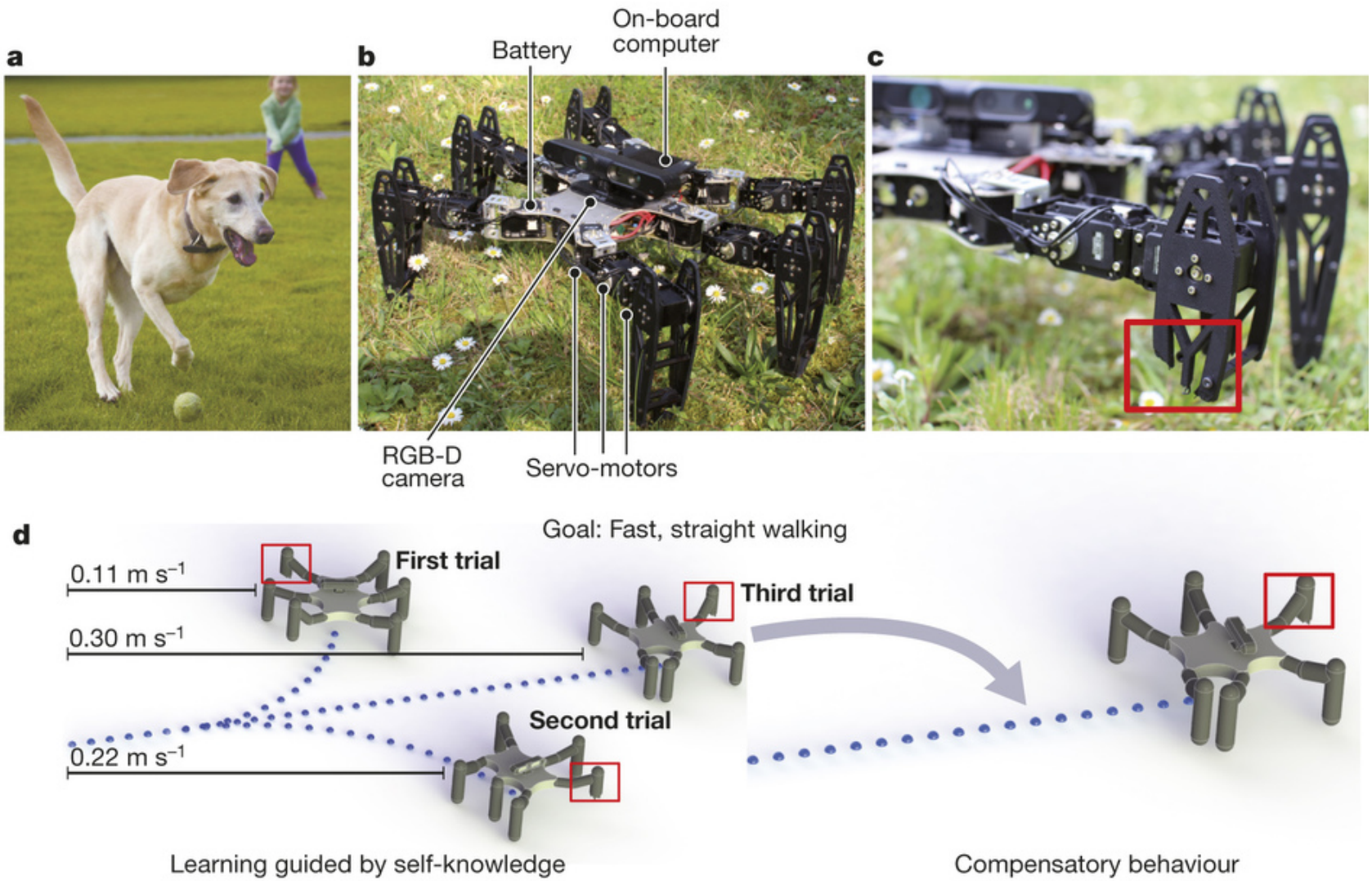


Figure: Figure from Snoek et al. (2012)

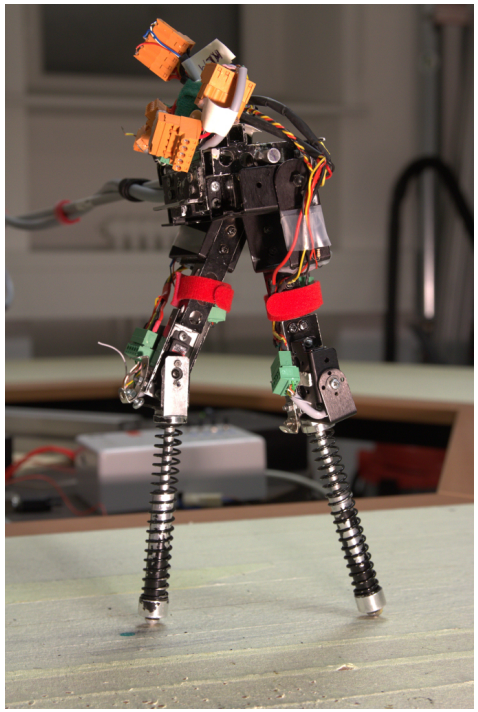
Robots That Learn to Recover from Damage



Cully et al. (2015)

