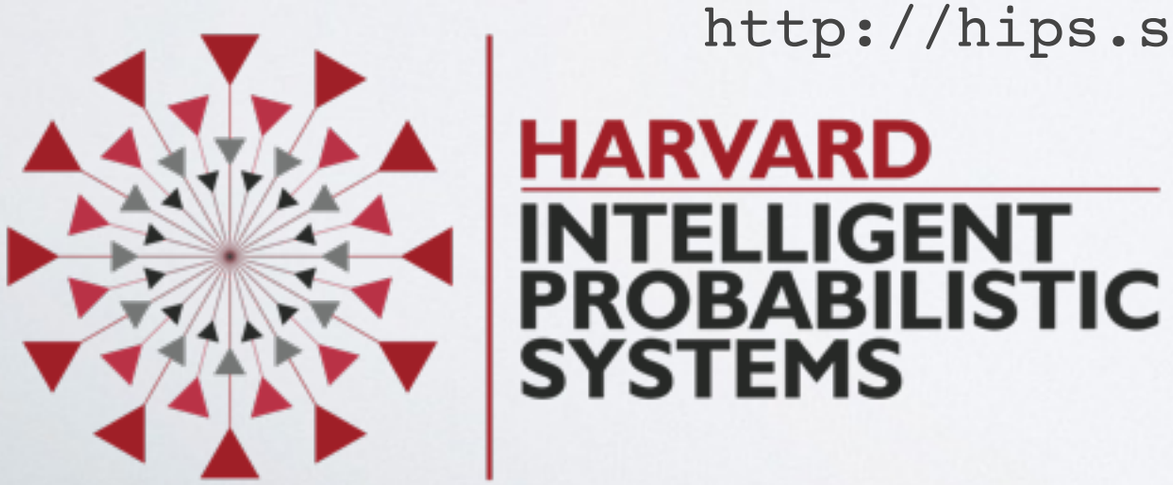
A Tutorial on Bayesian Optimization for Machine Learning

Ryan P. Adams
School of Engineering and Applied Sciences
Harvard University

Presented for CSC2515 Winter 2021
by Jonathan Lorraine

Source - https://www.iro.umontreal.ca/~bengioy/cifar/NCAP2014-summerschool/slides/Ryan_adams_140814_bayesopt_ncap.pdf

http://hips.seas.harvard.edu



Machine Learning Meta-Challenges

- **Increasing Model Complexity**
  More flexible models have more parameters.

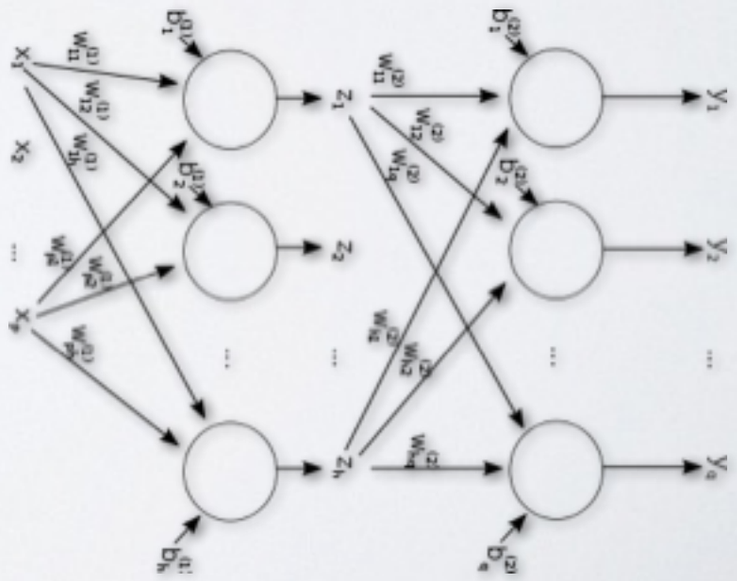
- **More Sophisticated Fitting Procedures**
  Non-convex optimization has many knobs to turn.

- **Less Accessible to Non-Experts**
  Harder to apply complicated techniques.

- **Results Are Less Reproducible**
  Too many important implementation details are missing.

Example: Deep Neural Networks

- Resurgent interest in large neural networks.
- When well-tuned, very successful for visual object identification, speech recognition, comp bio, ...
- Big investments by Google, Facebook, Microsoft, etc.
- Many choices: number of layers, weight regularization, layer size, which nonlinearity, batch size, learning rate schedule, stopping conditions



Search for Good Hyperparameters?

- **Define an objective function.**
  Most often, we care about generalization performance.
  Use cross validation to measure parameter quality.

- **How do people currently search? Black magic.**
  Grid search
  Random search
  Grad student descent

- **Painful!**
  Requires many training cycles.
  Possibly noisy.

Can We Do Better? Bayesian Optimization

- **Build a probabilistic model for the objective.**
  Include hierarchical structure about units, etc.

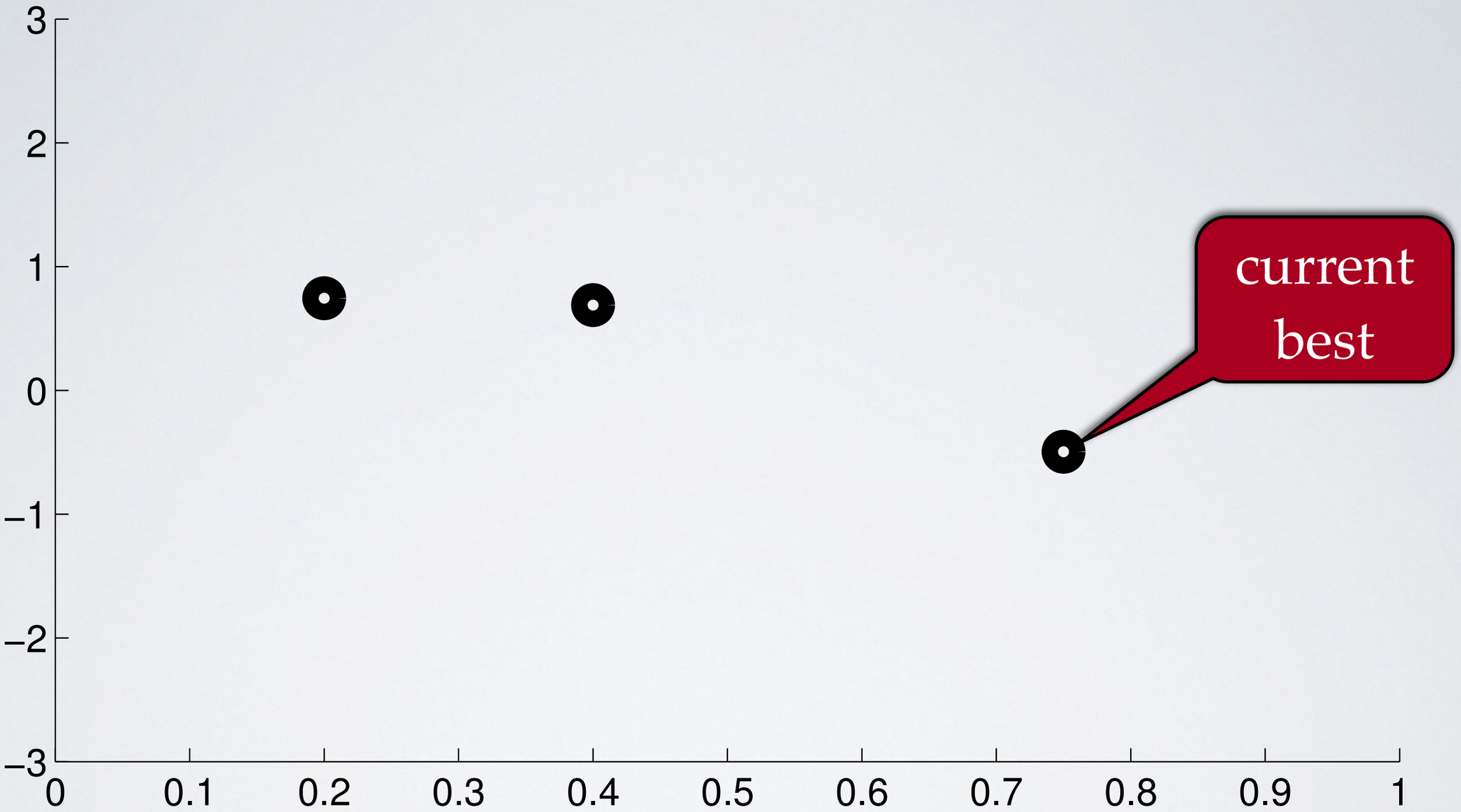
- **Compute the posterior predictive distribution.**
  Integrate out all the possible true functions.
  We use Gaussian process regression.

- **Optimize a cheap proxy function instead.**
  The model is much cheaper than that true objective.

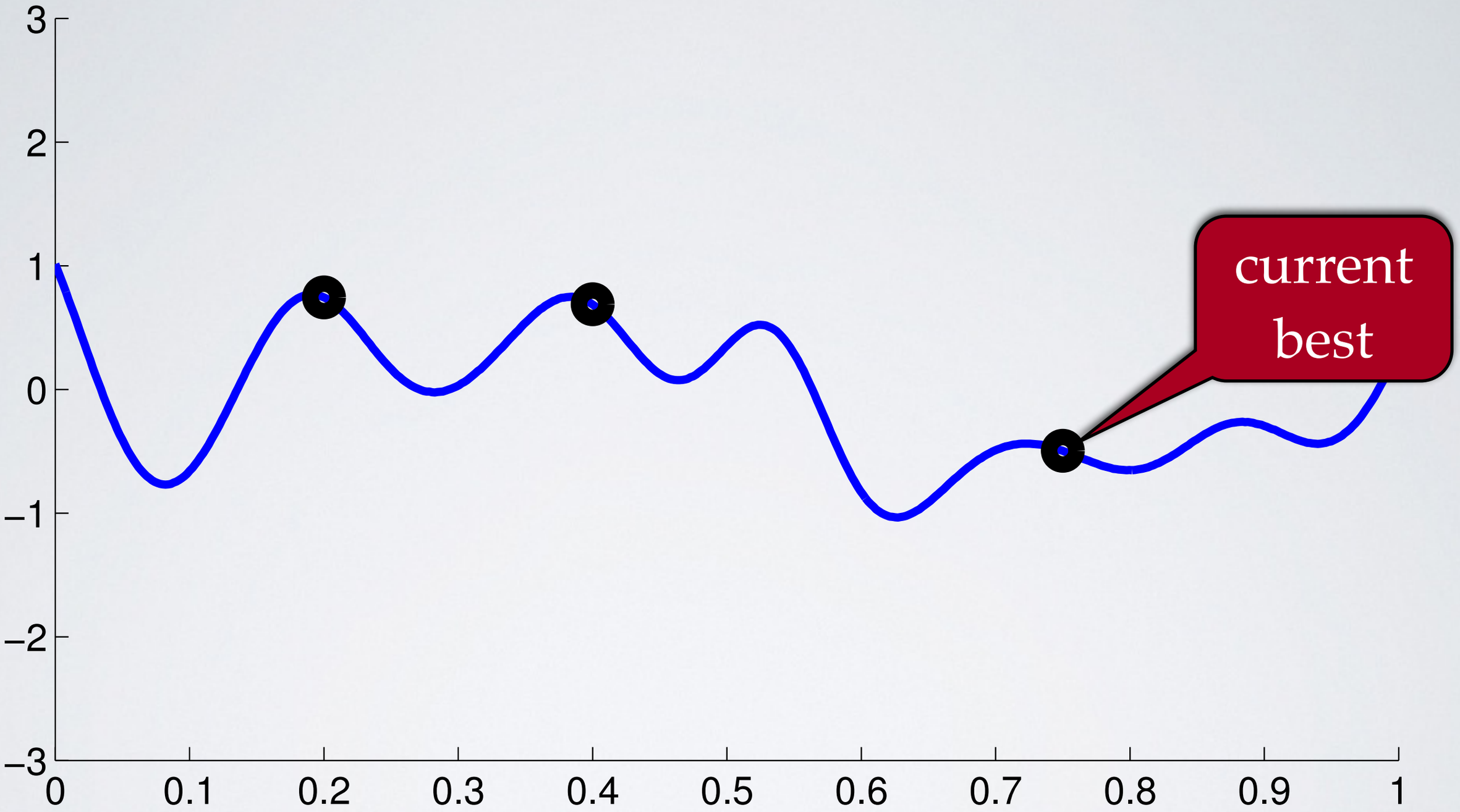
The main insight:

Make the proxy function exploit uncertainty to balance **exploration** against **exploitation**.

