CSC321 Lecture 21: Bayesian Hyperparameter Optimization

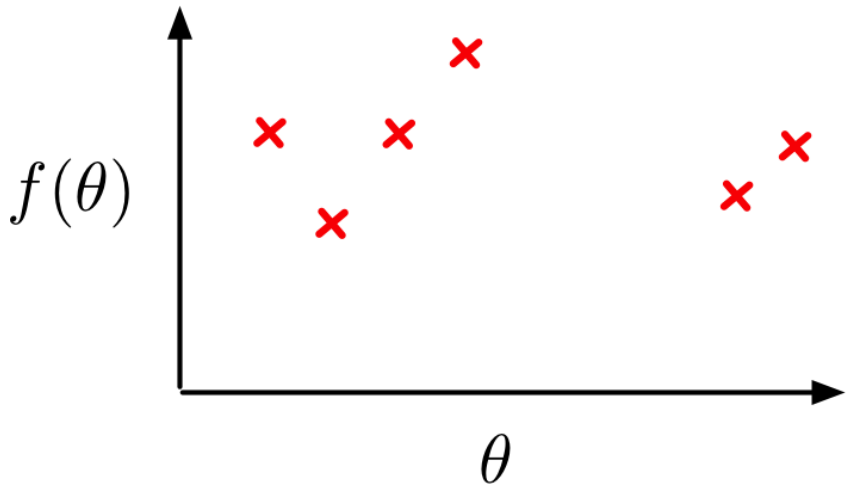
Roger Grosse

Overview

- Today's lecture: a neat application of Bayesian parameter estimation to automatically tuning hyperparameters
- Recall that neural nets have certain hyperparmaeters which aren't part of the training procedure
  - E.g. number of units, learning rate, L_2 weight cost, dropout probability
- You can evaluate them using a validation set, but there's still the problem of which values to try
  - Brute force search (e.g. grid search, random search) is very expensive, and wastes time trying silly hyperparameter configurations

Overview

- Hyperparamter tuning is a kind of black-box optimization: you want to minimize a function $f(\theta)$, but you only get to query values, not compute gradients
  - Input θ: a configuration of hyperparameters
  - Function value $f(\theta)$: error on the validation set
- Each evaluation is expensive, so we want to use few evaluations.
- Suppose you've observed the following function values. Where would you try next?



Overview

- You want to query a point which:
  - you expect to be good
  - you are uncertain about
- How can we model our uncertainty about the function?
- Bayesian regression lets us predict not just a value, but a distribution. That's what the first half of this lecture is about.

Linear Regression as Maximum Likelihood

- Recall linear regression:

$$y = \mathbf{w}^\top \mathbf{x} + b$$

$$\mathcal{L}(y, t) = \frac{1}{2}(t - y)^2$$

- This has a probabilistic interpretation, where the targets are assumed to be a linear function of the inputs, plus Gaussian noise:

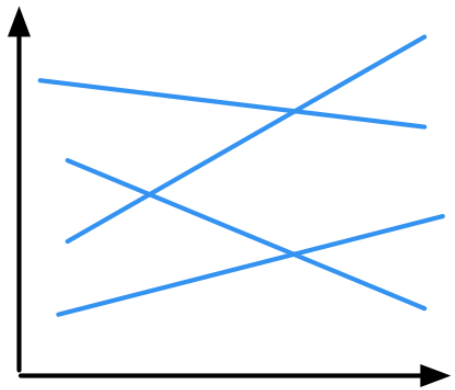
$$t \,|\, \mathbf{x} \sim \mathcal{N}(\mathbf{w}^\top \mathbf{x} + b,\ \sigma^2)$$

- Linear regression is just maximum likelihood under this model:

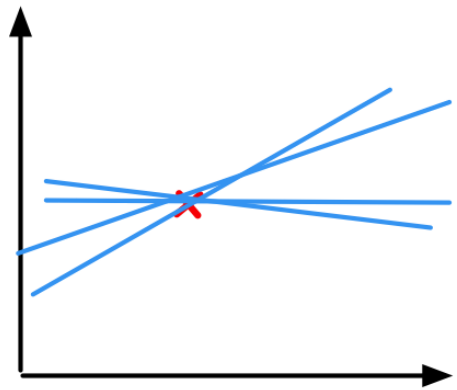
$$\begin{aligned}
\frac{1}{N}\sum_{i=1}^{N} \log p(t^{(i)} \,|\, \mathbf{x}^{(i)}; \mathbf{w}, b) &= \frac{1}{N}\sum_{i=1}^{N} \log \mathcal{N}(t^{(i)}; \mathbf{w}^\top \mathbf{x} + b, \sigma^2) \\
&= \frac{1}{N}\sum_{i=1}^{N} \log \left[\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(t^{(i)} - \mathbf{w}^\top \mathbf{x} - b)^2}{2\sigma^2}\right)\right] \\
&= \text{const} - \frac{1}{2N\sigma^2}\sum_{i=1}^{N}(t^{(i)} - \mathbf{w}^\top \mathbf{x} - b)^2
\end{aligned}$$