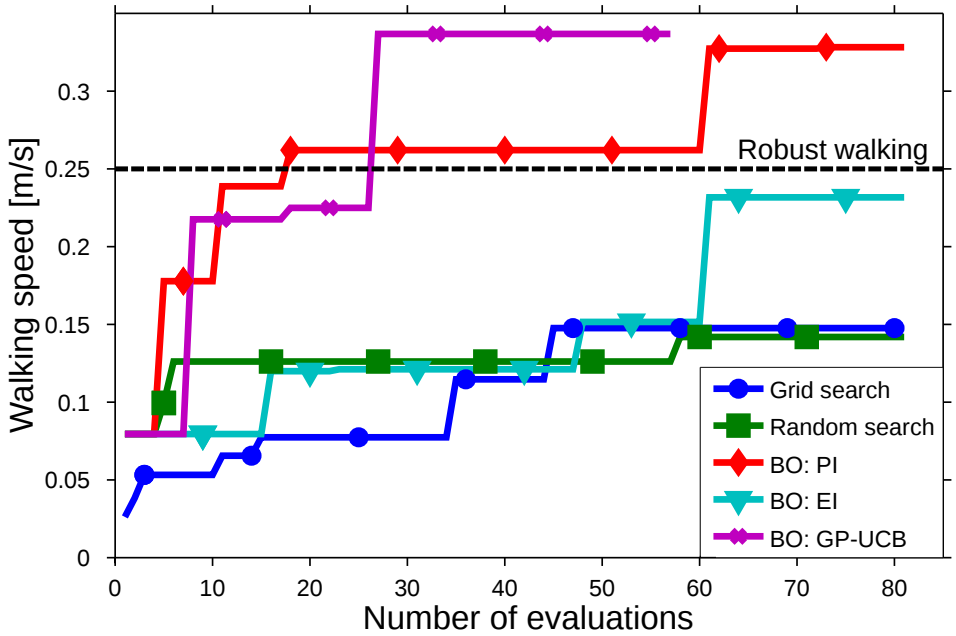
Application Example: Controller Learning in Robotics (Calandra et al., 2015)

- Fragile bipedal robot
  ⏩ Only few experiments feasible
- Maximize robustness and walking speed
- 4 motors:
  2 actuated hips + 2 actuated knees
- Controller implemented as a finite-state-machine (8 parameters)
- Good parameters found after 80–100 experiments
- Substantial speed-up compared to manual parameter search



Calandra et al. (2015)

Comparison



- Squared exponential covariance function
- Learned GP hyper-parameters (no MCMC for integrating them out)

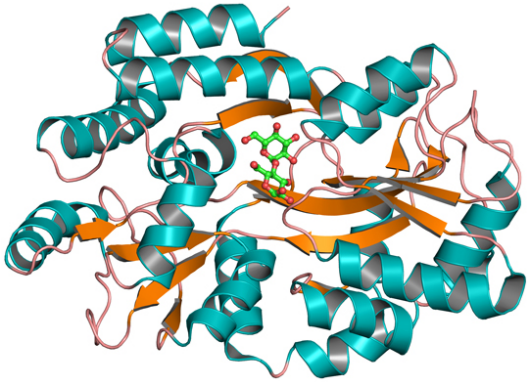
Further Topics in BO

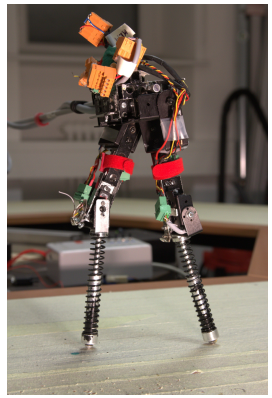
- Entropy-based acquisition functions: Directly describe the distribution over the best input location (Hennig & Schuler, 2012; Hernández-Lobato et al., 2014)
- Non-myopic Bayesian optimization (e.g., Osborne et al., 2009)
- High-dimensional optimization (e.g., Wang et al., 2016)
- Large-scale Bayesian optimization (Hutter et al., 2014)
- Efficient optimization of acquisition functions (Wilson et al., 2018)
- Non-GP Bayesian optimization (Hutter et al., 2014; Snoek et al., 2015)
- Constraints (e.g., Gelbart et al., 2014)
- Automated machine learning (e.g., Feurer et al., 2015)
- Multi-tasking, parallelizing, resource allocation, ... (e.g., Swersky et al., 2014; Snoek et al., 2012; Wilson et al., 2018)

Software

- BayesOpt `https://bitbucket.org/rmcantin/bayesopt/` (Martinez-Cantin, 2014)
- Spearmint `https://github.com/HIPS/Spearmint`
- Pybo `https://github.com/mwhoffman/pybo` (Hoffman & Shariari)
- GPyOpt `https://github.com/SheffieldML/GPyOpt` (Gonzalez et al.)
- Matlab toolbox (bayesopt)

Summary





- Global optimization of black-box functions, which are expensive to evaluate ⏩ Meta-challenges in machine learning, Auto-ML
- Use a probabilistic proxy model that is cheap to evaluate and use this to suggest next experiments
- Acquisition function trades of exploration and exploitation

Further Reading

- Brochu et al.: *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*, arXiv:1012.2599, 2012
- Shahriari et al.: *Taking the Human Out of the Loop: A Review of Bayesian Optimization*, Proceedings of the IEEE, 2016

References I