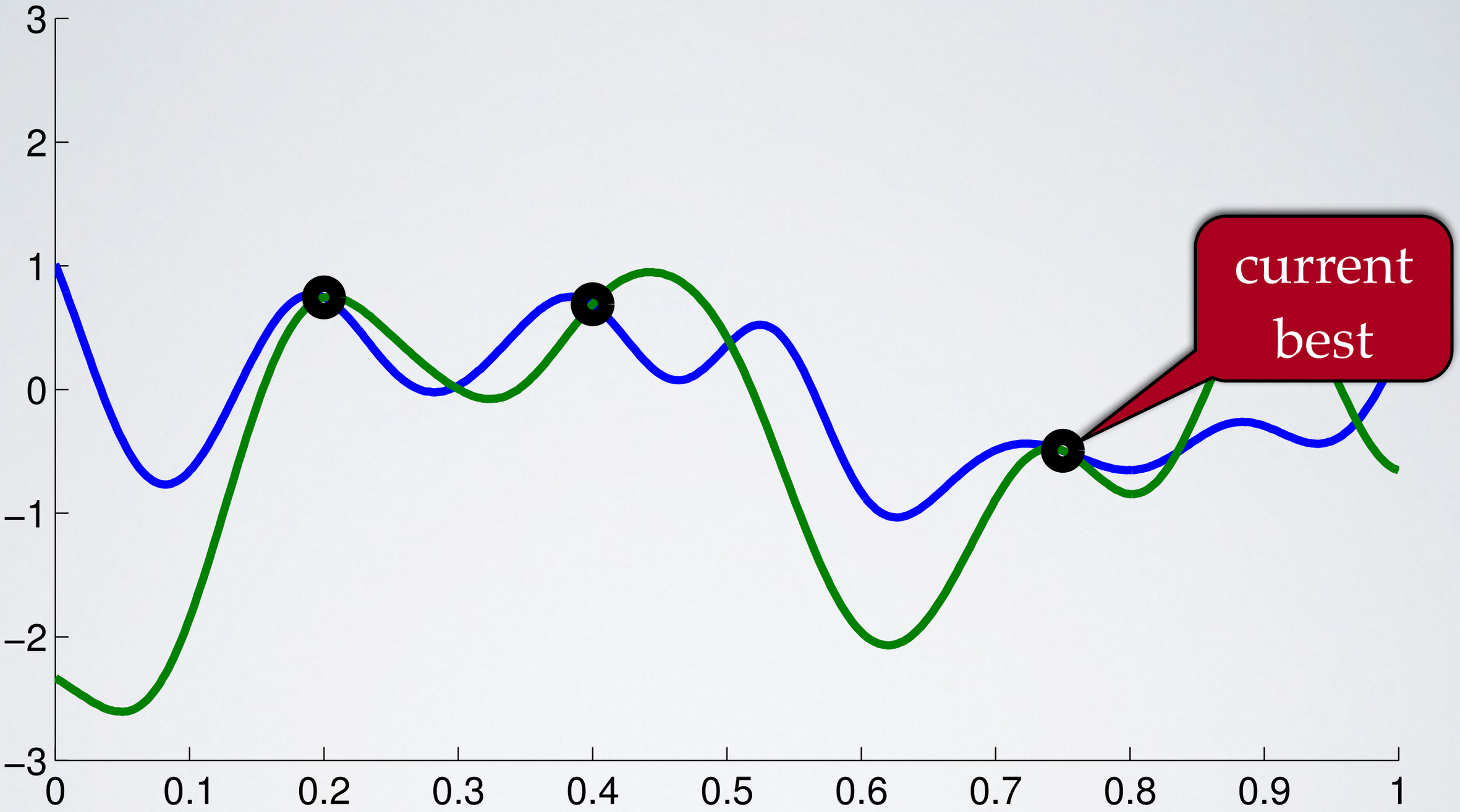
The Bayesian Optimization Idea



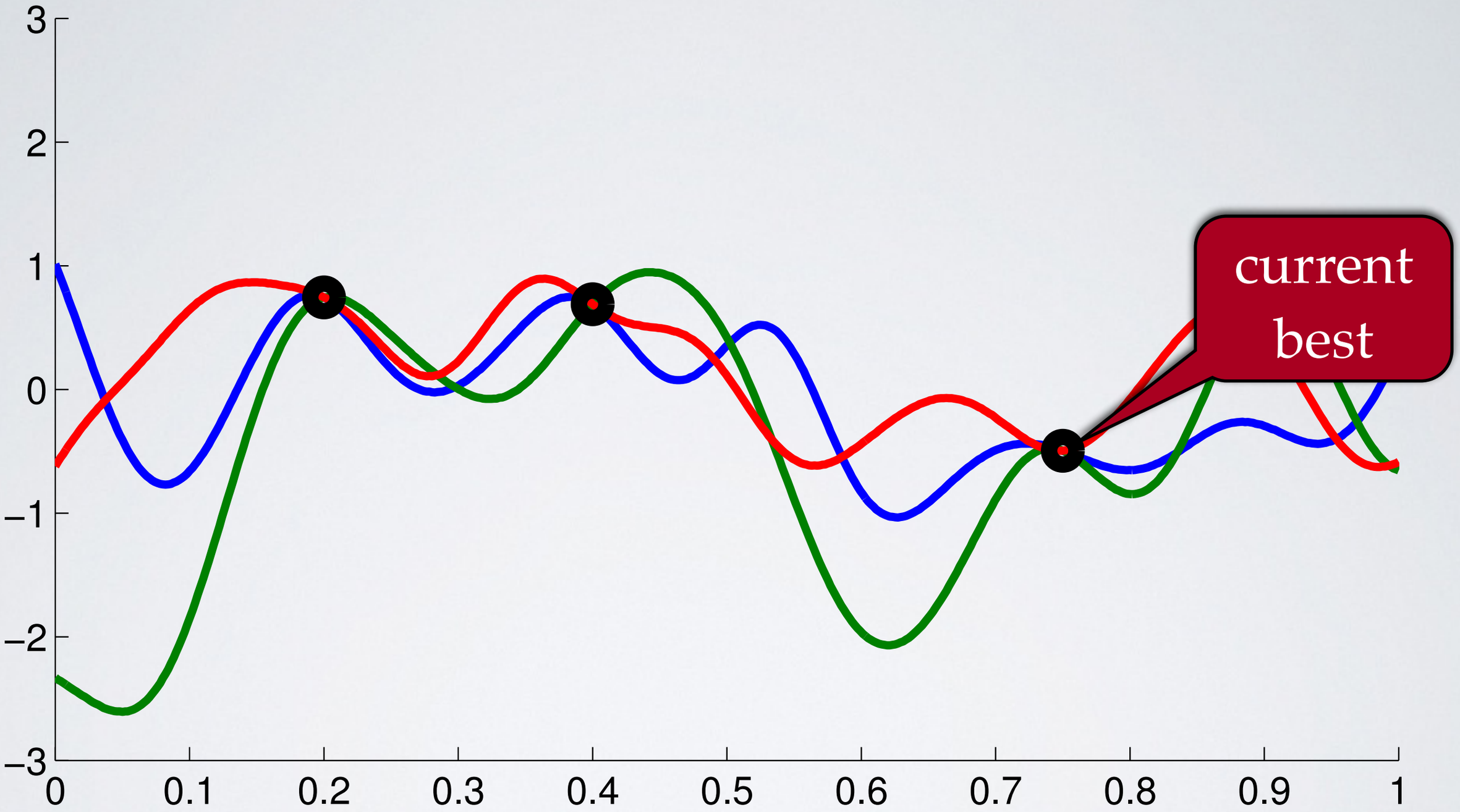
The Bayesian Optimization Idea



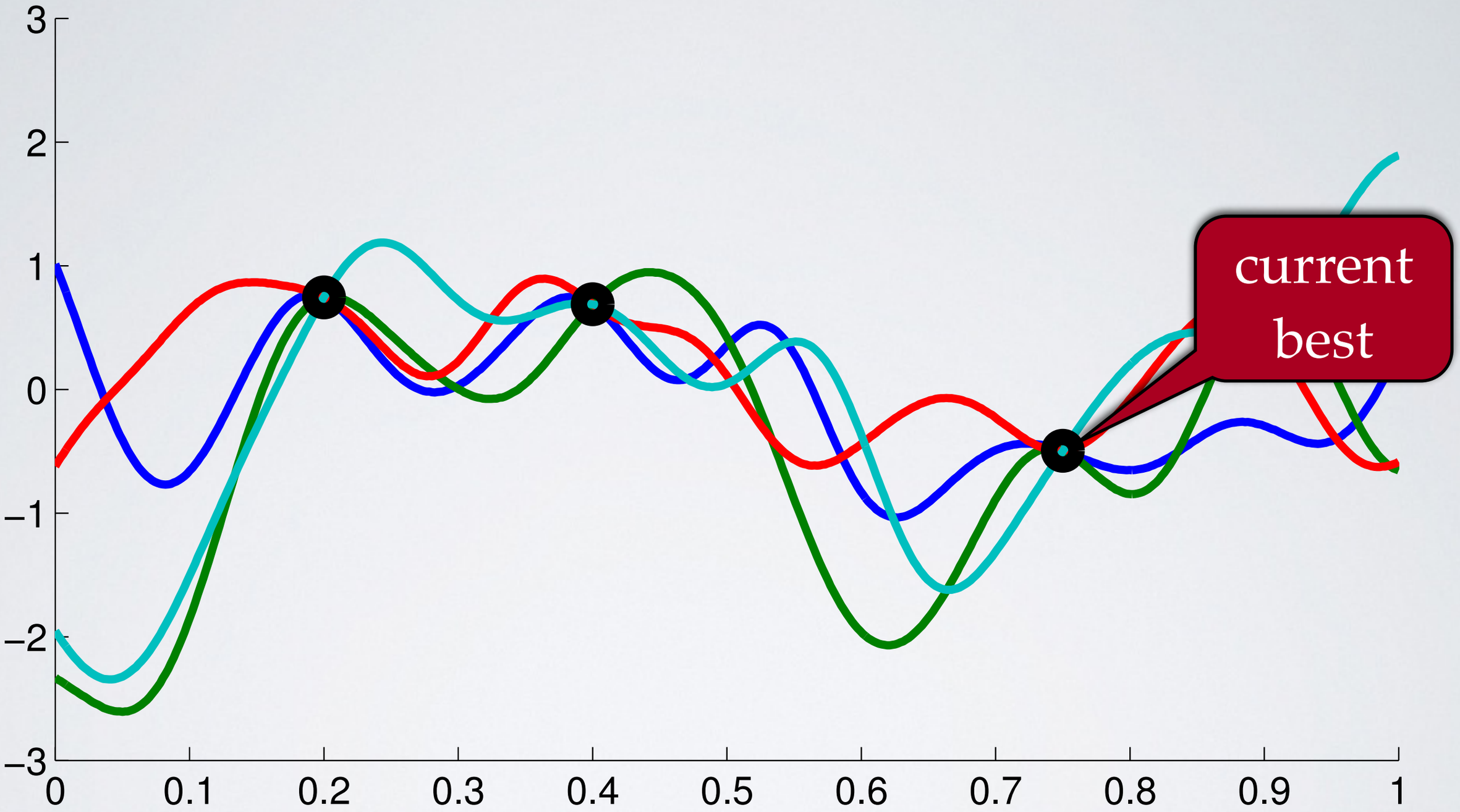
The Bayesian Optimization Idea



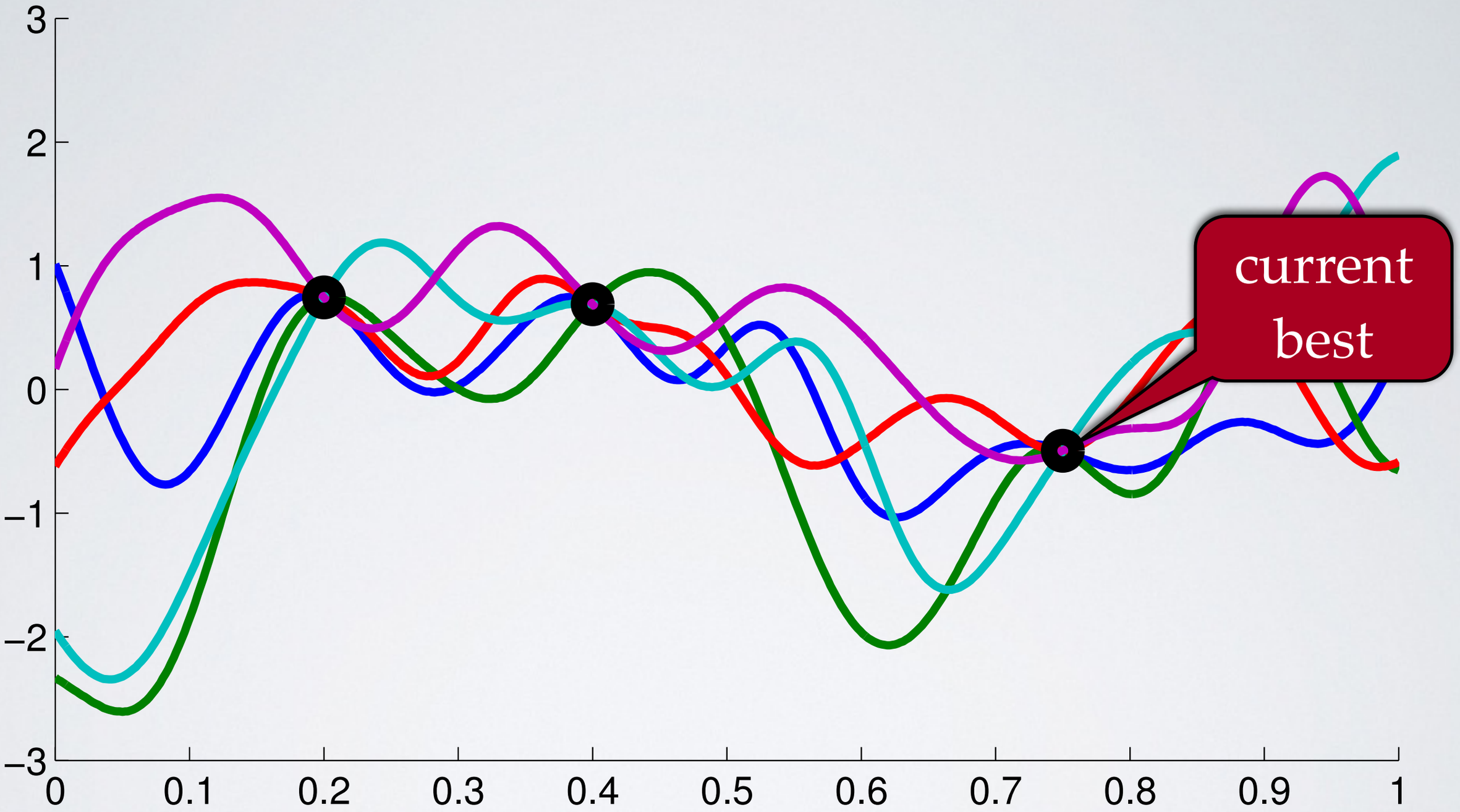
The Bayesian Optimization Idea



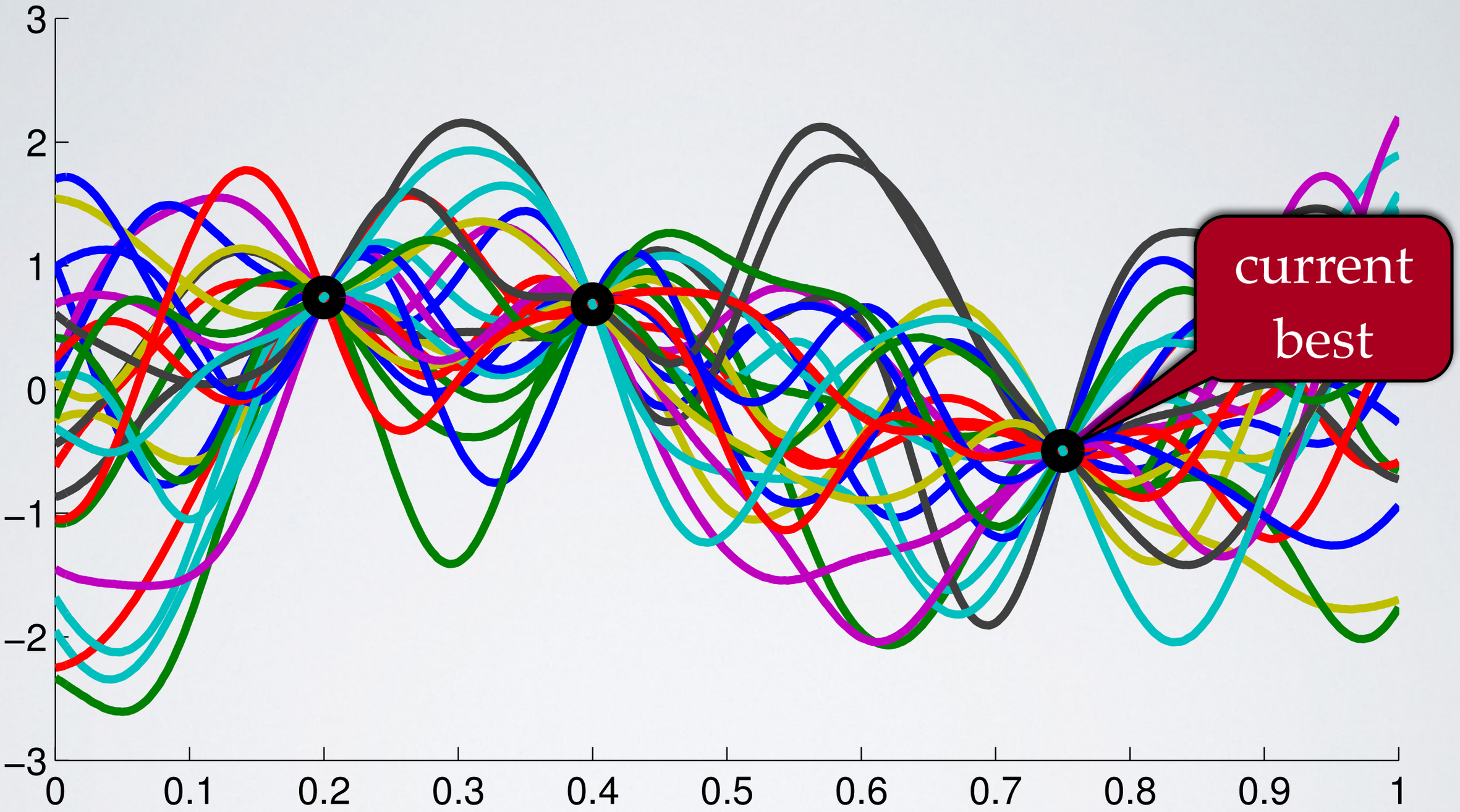
The Bayesian Optimization Idea



The Bayesian Optimization Idea

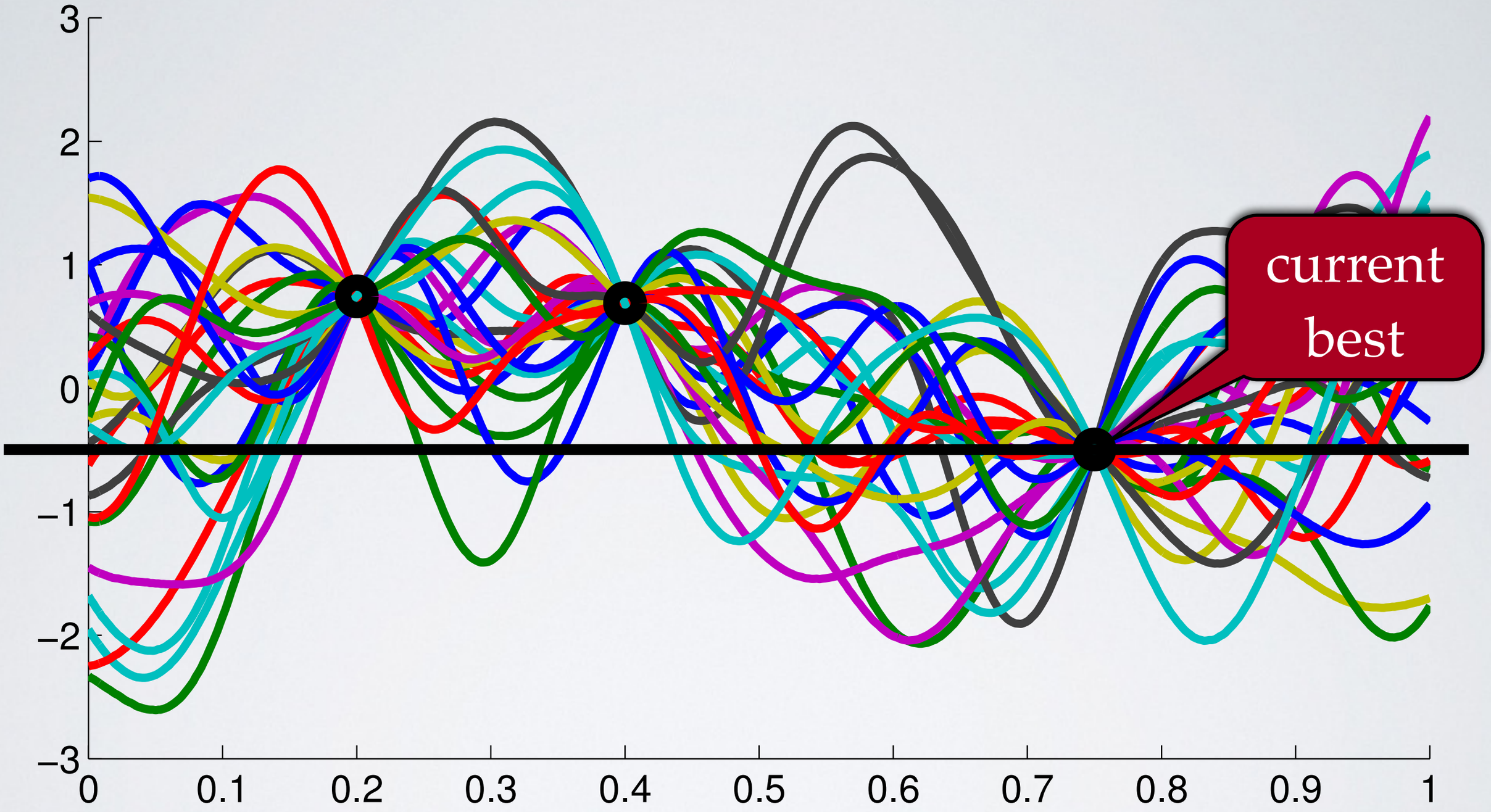


The Bayesian Optimization Idea



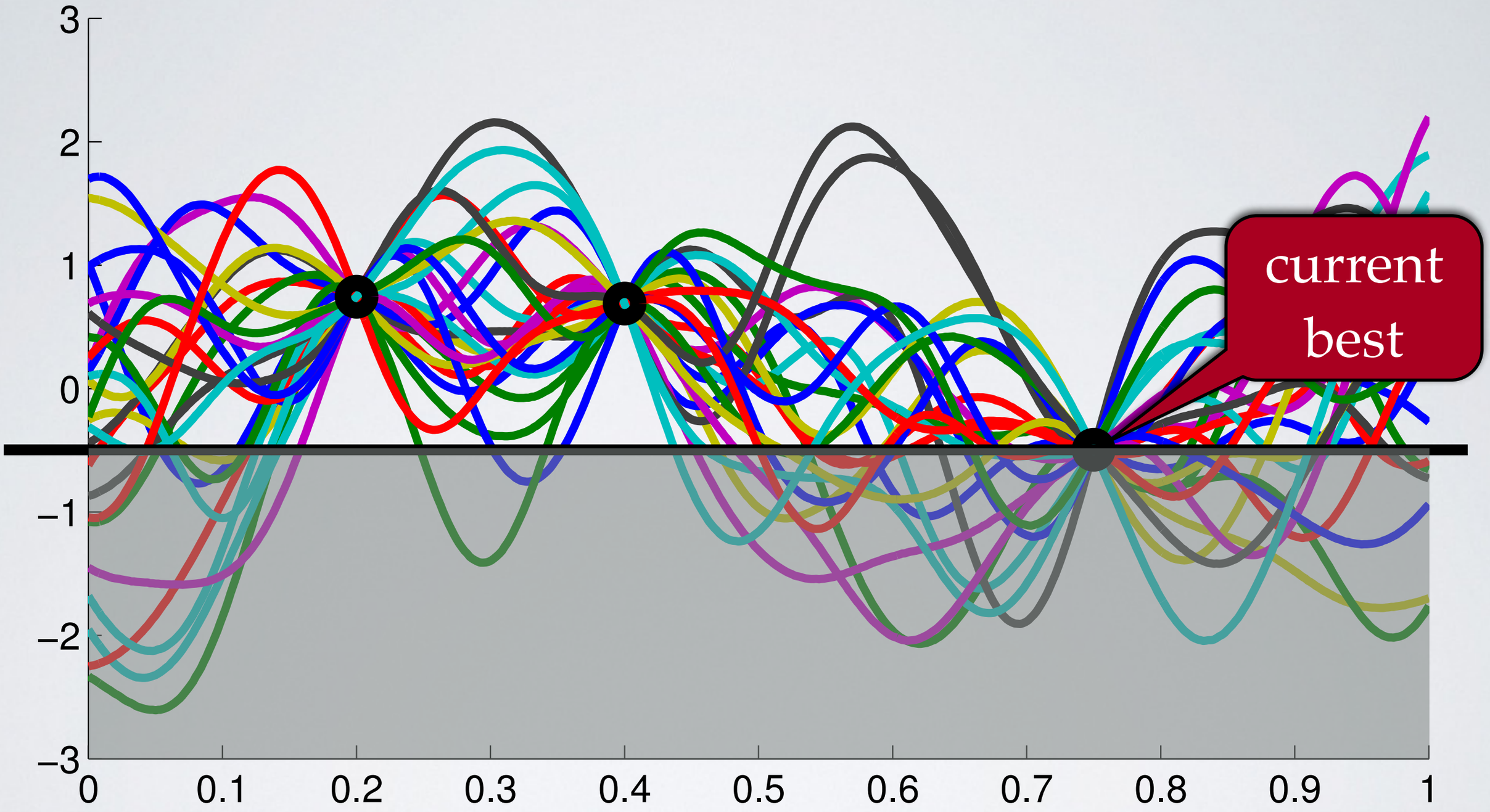
Where should we evaluate next
in order to improve the most?