Bayesian Linear Regression

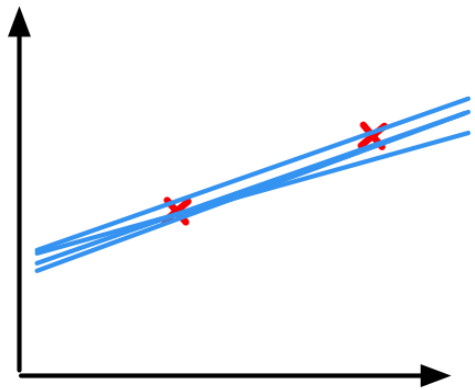
- We're interested in the uncertainty
- Bayesian linear regression considers various plausible explanations for how the data were generated.
- It makes predictions using all possible regression weights, weighted by their posterior probability.



no observations



one observation



two observations

Bayesian Linear Regression

- Leave out the bias for simplicity
- **Prior distribution:** a broad, spherical (multivariate) Gaussian centered at zero:

$$\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \nu^2 \mathbf{I})$$

- **Likelihood:** same as in the maximum likelihood formulation:

$$t \mid \mathbf{x}, \mathbf{w} \sim \mathcal{N}(\mathbf{w}^\top \mathbf{x},\ \sigma^2)$$

- **Posterior:**

$$\begin{aligned}
\log p(\mathbf{w} \mid \mathcal{D}) &= \text{const} + \log p(\mathbf{w}) + \sum_{i=1}^{N} \log p(t^{(i)} \mid \mathbf{w}, \mathbf{x}^{(i)}) \\
&= \text{const} + \log \mathcal{N}(\mathbf{w}; \mathbf{0}, \nu^2 \mathbf{I}) + \sum_{i=1}^{N} \log \mathcal{N}(t^{(i)}; \mathbf{w}^\top \mathbf{x}^{(i)}, \sigma) \\
&= \text{cost} - \frac{1}{2\nu^2} \mathbf{w}^\top \mathbf{w} - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (t^{(i)} - \mathbf{w}^\top \mathbf{x}^{(i)})^2
\end{aligned}$$

Bayesian Linear Regression

- Posterior distribution in the univariate case:

$$\log p(w \mid \mathcal{D}) = \text{const} - \frac{1}{2\nu^2} w^2 - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (t^{(i)} - w x^{(i)})^2$$

$$= \text{const} - \frac{1}{2} \left(\frac{1}{\nu^2} + \frac{1}{\sigma^2} \sum_{i=1}^{N} [x^{(i)}]^2 \right) w^2 + \left(\frac{1}{\sigma^2} \sum_{i=1}^{N} x^{(i)} t^{(i)} \right) w$$