The Bayesian Optimization Idea



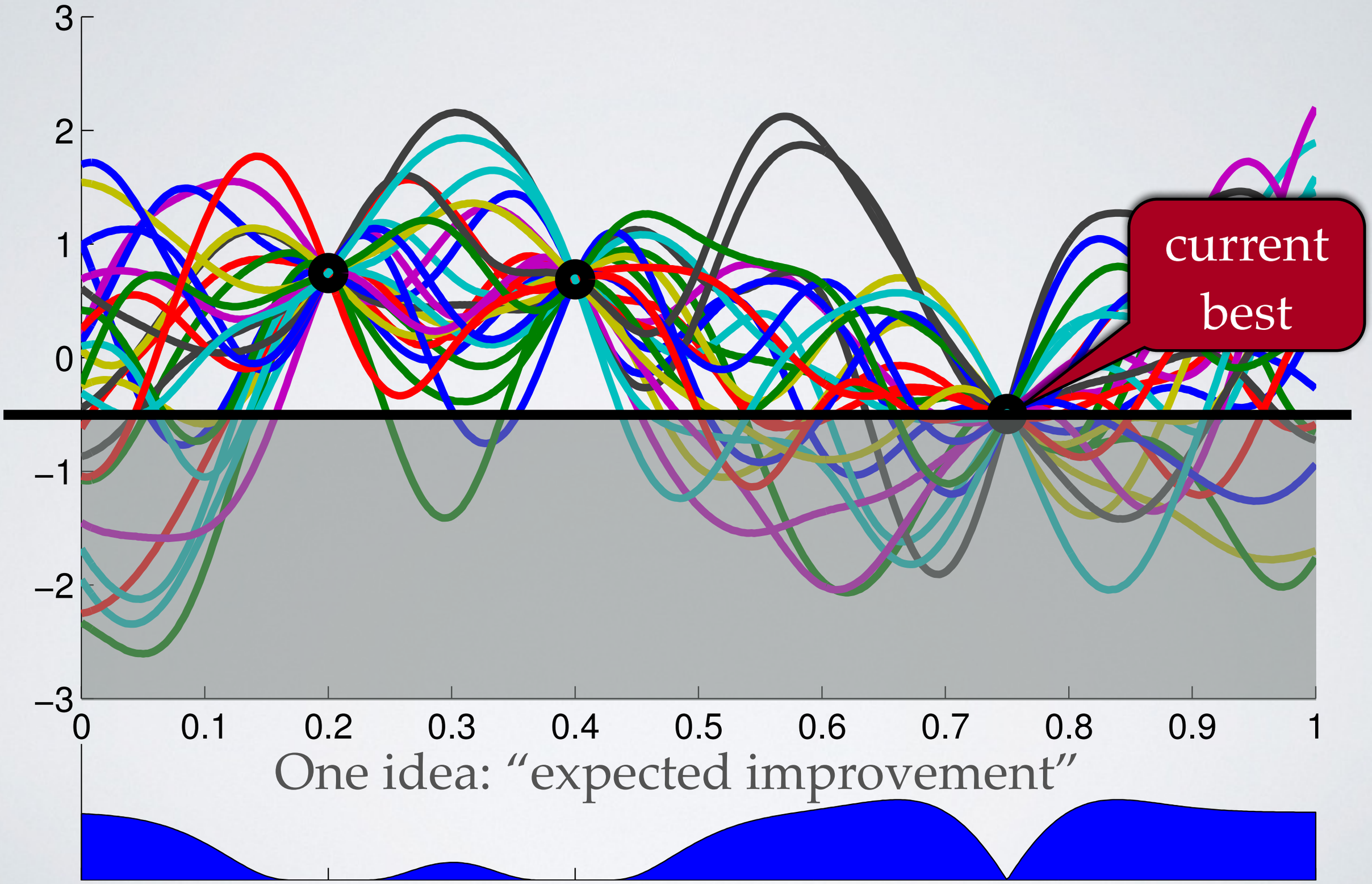
Where should we evaluate next
in order to improve the most?

The Bayesian Optimization Idea



Where should we evaluate next
in order to improve the most?

The Bayesian Optimization Idea



One idea: “expected improvement”

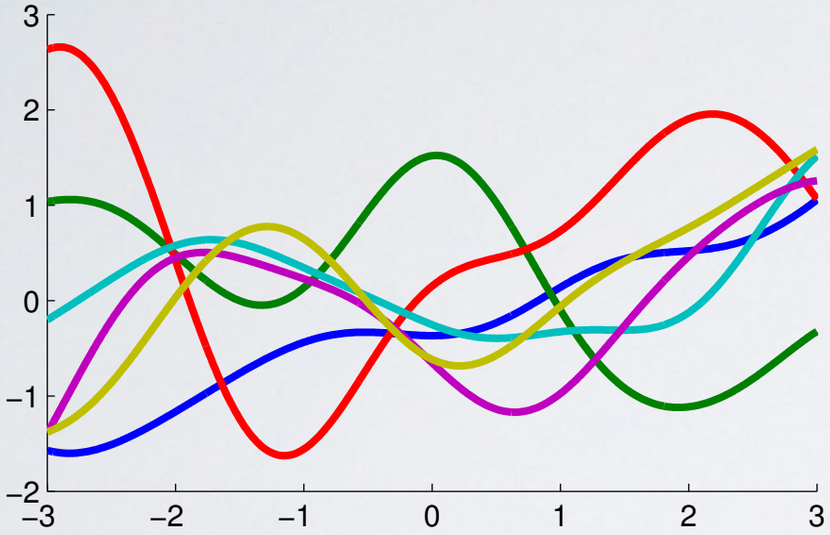
Today's Topics

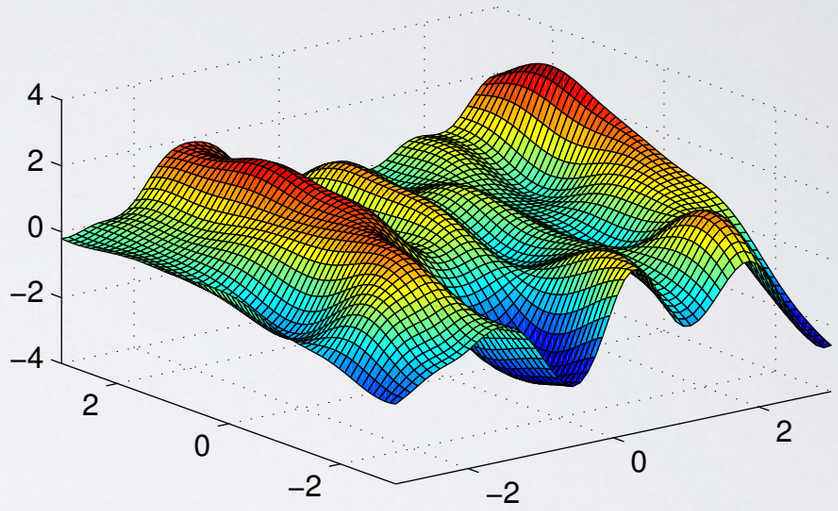
- Review of Gaussian process priors
- Bayesian optimization basics
- Managing covariances and kernel parameters
- ~~Accounting for the cost of evaluation~~
- ~~Parallelizing training~~
- ~~Sharing information across related problems~~
- ~~Better models for nonstationary functions~~
- ~~Random projections for high-dimensional problems~~
- ~~Accounting for constraints~~
- ~~Leveraging partially-completed training runs~~

The crossed out material is not necessary to cover in tutorial.

Gaussian Processes as Function Models

Nonparametric prior on functions specified in terms of a positive definite kernel.

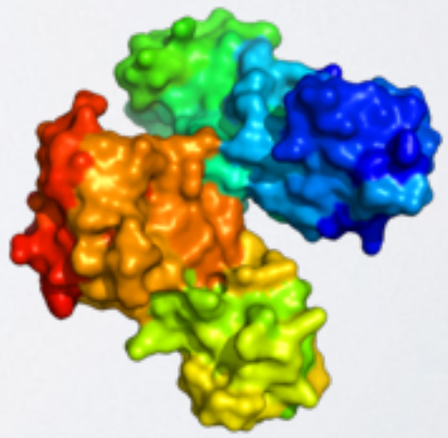





```
the quick brown fox
jumps over the lazy dog
```

strings (Haussler 1999)

permutations (Kondor 2008)



proteins (Borgwardt 2007)

Gaussian Processes

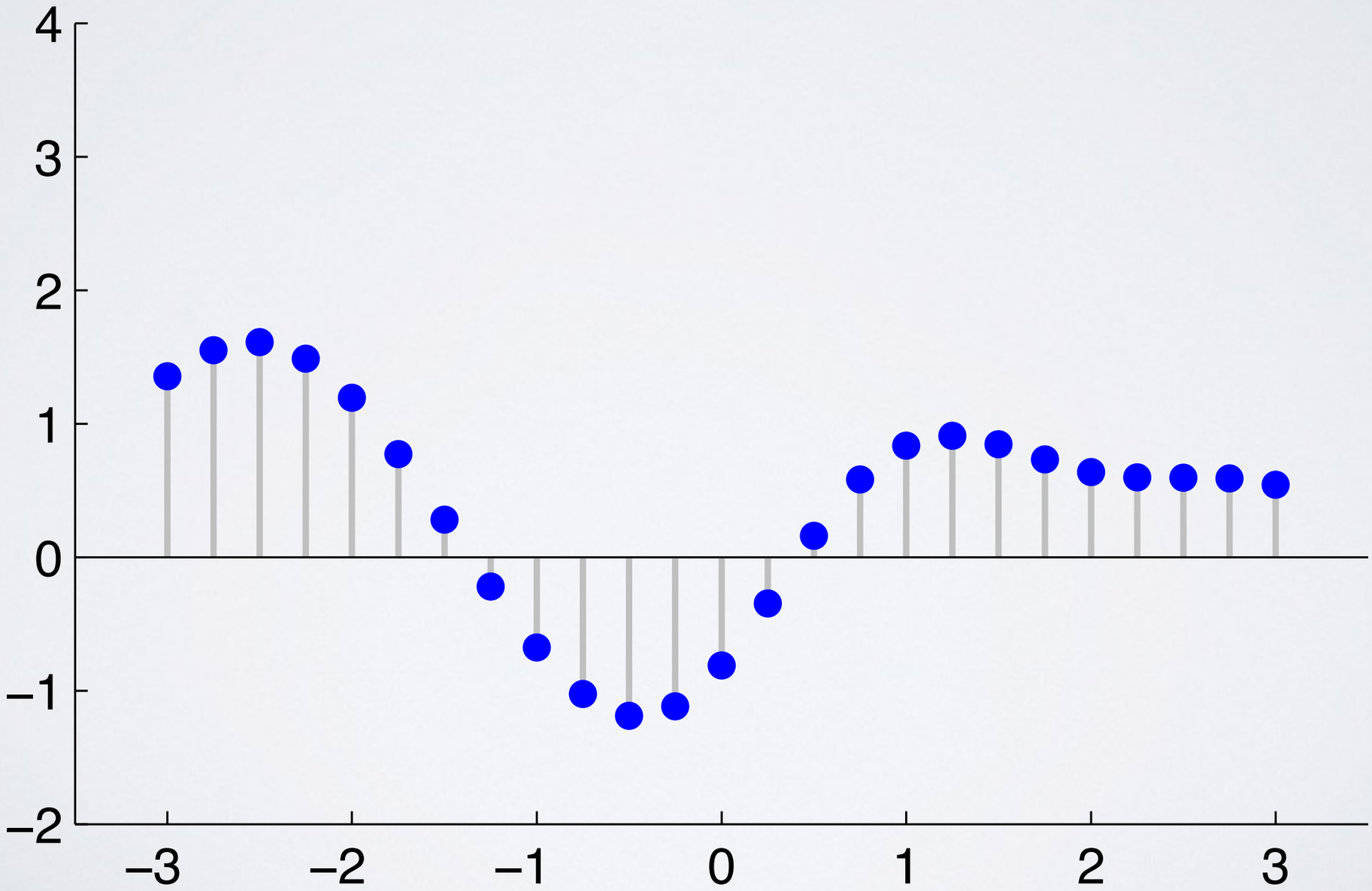
- Gaussian process (GP) is a distribution on functions.
- Allows tractable Bayesian modeling of functions without specifying a particular finite basis.
- Input space (where we're optimizing) $\mathcal{X}$
- Model scalar functions $f : \mathcal{X} \to \mathbb{R}$
- Positive definite covariance function $C : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$
- Mean function $m : \mathcal{X} \to \mathbb{R}$

Gaussian Processes

Any finite set of N points in $\mathcal{X}$, $\{x_n\}_{n=1}^N$ induces a homologous N-dimensional Gaussian distribution on $\mathbb{R}^N$, taken to be the distribution on $\{y_n = f(x_n)\}_{n=1}^N$

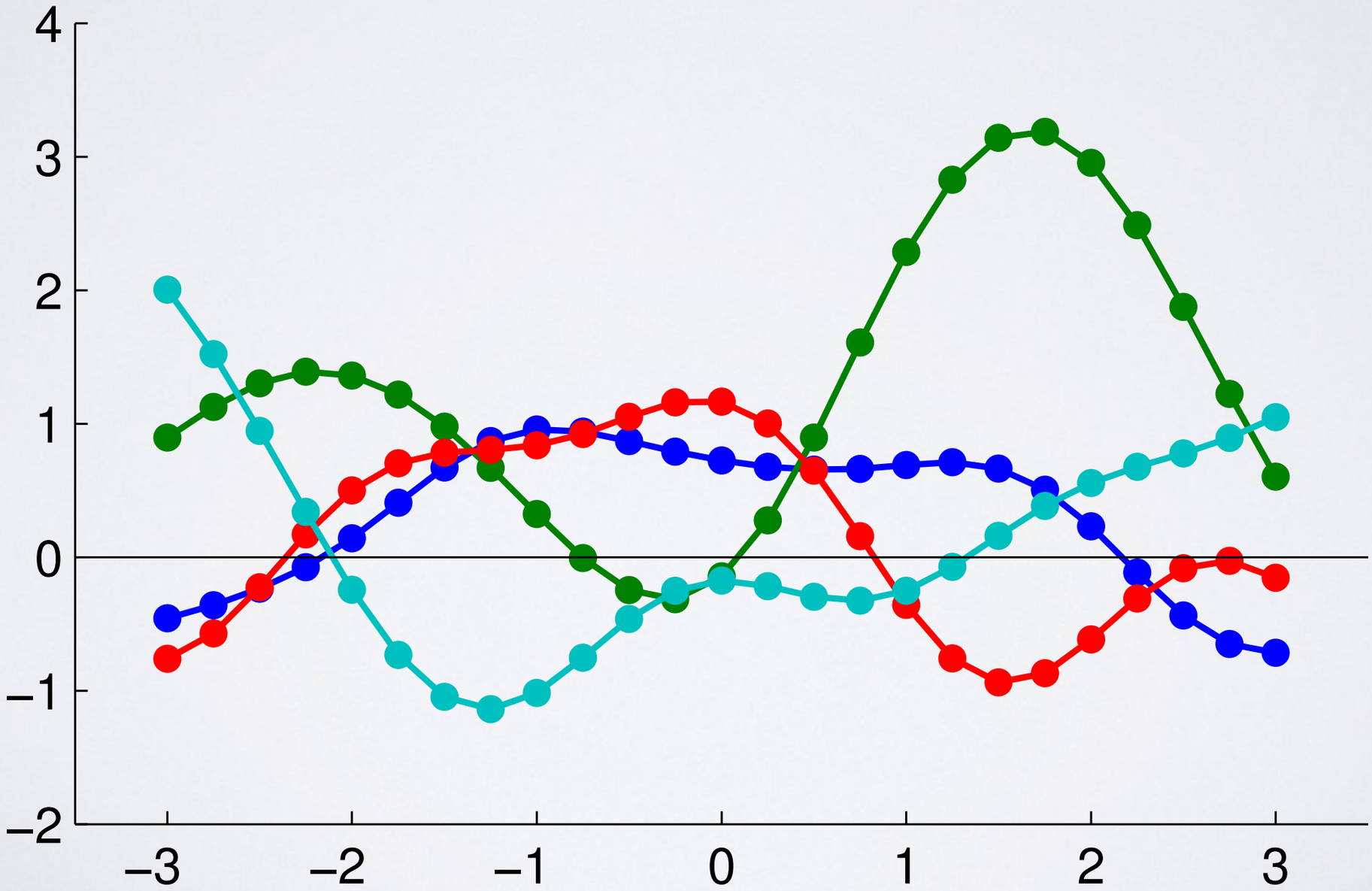
Gaussian Processes

Any finite set of N points in $\mathcal{X}$, $\{x_n\}_{n=1}^N$ induces a homologous N-dimensional Gaussian distribution on $\mathbb{R}^N$, taken to be the distribution on $\{y_n = f(x_n)\}_{n=1}^N$



Gaussian Processes

Any finite set of N points in $\mathcal{X}$, $\{x_n\}_{n=1}^N$ induces a homologous N-dimensional Gaussian distribution on $\mathbb{R}^N$, taken to be the distribution on $\{y_n = f(x_n)\}_{n=1}^N$



Gaussian Processes

- Due to Gaussian form, closed-form solutions for many useful questions about finite data.
- Marginal likelihood:

$$\ln p(\boldsymbol{y} \mid \boldsymbol{X}, \theta) = -\frac{N}{2} \ln 2\pi - \frac{1}{2} \ln |\boldsymbol{K}_\theta| - \frac{1}{2} \boldsymbol{y}^\mathsf{T} \boldsymbol{K}_\theta^{-1} \boldsymbol{y}$$

- Predictive distribution at test points $\{\boldsymbol{x}_m\}_{m=1}^M$:

$$\boldsymbol{y}^{\mathsf{test}} \sim \mathcal{N}(\boldsymbol{m}, \boldsymbol{\Sigma})$$

$$\boldsymbol{m} = \boldsymbol{k}_\theta^\mathsf{T} \boldsymbol{K}_\theta^{-1} \boldsymbol{y} \qquad \boldsymbol{\Sigma} = \boldsymbol{\kappa}_\theta - \boldsymbol{k}_\theta^\mathsf{T} \boldsymbol{K}_\theta^{-1} \boldsymbol{k}_\theta$$

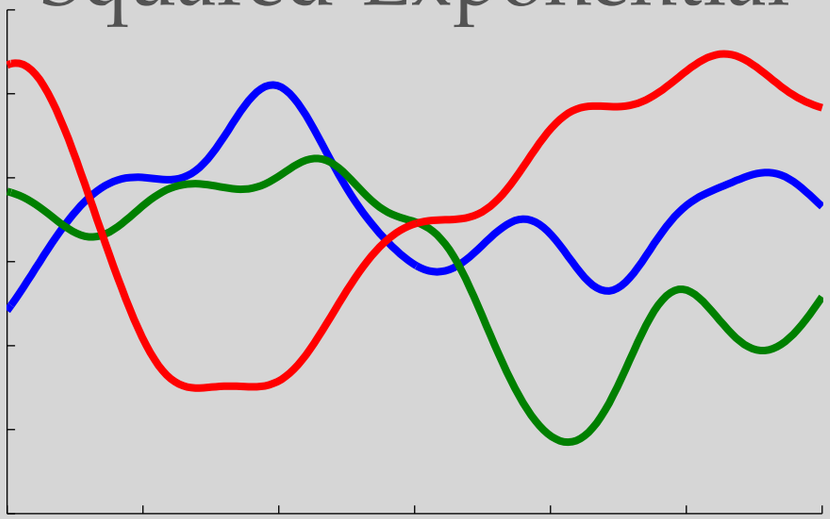
- We compute these matrices from the covariance:

$$[\boldsymbol{K}_\theta]_{n,n'} = C(\boldsymbol{x}_n, \boldsymbol{x}_{n'}\,;\,\theta)$$

$$[\boldsymbol{k}_\theta]_{n,m} = C(\boldsymbol{x}_n, \boldsymbol{x}_m\,;\,\theta) \qquad [\boldsymbol{\kappa}_\theta]_{m,m'} = C(\boldsymbol{x}_m, \boldsymbol{x}_{m'}\,;\,\theta)$$

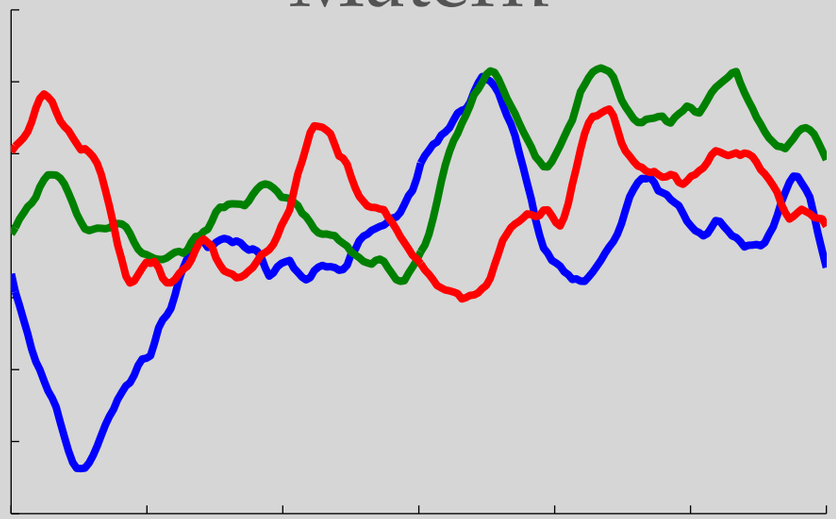
Examples of GP Covariances

Squared-Exponential



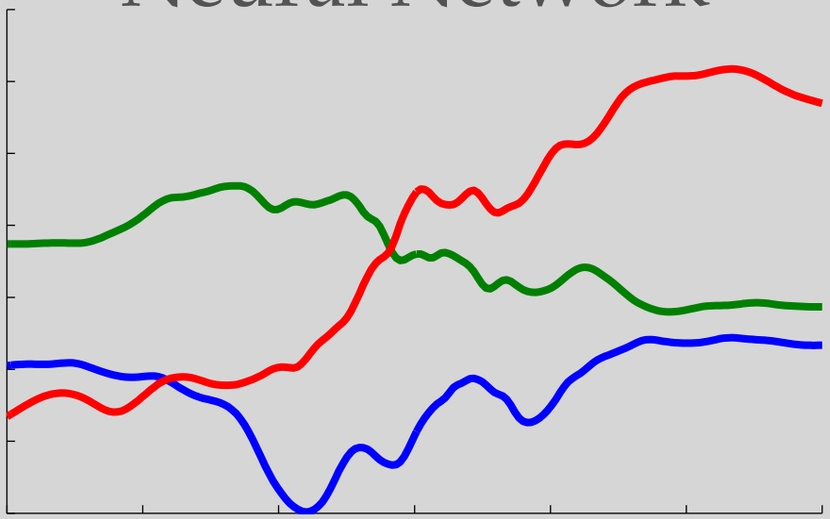
$$C(x, x') = \alpha \exp\left\{-\frac{1}{2}\sum_{d=1}^{D}\left(\frac{x_d - x'_d}{\ell_d}\right)^2\right\}$$

Matérn



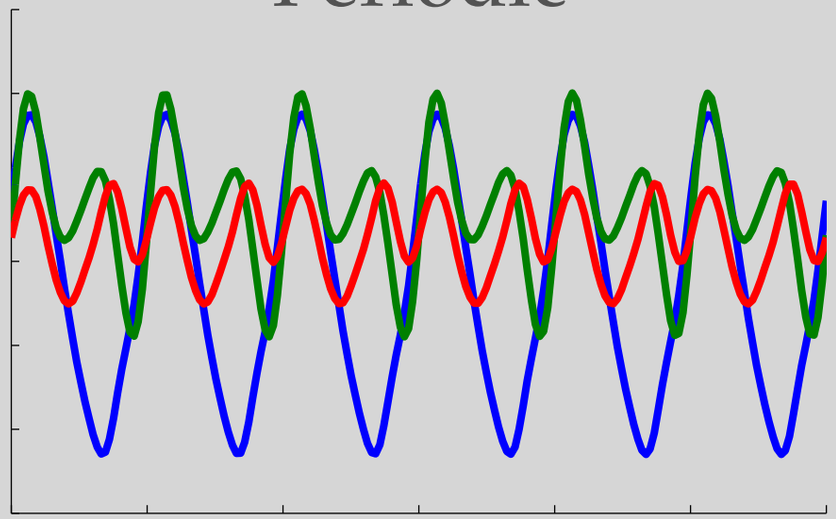
$$C(r) = \frac{2^{1-\nu}}{\Gamma(\nu)}\left(\frac{\sqrt{2\nu}\,r}{\ell}\right)^{\nu} K_\nu\left(\frac{\sqrt{2\nu}\,r}{\ell}\right)$$

"Neural Network"



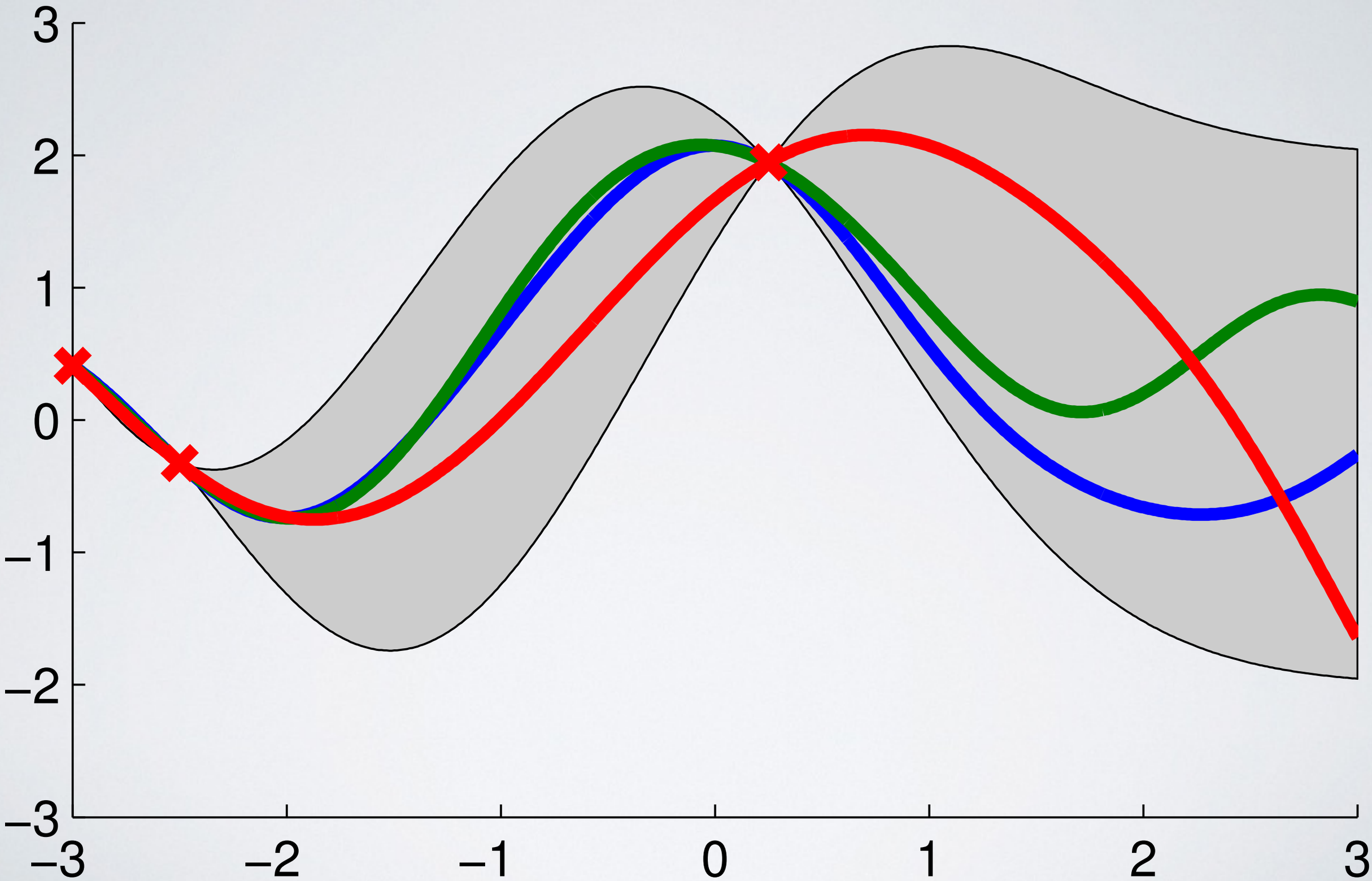
$$C(x, x') = \frac{2}{\pi}\sin^{-1}\left\{\frac{2x^{\mathsf{T}}\Sigma x'}{\sqrt{(1 + 2x^{\mathsf{T}}\Sigma x)(1 + 2x'^{\mathsf{T}}\Sigma x')}}\right\}$$

Periodic



$$C(x, x') = \exp\left\{-\frac{2\sin^2\left(\frac{1}{2}(x - x')\right)}{\ell^2}\right\}$$

GPs Provide Closed-Form Predictions



Using Uncertainty in Optimization

- Find the minimum: $x^\star = \arg\min_{x \in \mathcal{X}} f(x)$
- We can evaluate the objective pointwise, but do not have an easy functional form or gradients.
- After performing some evaluations, the GP gives us easy closed-form marginal means and variances.
- **Exploration**: Seek places with high variance.
- **Exploitation**: Seek places with low mean.
- The **acquisition function** balances these for our proxy optimization to determine the next evaluation.

Closed-Form Acquisition Functions

- The GP posterior gives a predictive mean function $\mu(x)$ and a predictive marginal variance function $\sigma^2(x)$

$$\gamma(x) = \frac{f(x_{\mathsf{best}}) - \mu(x)}{\sigma(x)}$$

- Probability of Improvement (Kushner 1964):

$$a_{\mathsf{PI}}(x) = \Phi(\gamma(x))$$

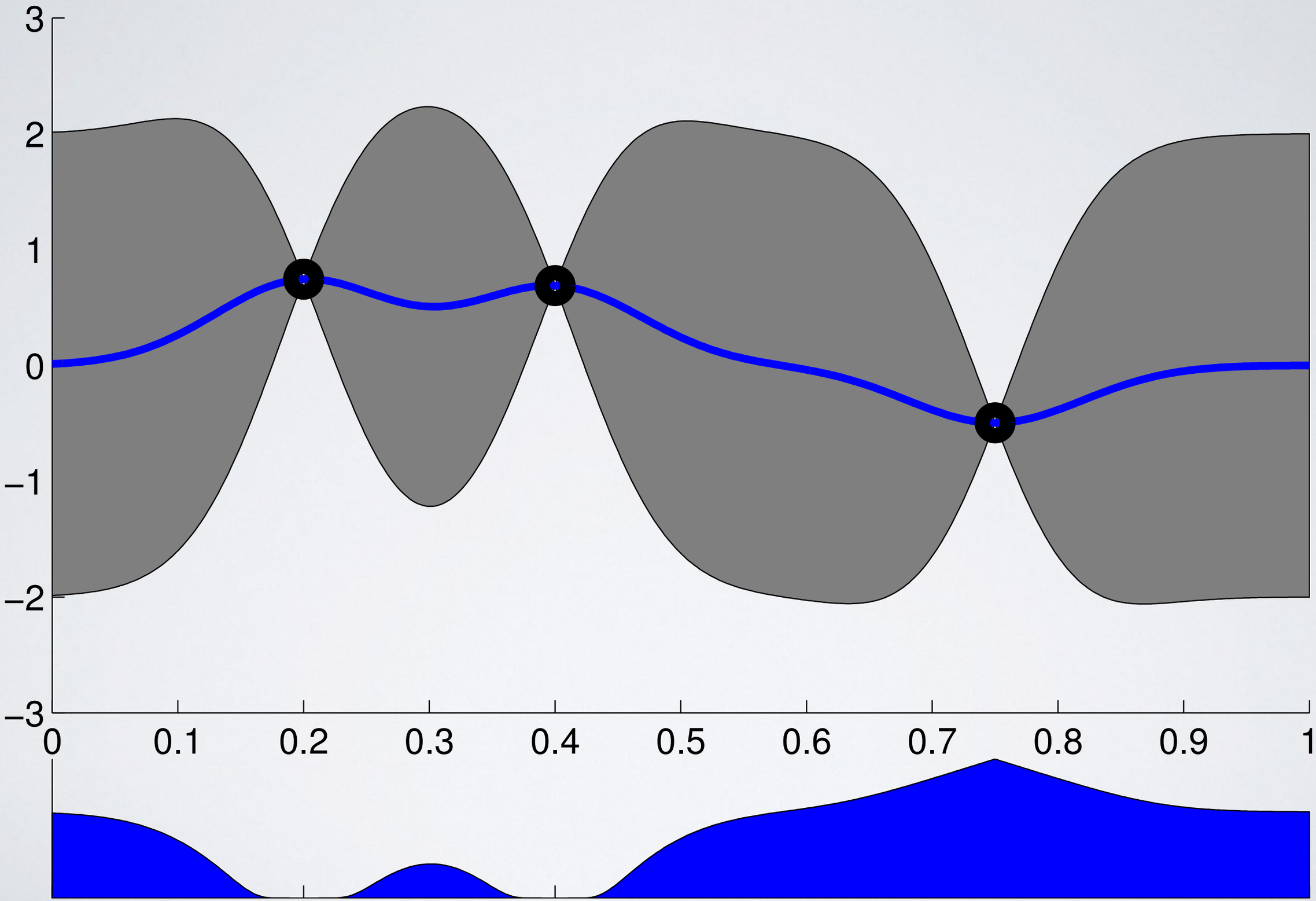
- Expected Improvement (Mockus 1978):

$$a_{\mathsf{EI}}(x) = \sigma(x)(\gamma(x)\Phi(\gamma(x)) + \mathcal{N}(\gamma(x)\,;\,0,1))$$

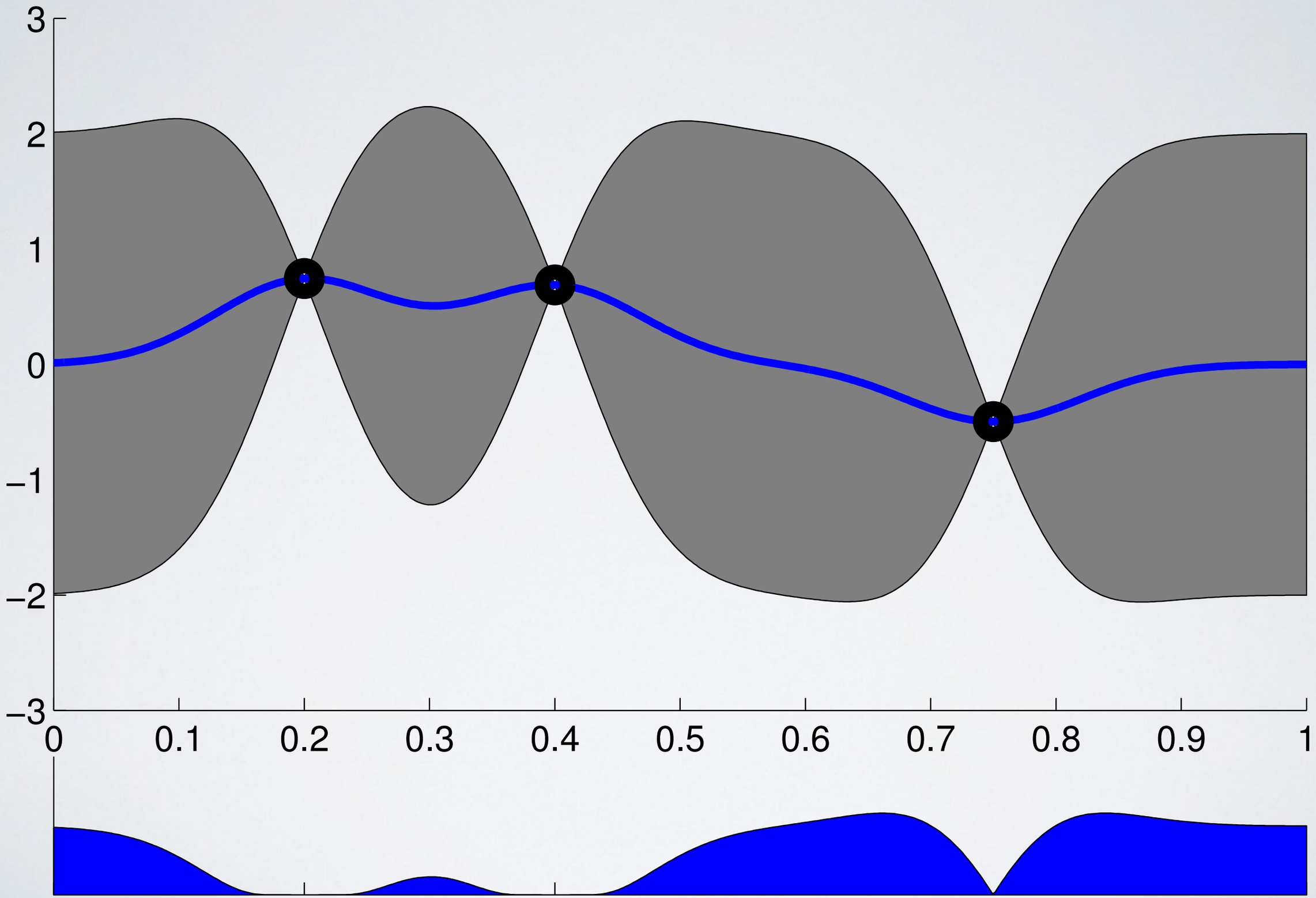
- GP Upper Confidence Bound (Srinivas et al. 2010):