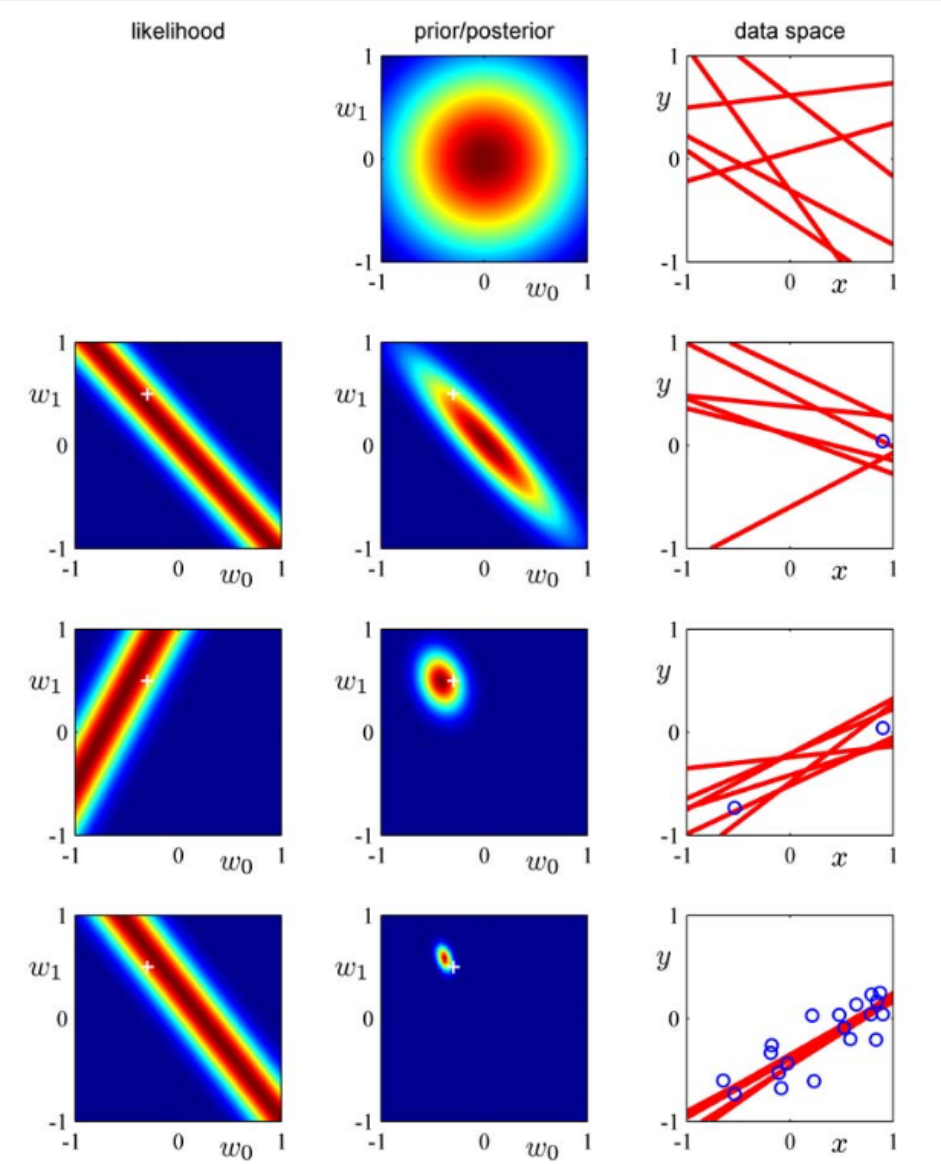
- This is a Gaussian distribution with

$$\mu_{\text{post}} = \frac{\frac{1}{\sigma^2} \sum_{i=1}^{N} x^{(i)} t^{(i)}}{\frac{1}{\nu^2} + \frac{1}{\sigma^2} \sum_{i=1}^{N} [x^{(i)}]^2}$$

$$\sigma^2_{\text{post}} = \frac{1}{\frac{1}{\nu^2} + \frac{1}{\sigma^2} \sum_{i=1}^{N} [x^{(i)}]^2}$$

- The formula for μ_{post} is basically the same as Homework 5, Question 1
- The posterior in the multivariate case is a multivariate Gaussian. The derivation is analogous, but with some linear algebra.

Bayesian Linear Regression

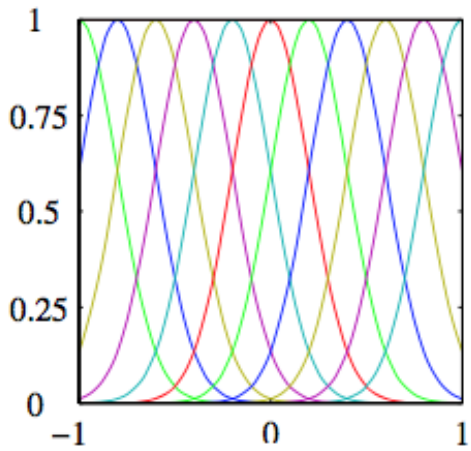


— Bishop, Pattern Recognition and Machine Learning

Bayesian Linear Regression

- We can turn this into nonlinear regression using basis functions.
- E.g., Gaussian basis functions