$$a_{\mathsf{LCB}}(x) = \mu(x) - \kappa\,\sigma(x)$$

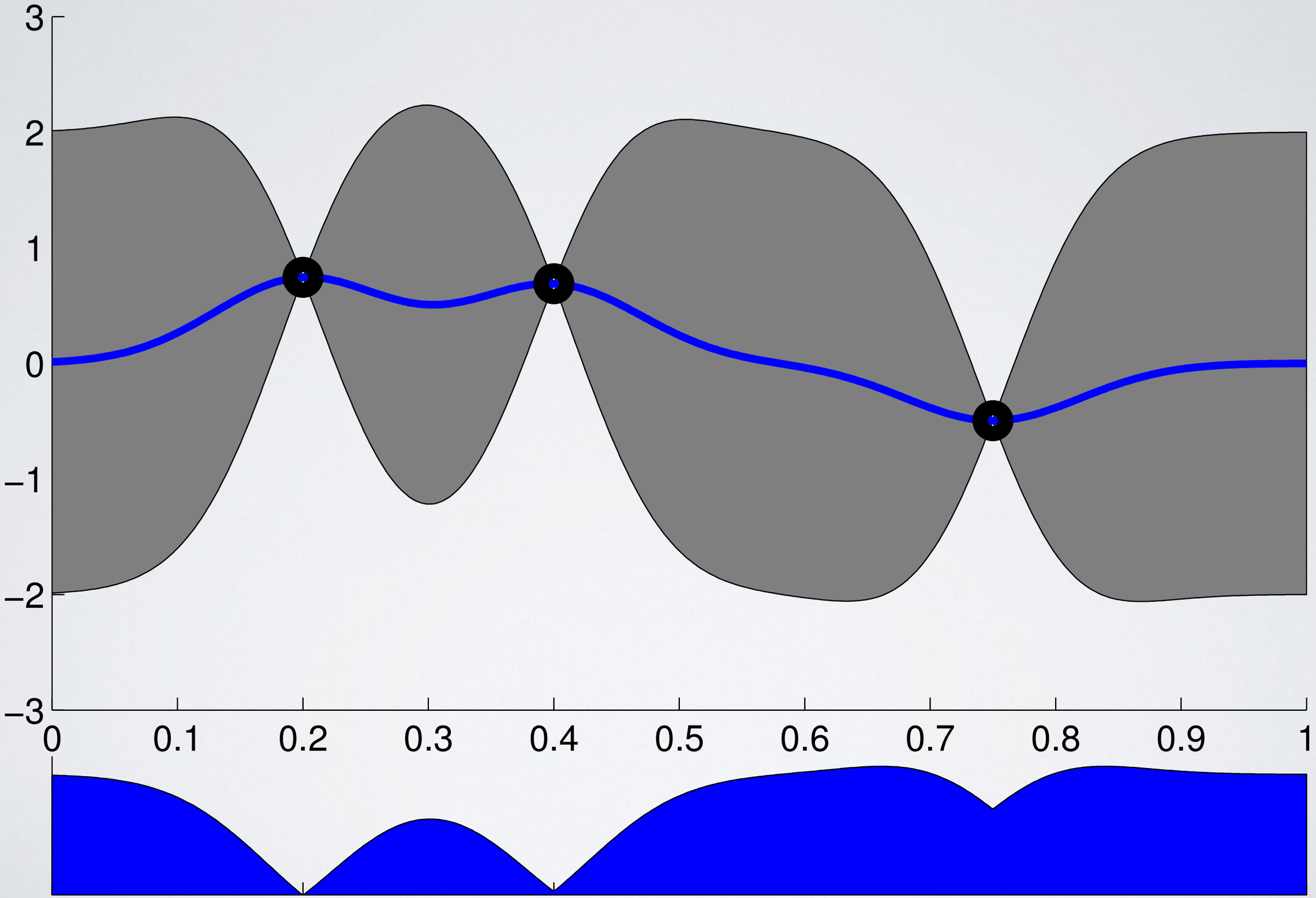
Probability of Improvement



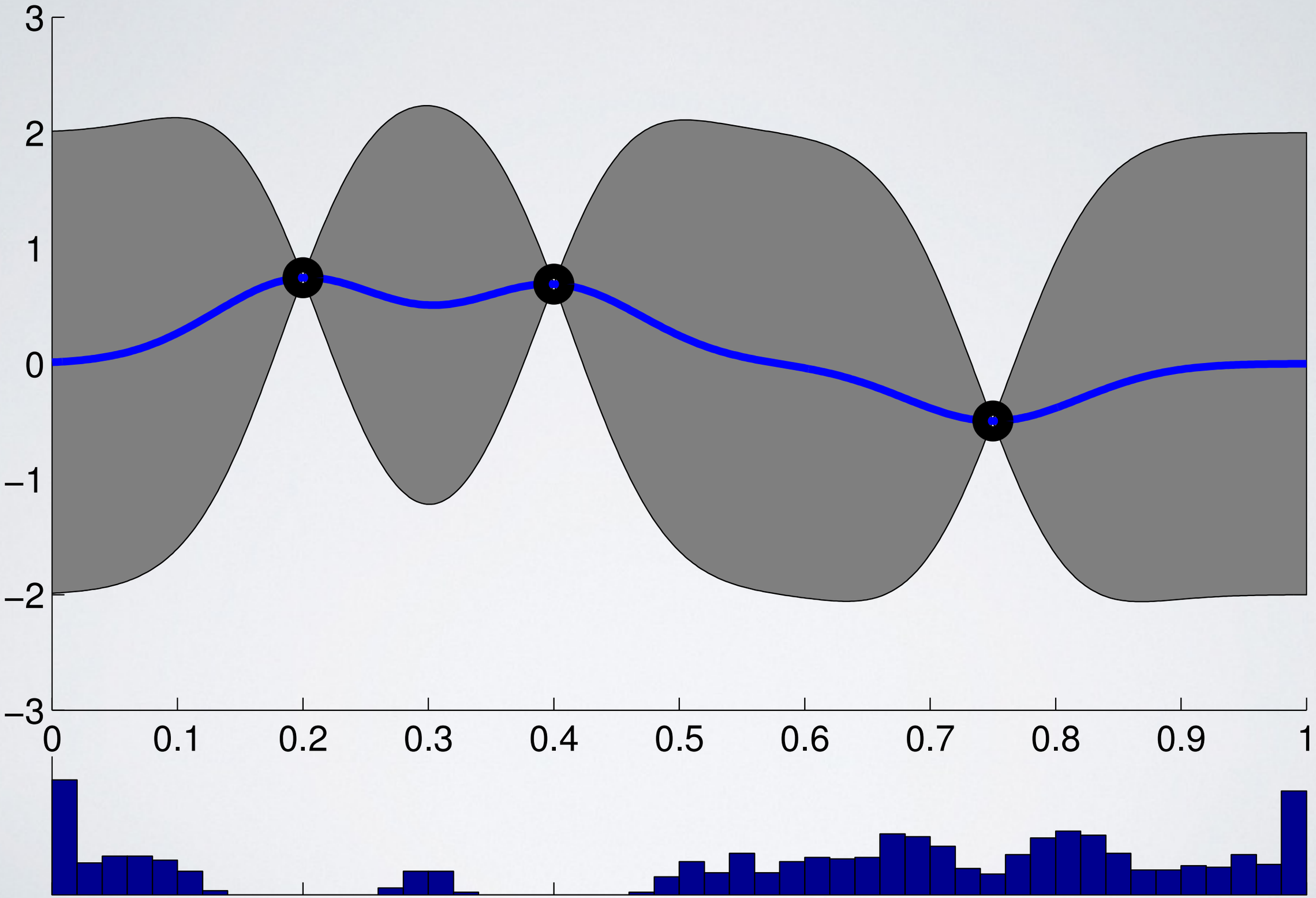
Expected Improvement



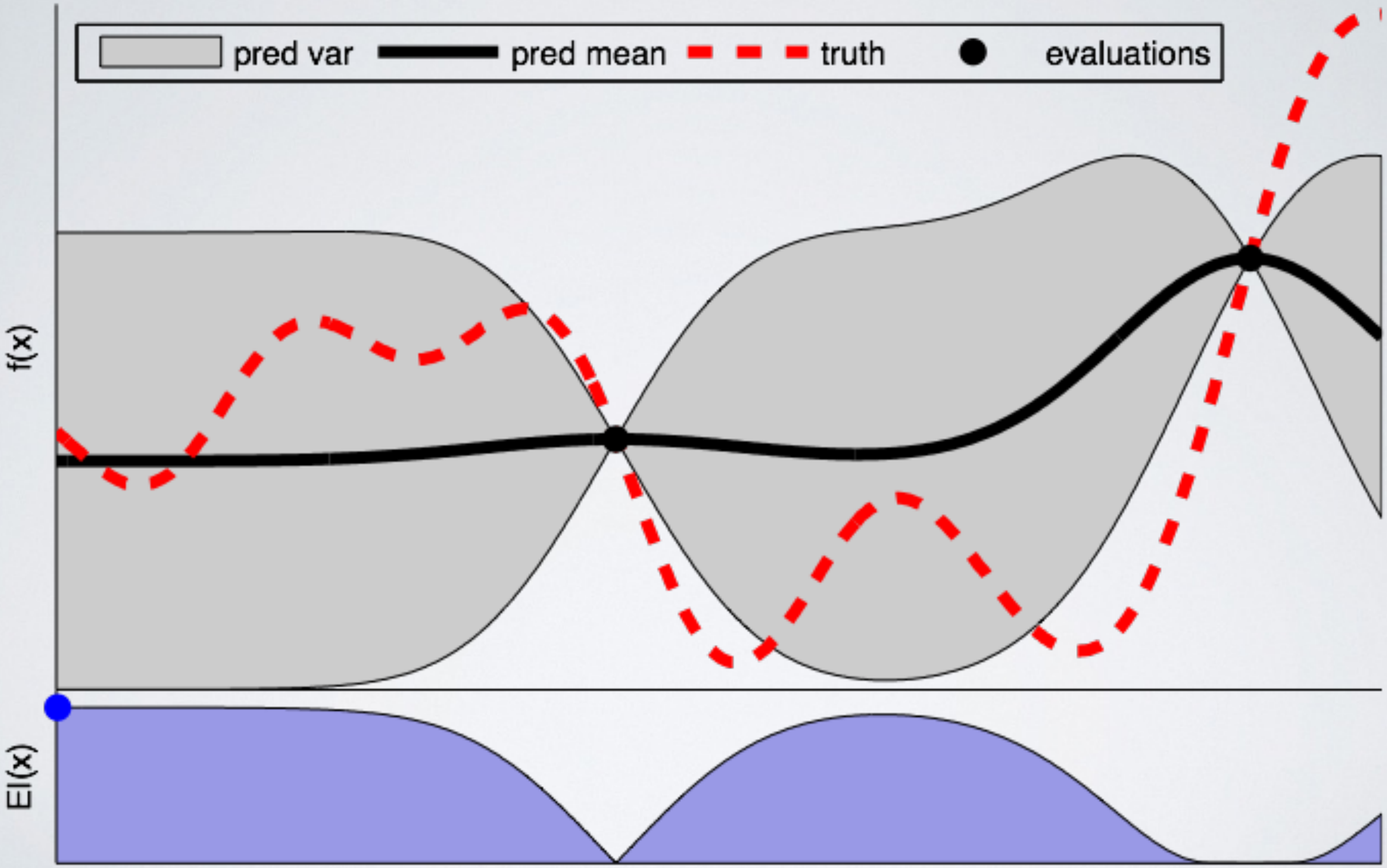
GP Upper (Lower) Confidence Bound



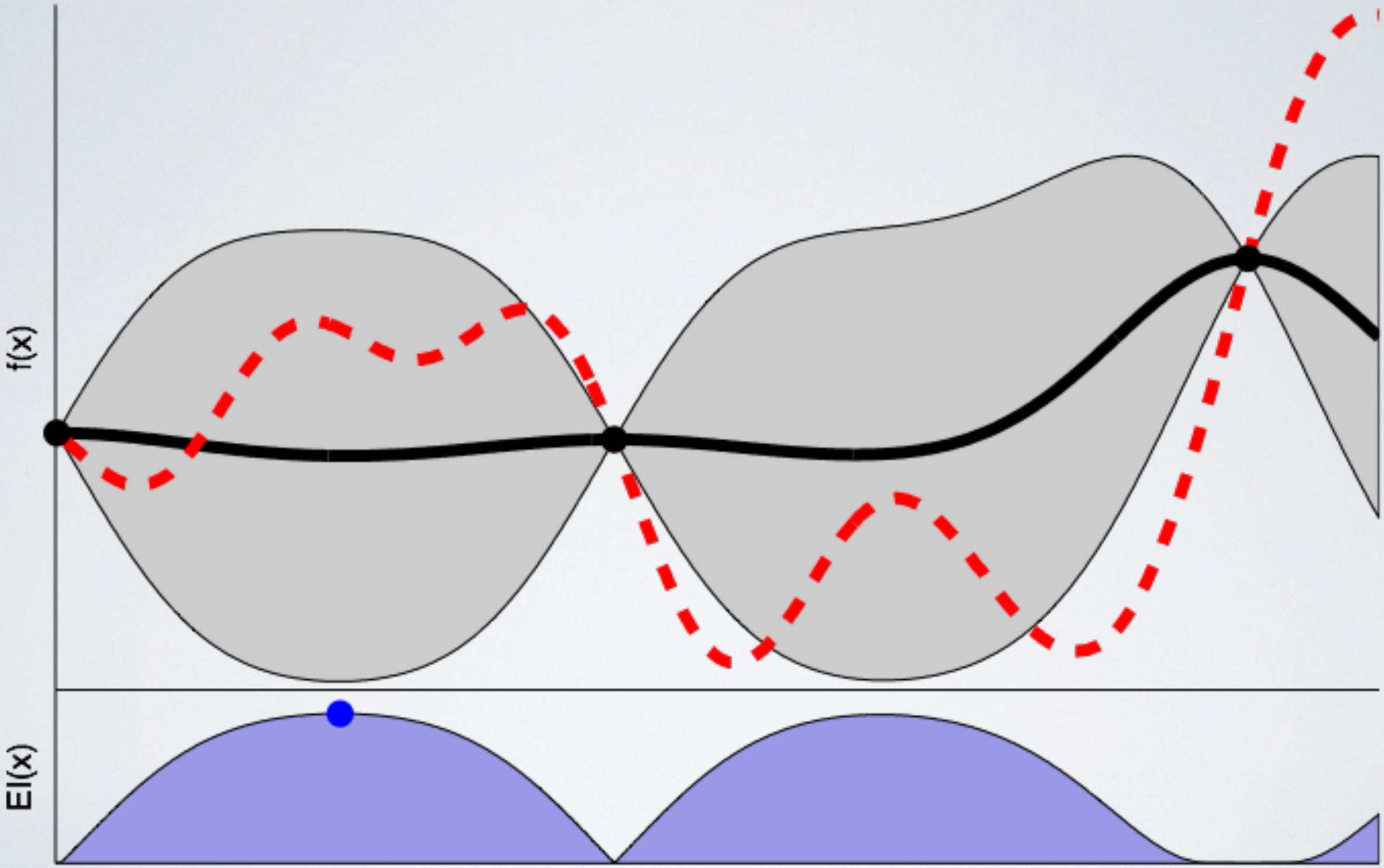
Distribution Over Minimum (Entropy Search)



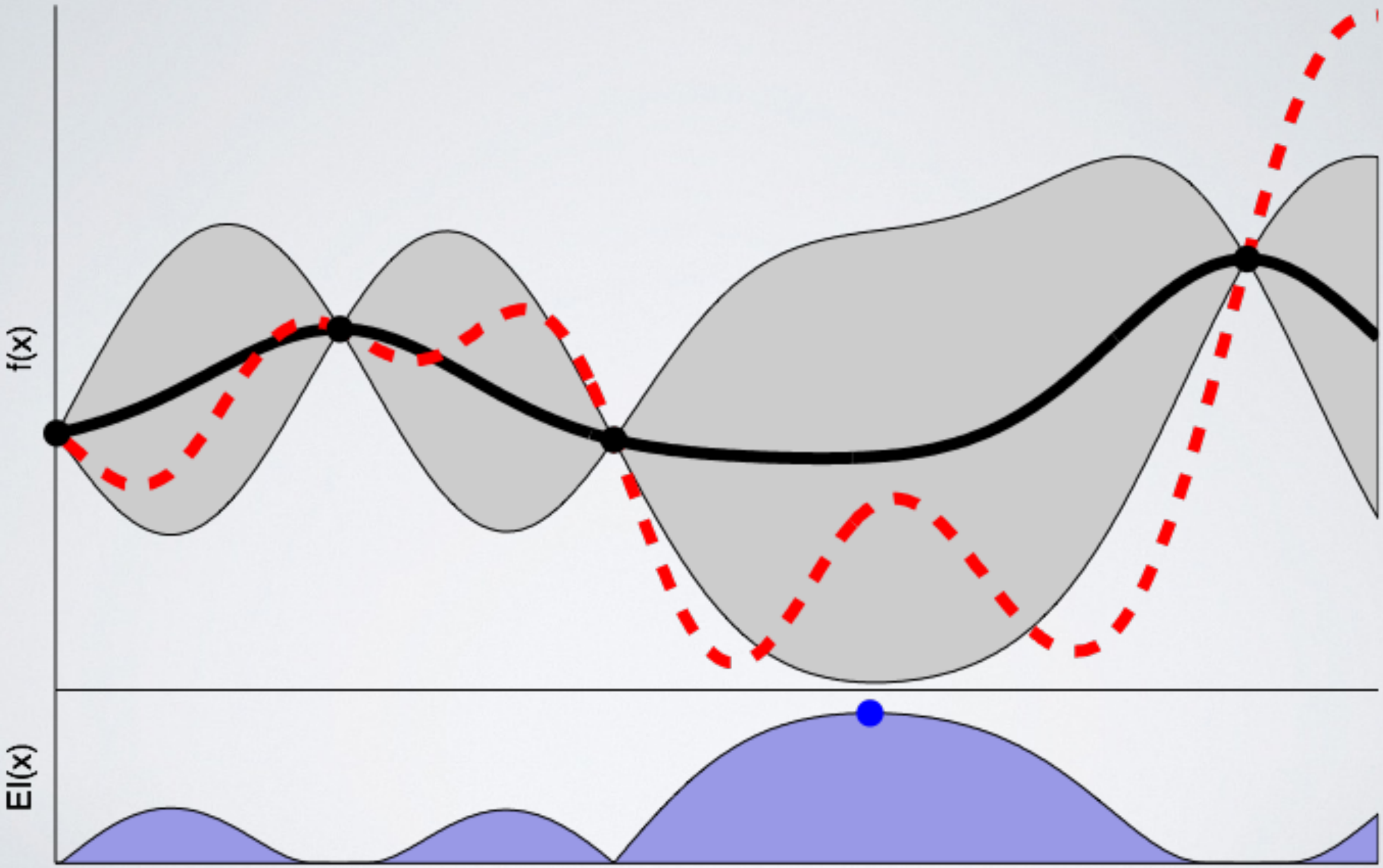
Illustrating Bayesian Optimization



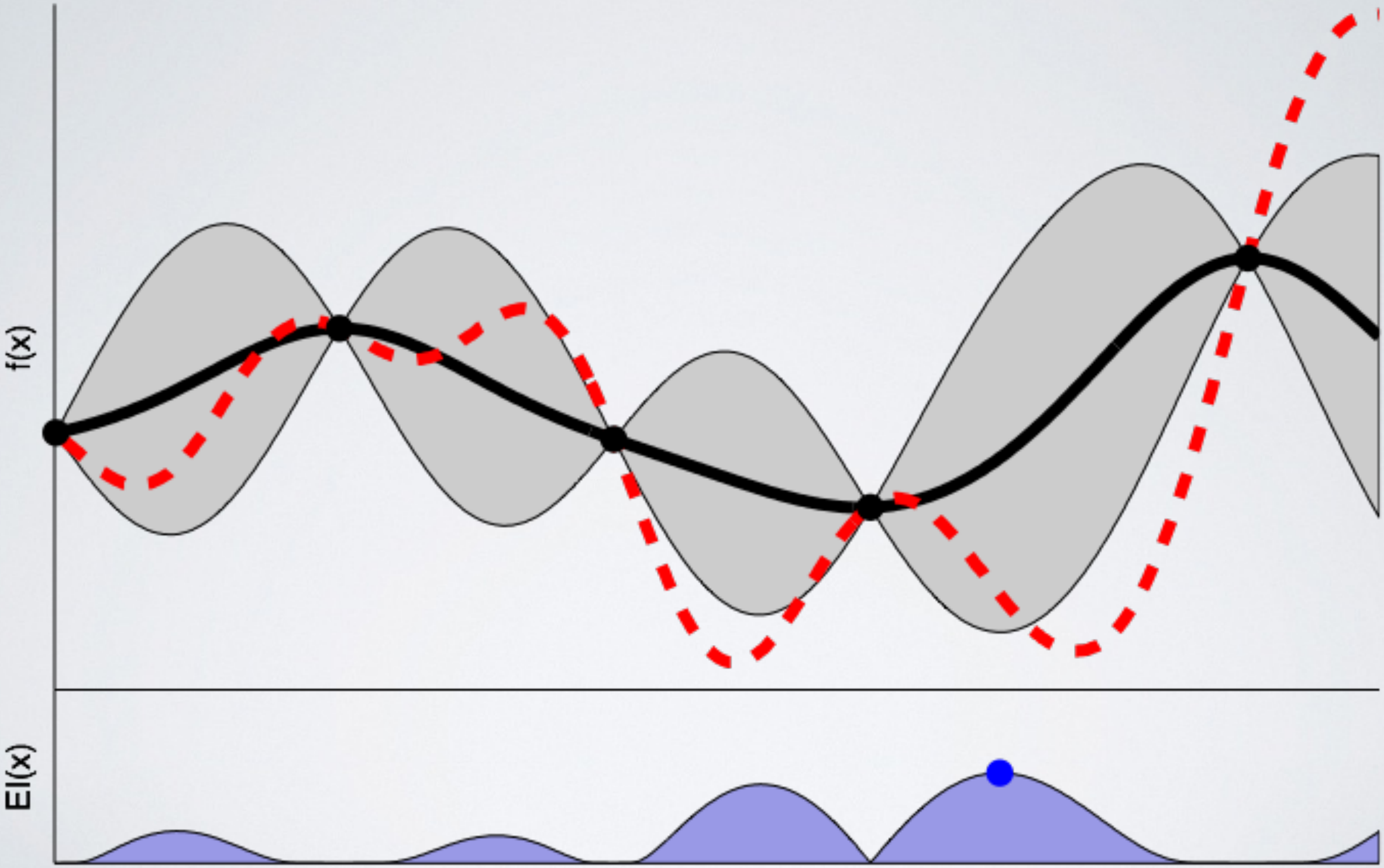
Illustrating Bayesian Optimization



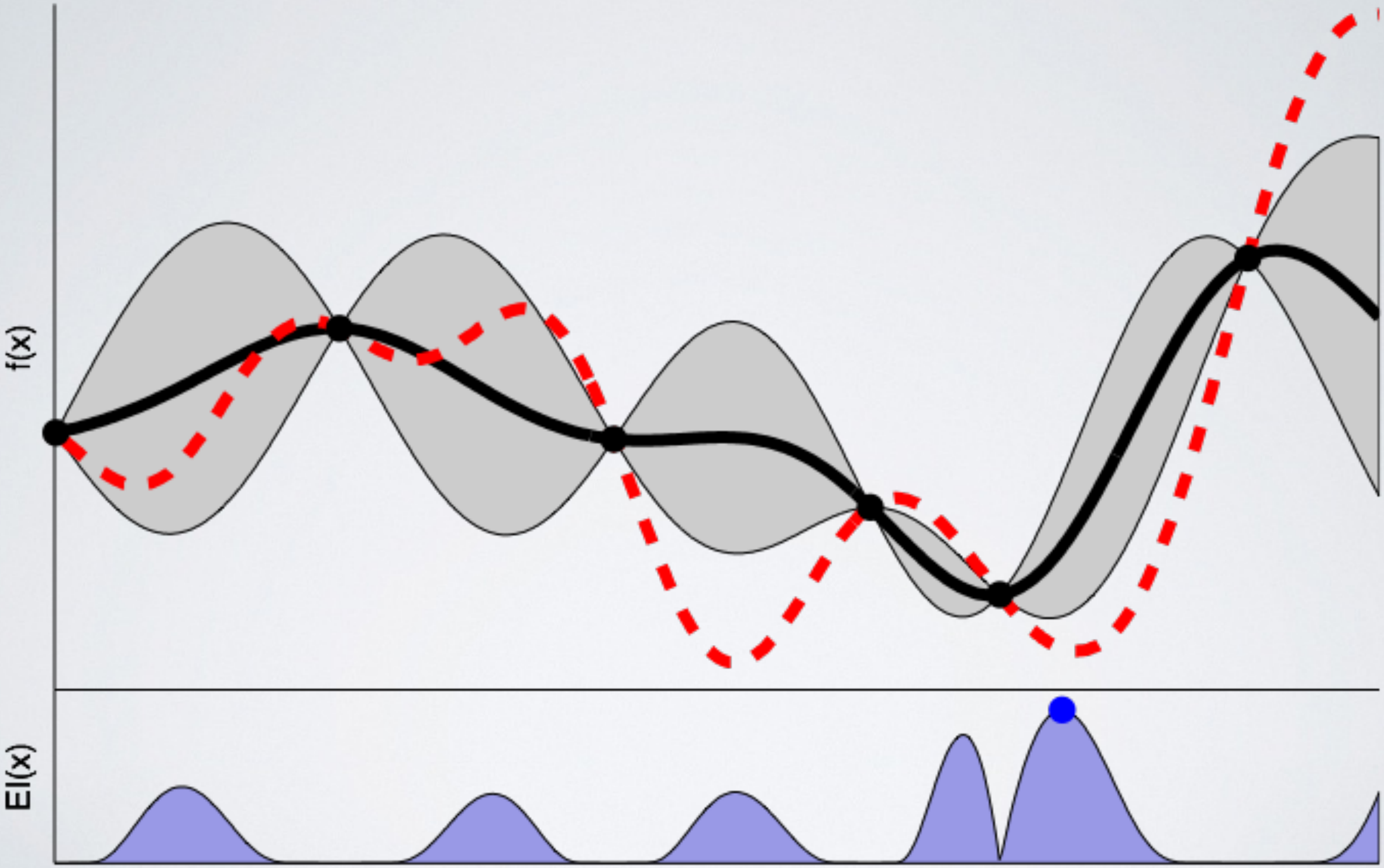
Illustrating Bayesian Optimization



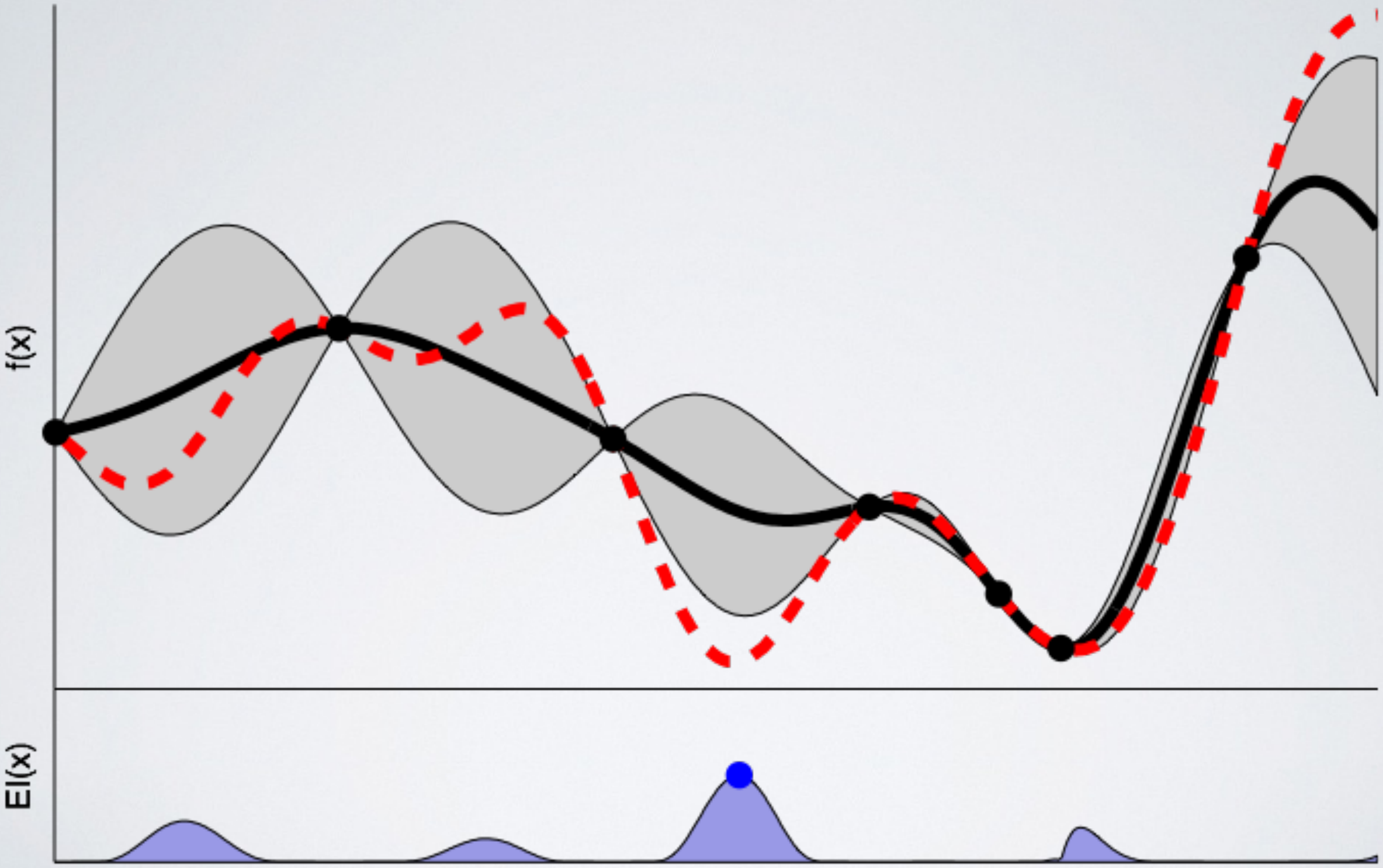
Illustrating Bayesian Optimization



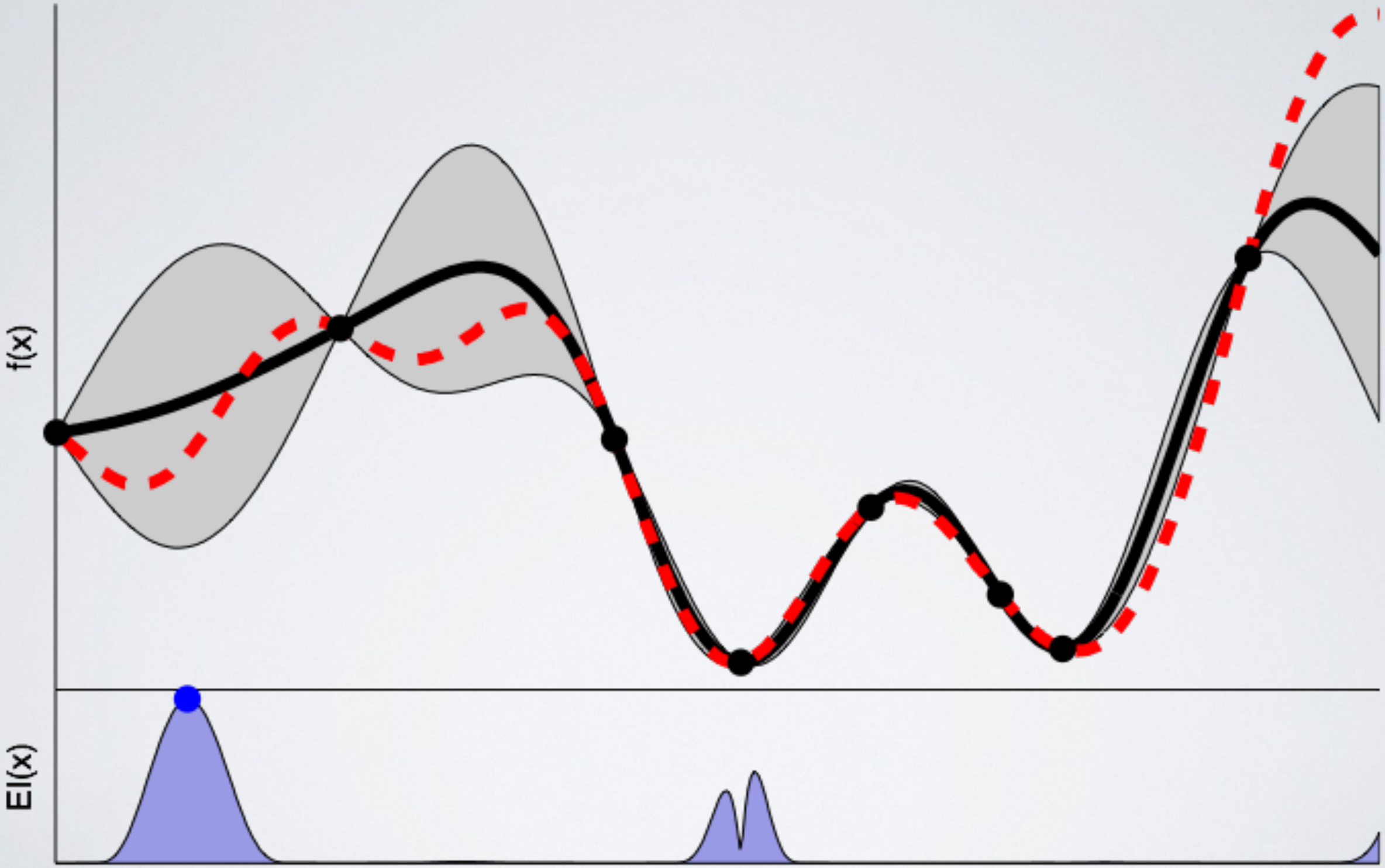
Illustrating Bayesian Optimization



Illustrating Bayesian Optimization



Illustrating Bayesian Optimization



Why Doesn't Everyone Use This?

These ideas have been around for *decades*.
Why is Bayesian optimization in broader use?

- **Fragility and poor default choices.**
  Getting the function model wrong can be catastrophic.
- **There hasn't been standard software available.**
  It's a bit tricky to build such a system from scratch.
- **Experiments are run sequentially.**
  We want to take advantage of cluster computing.
- **Limited scalability in dimensions and evaluations.**
  We want to solve big problems.

Fragility and Poor Default Choices

Ironic Problem:

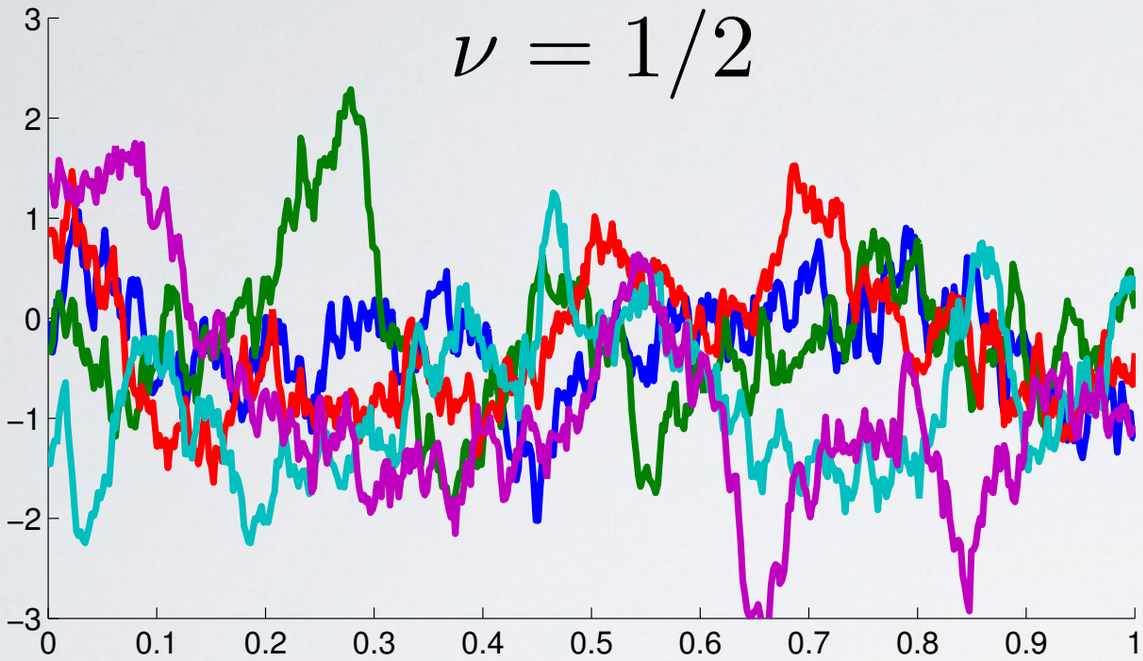
Bayesian optimization has its own hyperparameters!