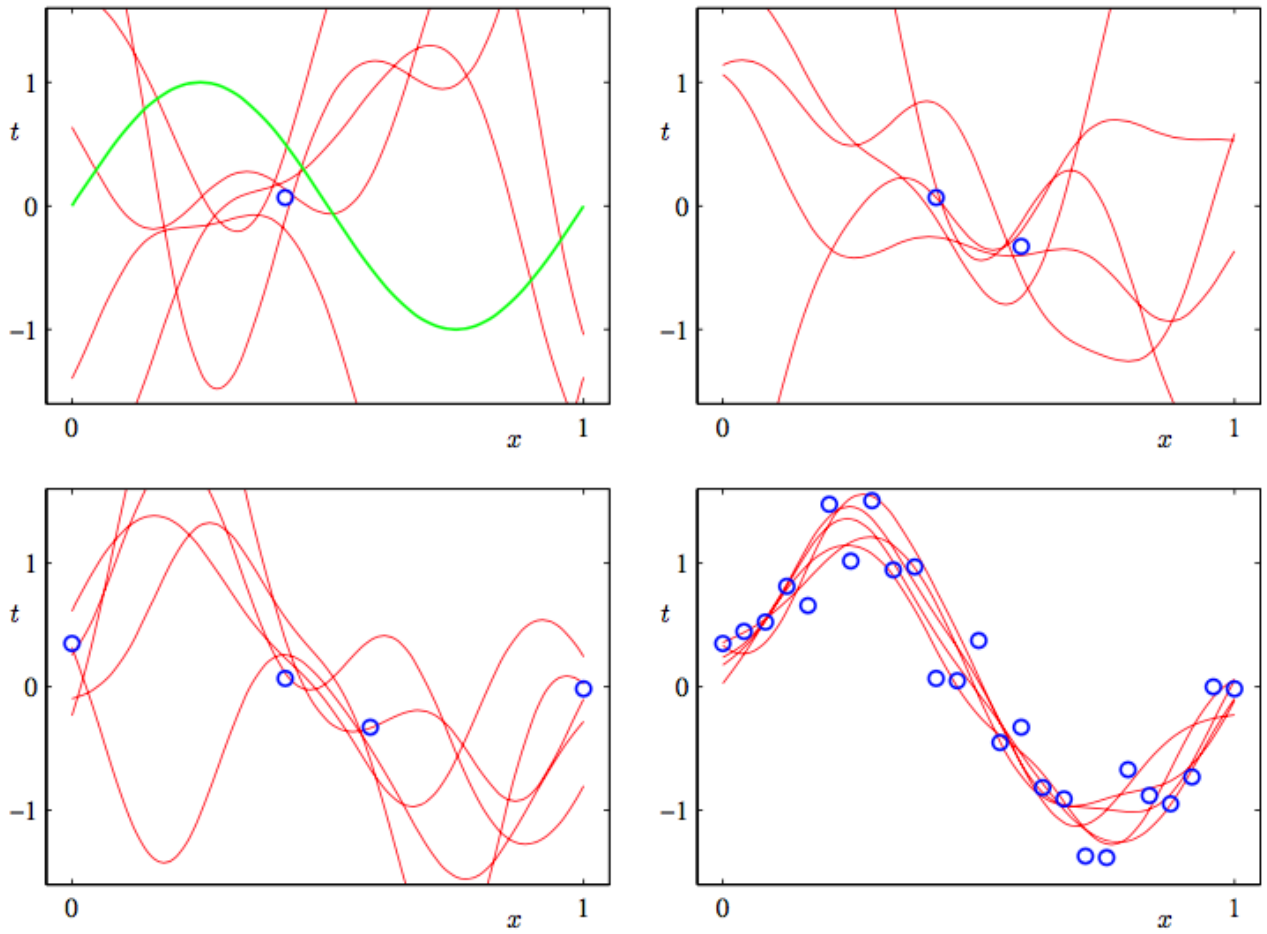
$$\phi_j(x) = \exp\left(-\frac{(x-\mu_j)^2}{2s^2}\right)$$