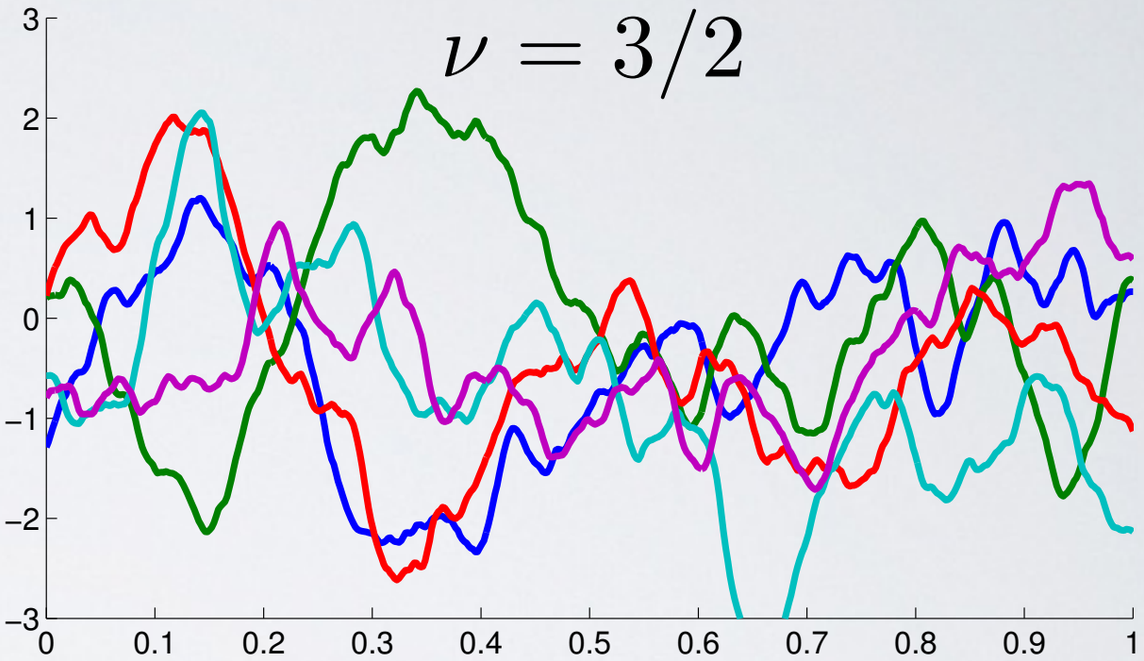
- **Covariance function selection**
  This turns out to be crucial to good performance.
  The default choice for regression is *way* too smooth.
  Instead: use adaptive Matèrn 3/5 kernel.

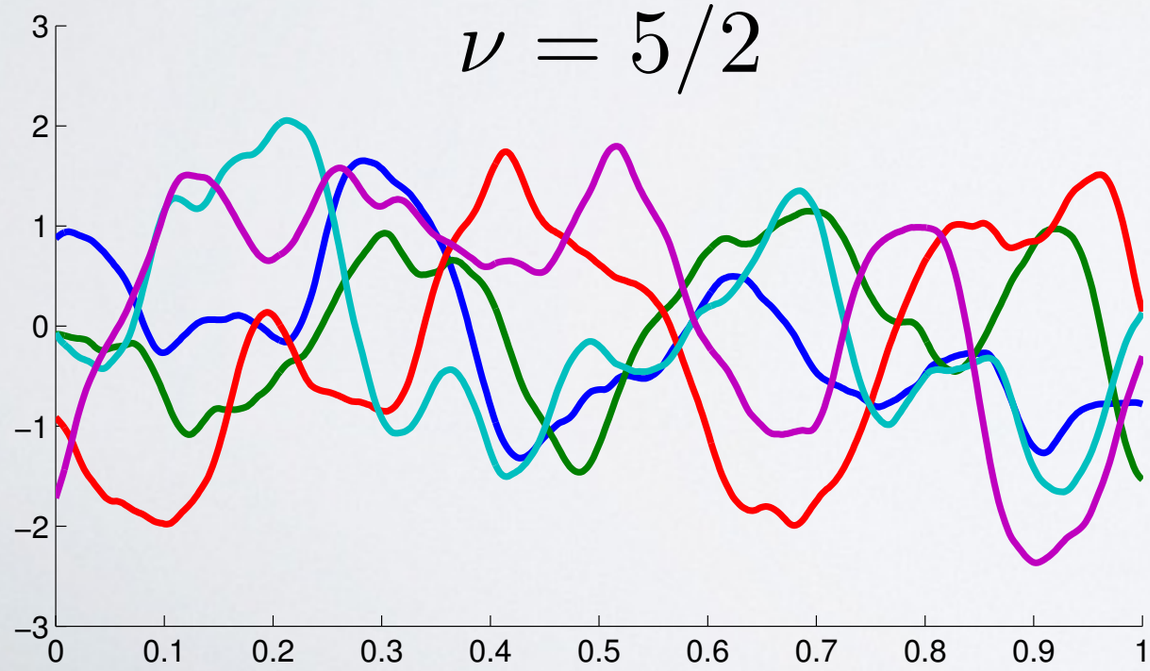
- **Gaussian process hyperparameters**
  Typical empirical Bayes approach can fail horribly.
  Instead: use Markov chain Monte Carlo integration.
  Slice sampling means no additional parameters!

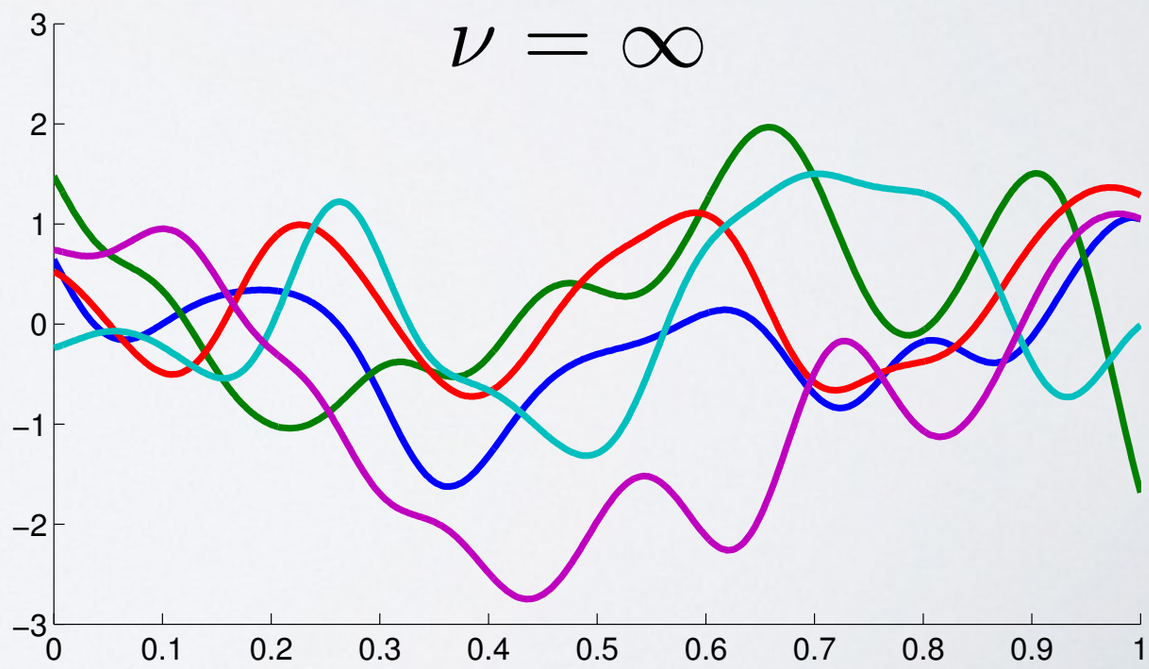
Covariance Function Choice

$$C(r) = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\sqrt{2\nu}\, r}{\ell} \right)^{\nu} K_\nu \left(\frac{\sqrt{2\nu}\, r}{\ell} \right)$$



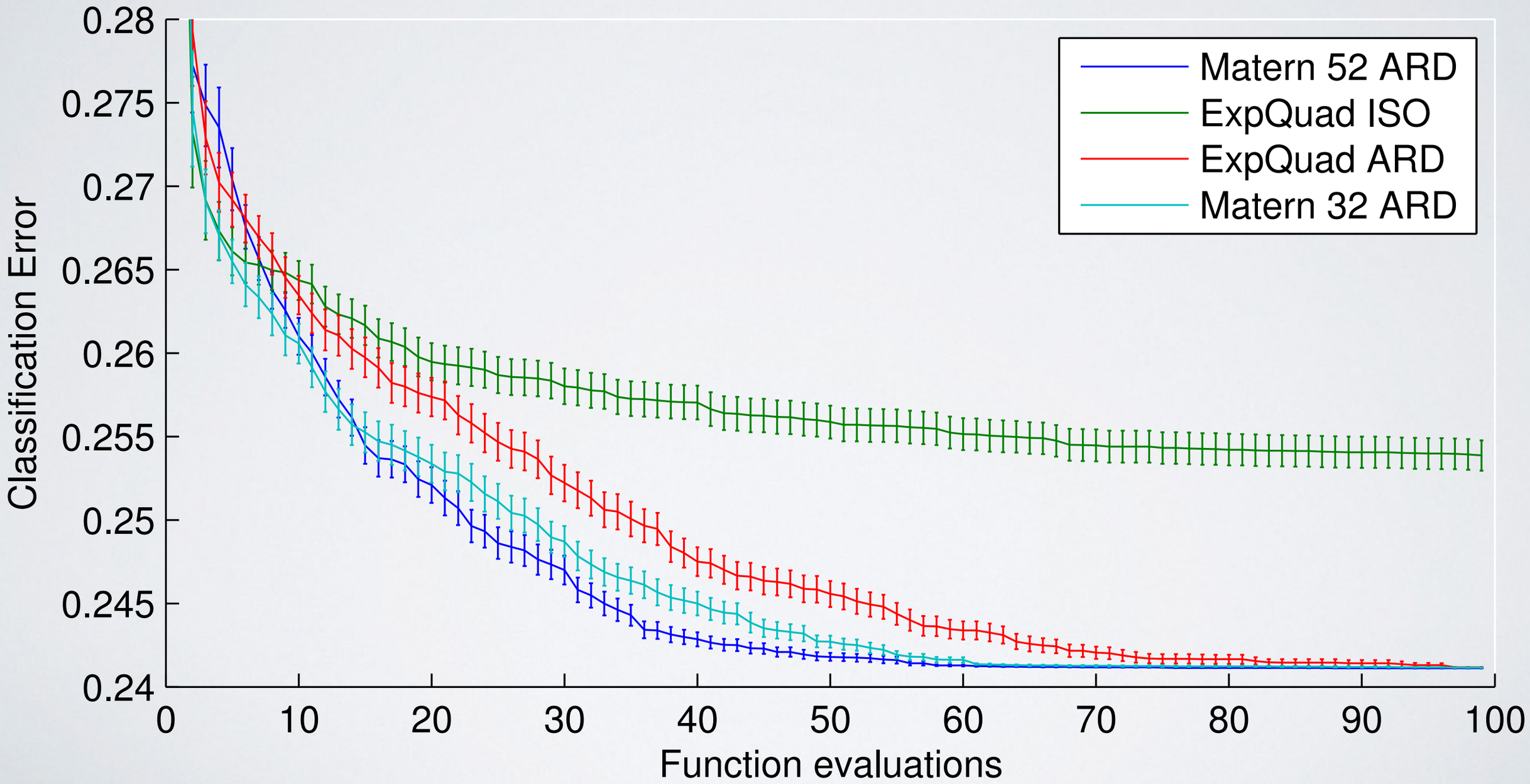






Choosing Covariance Functions

Structured SVM for Protein Motif Finding
Miller et al (2012)



Snoek, Larochelle & RPA, NIPS 2012

MCMC for GP Hyperparameters

- Covariance hyperparameters are often optimized rather than marginalized, typically in the name of convenience and efficiency.
- Slice sampling of hyperparameters (e.g., Murray and Adams 2010) is comparably fast and easy, but accounts for uncertainty in length scale, mean, and amplitude.
- Integrated Acquisition Function:

$$\hat{a}(x) = \int a(x\,;\,\theta)\, p(\theta \,|\, \{x_n, y_n\}_{n=1}^N)\, \mathrm{d}\theta$$

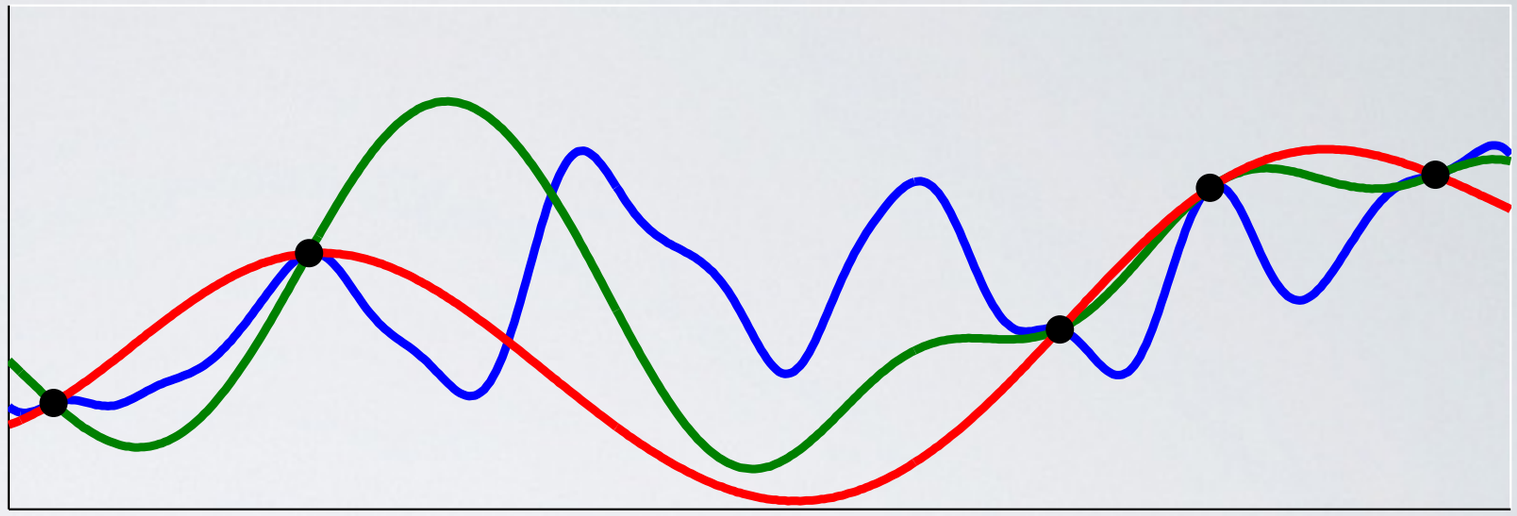
$$\approx \frac{1}{K} \sum_{k=1}^{K} a(x\,;\,\theta^{(k)}) \qquad \theta^{(k)} \sim p(\theta \,|\, \{x_n, y_n\}_{n=1}^N)$$

- For a theoretical discussion of the implications of inferring hyperparameters with BayesOpt, see recent work by Wang and de Freitas (http://arxiv.org/abs/1406.7758)

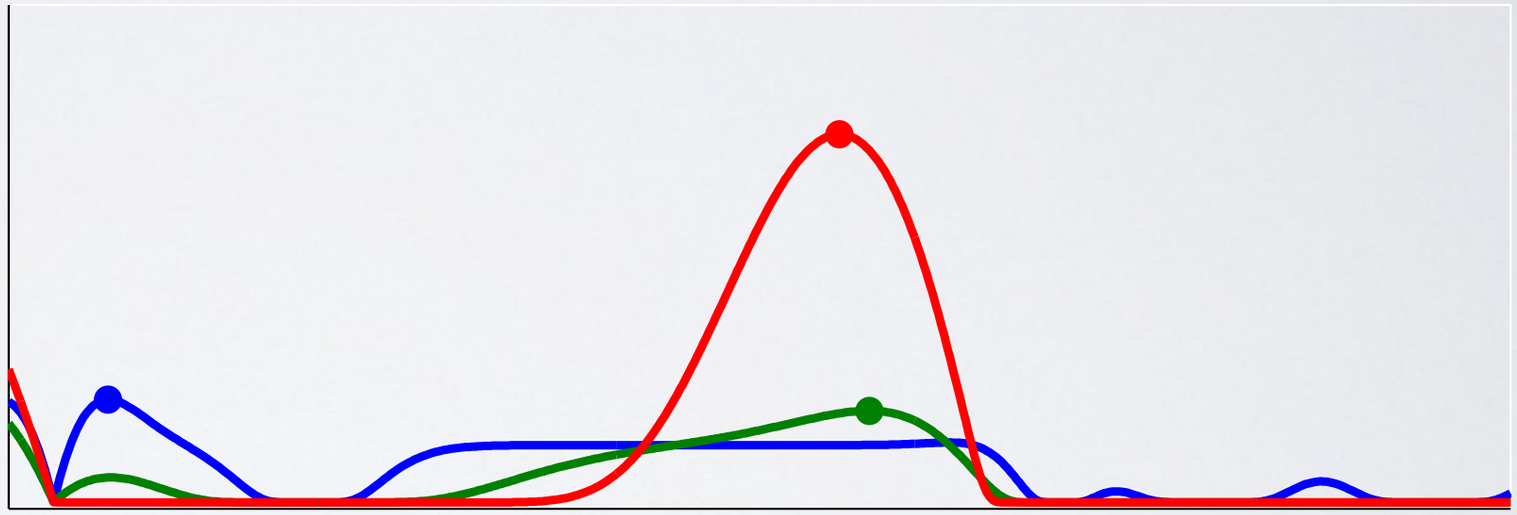
Snoek, Larochelle & RPA, NIPS 2012

Integrating Out GP Hyperparameters

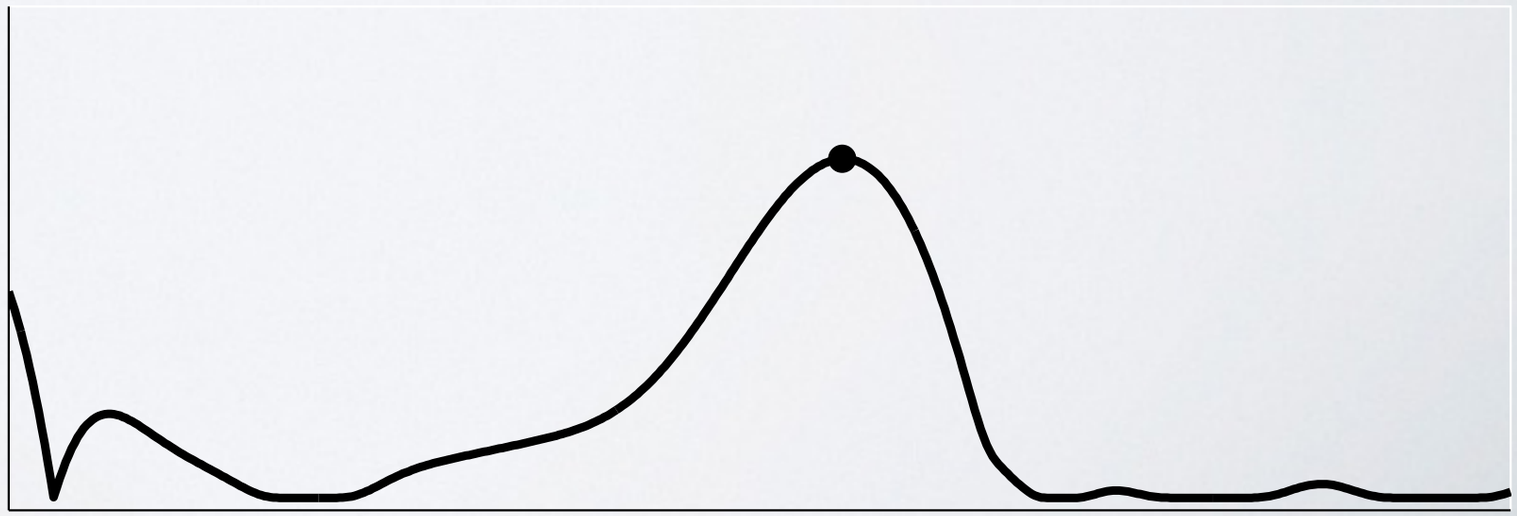
Posterior samples with three different length scales



Length scale specific expected improvement

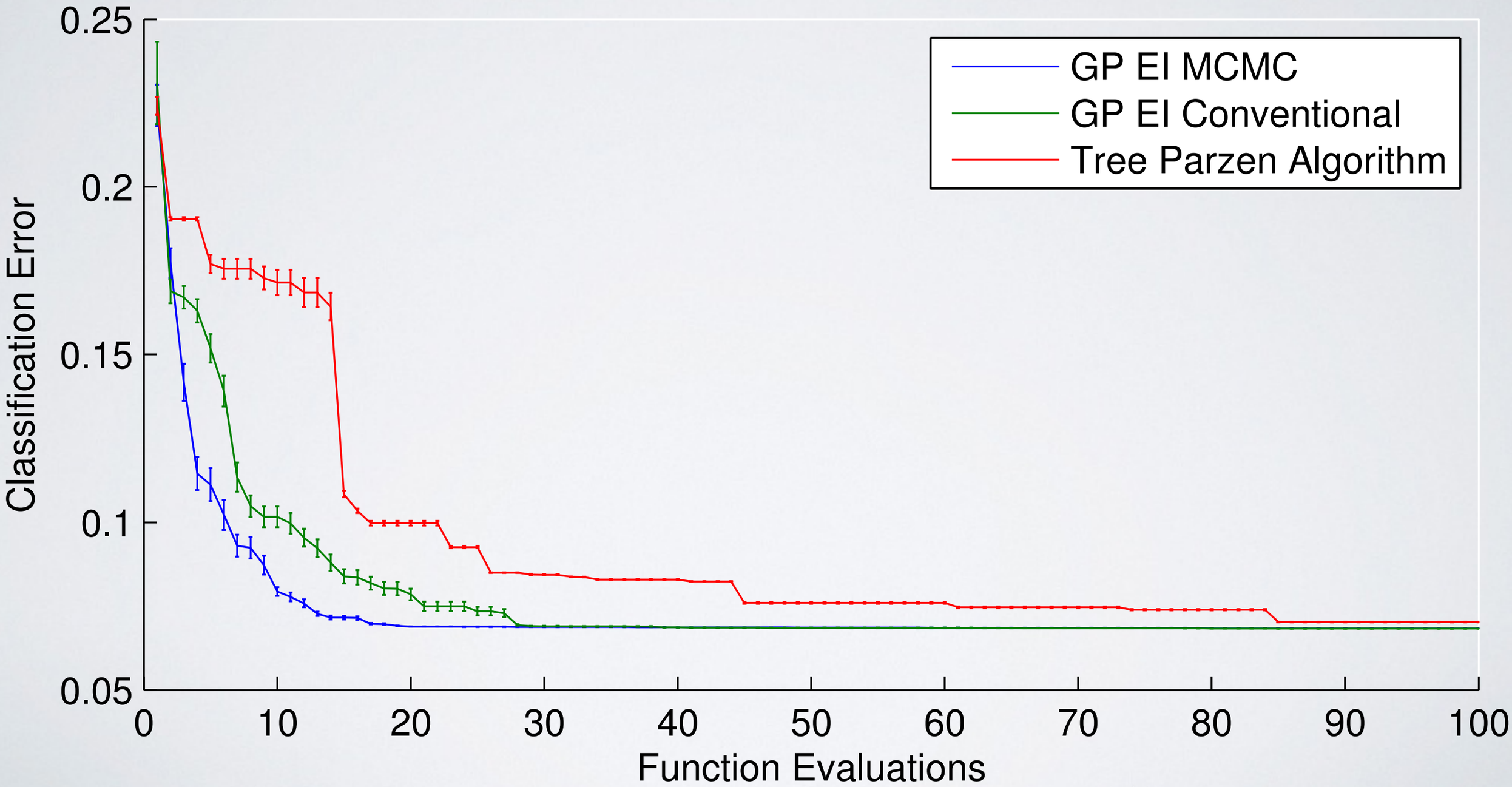


Integrated expected improvement



MCMC for Hyperparameters

Logistic regression for handwritten digit recognition



Snoek, Larochelle & RPA, NIPS 2012

The original source for these slides has advanced topics beyond this point. If interested please reference:

https://www.iro.umontreal.ca/~bengioy/cifar/NCAP2014-summerschool/slides/Ryan_adams_140814_bayesopt_ncap.pdf

Gradient Based Hyperparameter Optimization

Jonathan Lorraine
Winter 2021

Gradient Based HO vs Bayesian Optimization

Gradient Based HO Pros:

- Can often tune as many hyperparameters as model parameters.

Gradient Based HO vs Bayesian Optimization

Gradient Based HO Pros:

Can often tune as many hyperparameters as model parameters.

Can be done online -- i.e., we tune the hyperparameters and model parameters jointly in one run.

Gradient Based HO vs Bayesian Optimization

Gradient Based HO Pros:

Can often tune as many hyperparameters as model parameters.

Can be done online -- i.e., we tune the hyperparameters and model parameters jointly in one run.

But, need a differentiable validation objective and hyperparameter

Gradient Based HO vs Bayesian Optimization

Gradient Based HO Pros:

Can often tune as many hyperparameters as model parameters.

Can be done online -- i.e., we tune the hyperparameters and model parameters jointly in one run.

But, need a differentiable validation objective and hyperparameter

Can behave strangely in some situations -- ex., short horizon bias [1]

[1] Wu, Yuhuai, et al. "Understanding short-horizon bias in stochastic meta-optimization." *arXiv preprint arXiv:1803.02021* (2018).

Honorable Mention -- Graduate Student Descent

I.e., a person sitting there and tuning it.

Pros:

- Can tune hyperparameters online with complicated schedules
- Can save model and good checkpoints and try different methods.
- Can use complicated intuitions about how the models should work
- Can tune non-differentiable objectives (ex., accuracy) or even mixtures of objectives
- Cab tune non-differentiable hyperparameters (ex., architecture)
- Sometimes interpretable solutions
- Can account for uncertainty
- Can account for variable computational budgets
- Transfer learning from other problems you've seen

Hyperparameter Optimization is bi-level optimization

These methods apply to bi-level optimization more generally

How approximate w*?

$$\boldsymbol{\lambda}^* := \arg\min_{\boldsymbol{\lambda}} \mathcal{L}_V^*(\boldsymbol{\lambda}) \text{ where} \tag{1}$$

$$\mathcal{L}_V^*(\boldsymbol{\lambda}) := \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})) \text{ and } \mathbf{w}^*(\boldsymbol{\lambda}) := \arg\min_{\mathbf{w}} \mathcal{L}_T(\boldsymbol{\lambda}, \mathbf{w}) \tag{2}$$

$$\underbrace{\frac{\partial \mathcal{L}_V^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{hypergradient}} = \left(\frac{\partial \mathcal{L}_V}{\partial \boldsymbol{\lambda}} + \frac{\partial \mathcal{L}_V}{\partial \mathbf{w}} \frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}} \right) \Bigg|_{\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})} =$$

$$\underbrace{\frac{\partial \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \boldsymbol{\lambda}}}_{\text{hyperparam direct grad.}} + \overbrace{\underbrace{\frac{\partial \mathcal{L}_V(\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda}))}{\partial \mathbf{w}^*(\boldsymbol{\lambda})}}_{\text{parameter direct grad.}} \times \underbrace{\frac{\partial \mathbf{w}^*(\boldsymbol{\lambda})}{\partial \boldsymbol{\lambda}}}_{\text{best-response Jacobian}}}^{\text{hyperparam indirect grad.}} \tag{3}$$

Kinds of Gradient Based HO

$$\boldsymbol{\lambda}^* \approx \arg\min_{\boldsymbol{\lambda}} \mathcal{L}_V(\boldsymbol{\lambda}, \hat{\mathbf{w}}_\phi(\boldsymbol{\lambda}))$$

Amortized gradient / evolutionary methods -- [2], [3], others

Iterative differentiation (ITD) based methods -- [4], [5], others

$$\theta_D^0 = \theta_D$$

$$\theta_D^{k+1} = \theta_D^k + \eta^k \frac{\mathrm{d}f\left(\theta_G, \theta_D^k\right)}{\mathrm{d}\theta_D^k}$$

$$\theta_D^*\left(\theta_G\right) = \lim_{k\to\infty} \theta_D^k,$$

Approximate implicit differentiation (AID) methods -- [6], [7], others

Theorem 1 (Cauchy, Implicit Function Theorem). *If for some* $(\boldsymbol{\lambda}', \mathbf{w}')$, $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}', \mathbf{w}'} = 0$ *and regularity conditions are satisfied, then surrounding* $(\boldsymbol{\lambda}', \mathbf{w}')$ *there is a function* $\mathbf{w}^*(\boldsymbol{\lambda})$ *s.t.* $\frac{\partial \mathcal{L}_T}{\partial \mathbf{w}}|_{\boldsymbol{\lambda}, \mathbf{w}^*(\boldsymbol{\lambda})} = 0$ *and we have:*

$$\left.\frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\lambda}}\right|_{\boldsymbol{\lambda}'} = -\underbrace{\left[\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \mathbf{w}}\right]^{-1}}_{\text{training Hessian}} \times \underbrace{\frac{\partial^2 \mathcal{L}_T}{\partial \mathbf{w} \partial \boldsymbol{\lambda}}}_{\text{training mixed partials}} \Bigg|_{\boldsymbol{\lambda}', \mathbf{w}^*(\boldsymbol{\lambda}')} \quad \text{(IFT)}$$

ITD and AID based methods can easily scale to as many hyperparameters as model parameters.

Citations

[2] Lorraine, Jonathan, and David Duvenaud. "Stochastic hyperparameter optimization through hypernetworks." *arXiv preprint arXiv:1802.09419* (2018).

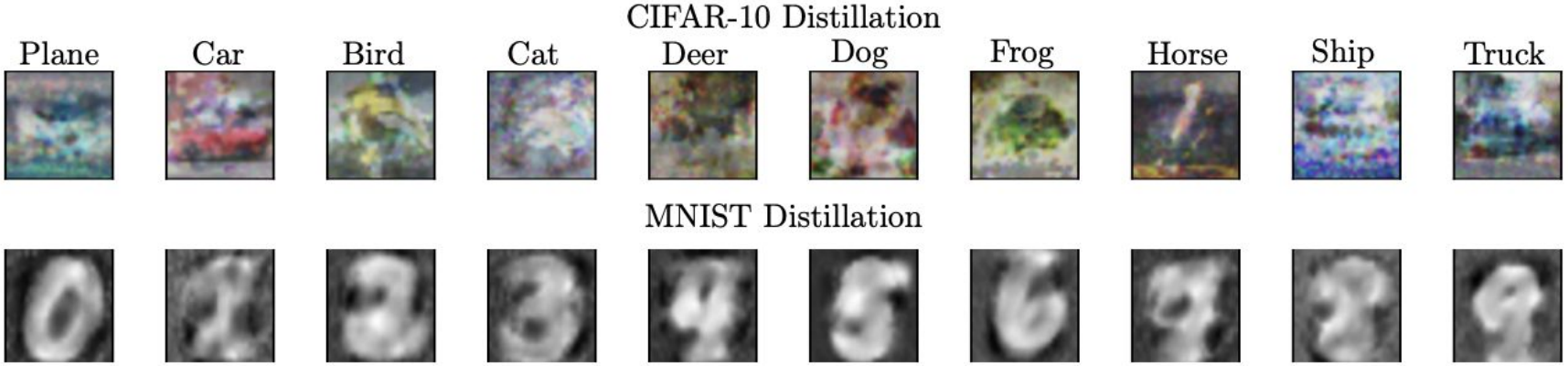
[3] MacKay, Matthew, et al. "Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions." *arXiv preprint arXiv:1903.03088* (2019).

[4] Maclaurin, Dougal, David Duvenaud, and Ryan Adams. "Gradient-based hyperparameter optimization through reversible learning." *International conference on machine learning*. PMLR, 2015.

[5] Domke, Justin. Generic methods for optimization-based modeling. In International Conference on Artificial Intelligence and Statistics, pp. 318–326, 2012.

[6] Lorraine, Jonathan, Paul Vicol, and David Duvenaud. "Optimizing millions of hyperparameters by implicit differentiation." *International Conference on Artificial Intelligence and Statistics*. PMLR, 2020.

[7] Pedregosa, Fabian. "Hyperparameter optimization with approximate gradient." *International conference on machine learning*. PMLR, 2016.



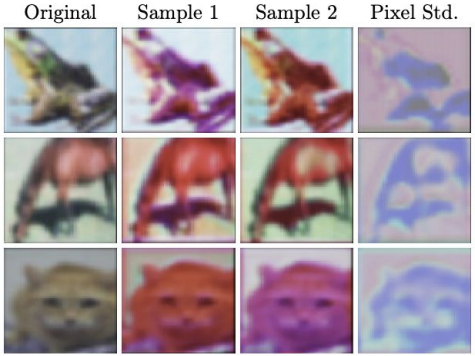


Figure 7: Learned data augmentations. The original image is on the left, followed by two augmented samples and the standard deviation of the pixel intensities from the augmentation distribution.

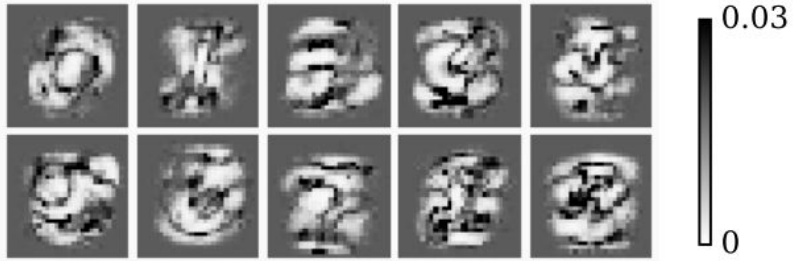
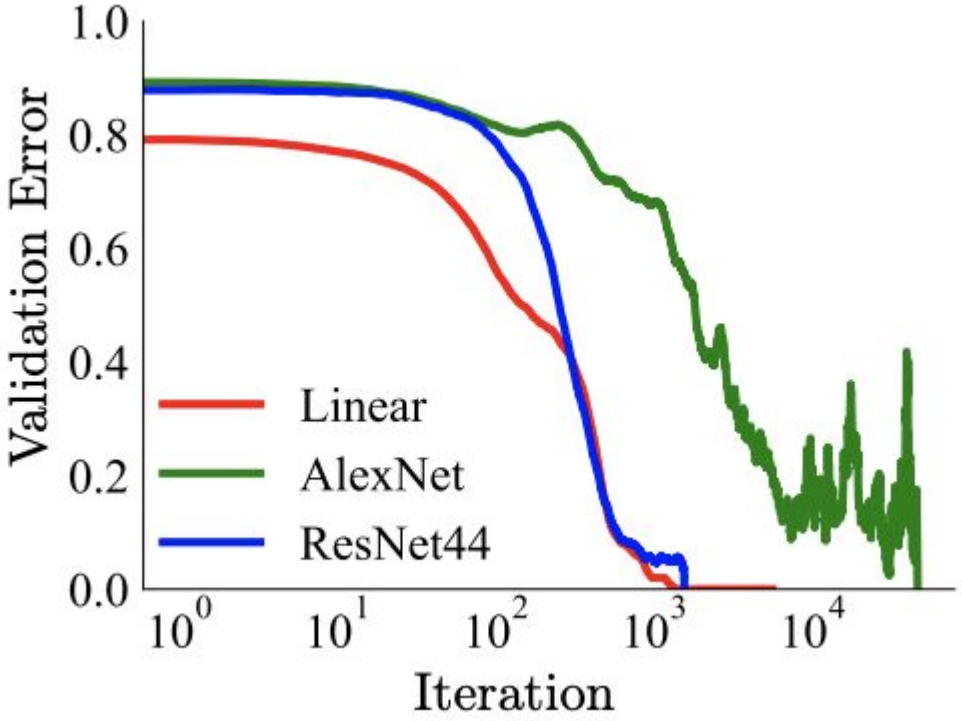


Figure 6. Optimized L_2 regularization hyperparameters for each weight in a logistic regression trained on MNIST. The weights corresponding to each output label (0 through 9 respectively) have been rendered separately. High values (black) indicate strong regularization.

We can overfit the validation set - careful!



Think about validation partition?

Re-training is critical for performance

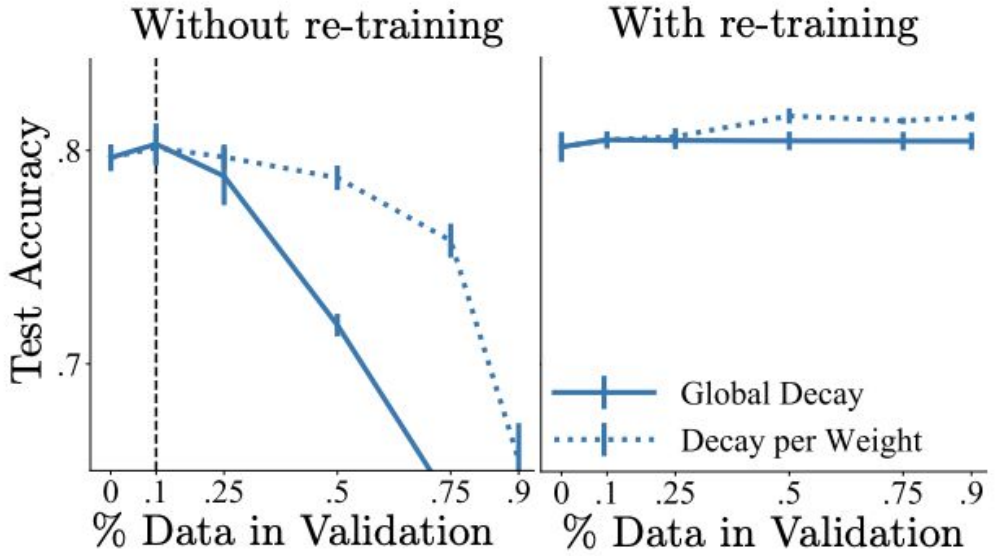


Figure 9: Test accuracy of logistic regression on MNIST, with different size validation splits. Solid lines correspond to a single global weight decay (1 hyperparameter), while dotted lines correspond to a separate weight decay per weight (many hyperparameters). The best validation proportion for test performance is different after re-training for many hyperparameters, but similar for few hyperparameters.