— Bishop, Pattern Recognition and Machine Learning

Bayesian Linear Regression

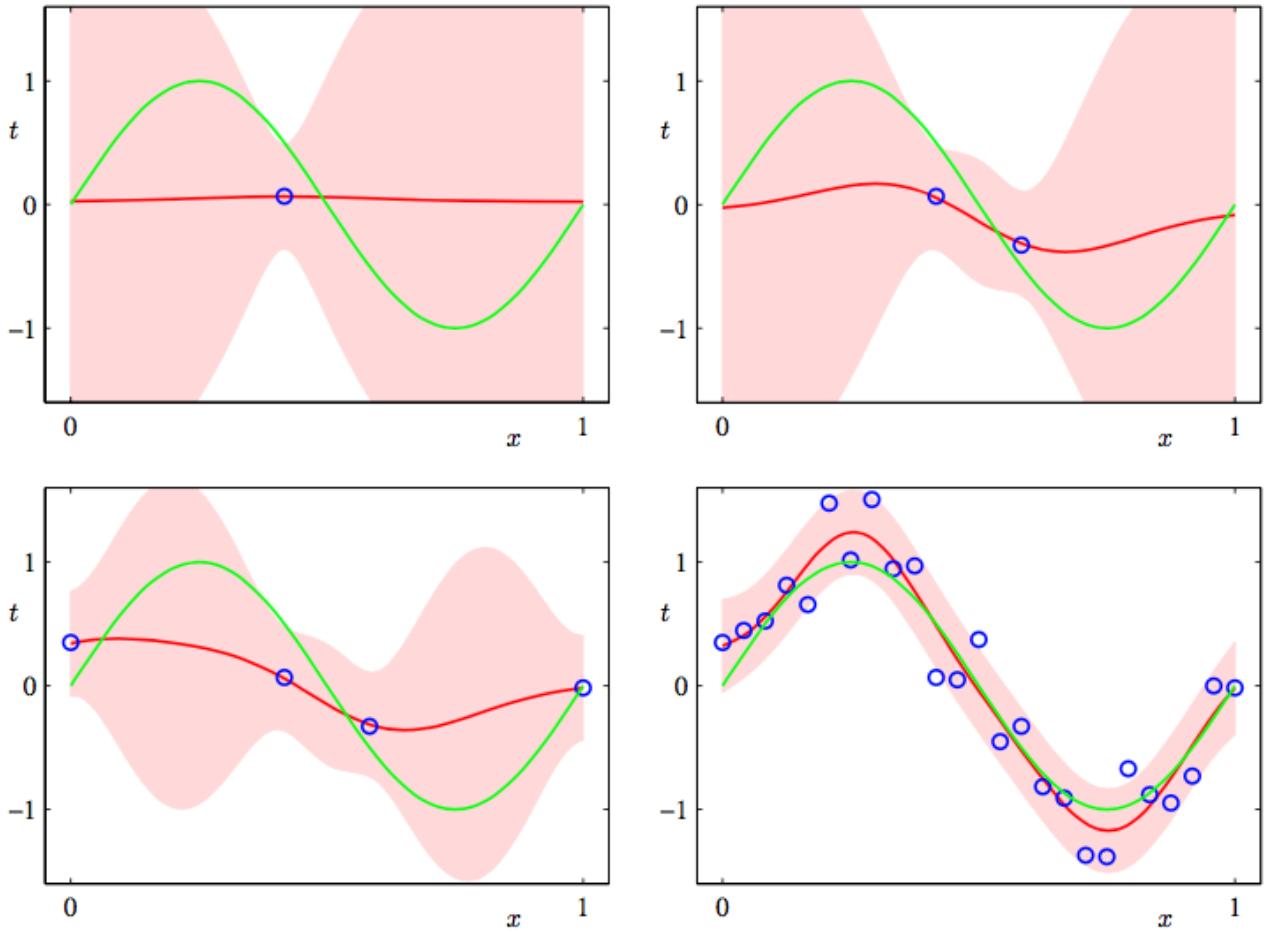
Functions sampled from the posterior:



— Bishop, Pattern Recognition and Machine Learning

Bayesian Linear Regression

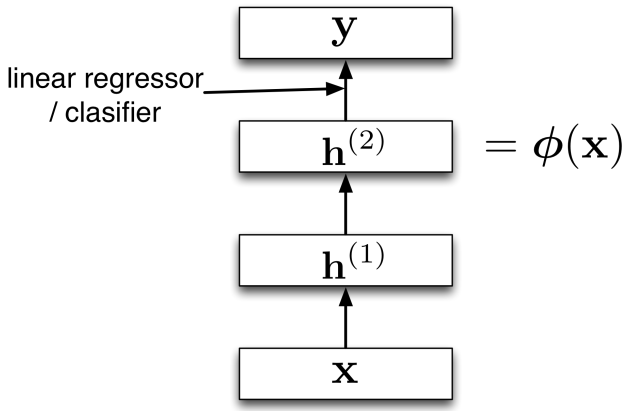
Posterior predictive distribution:



— Bishop, Pattern Recognition and Machine Learning

Bayesian Neural Networks

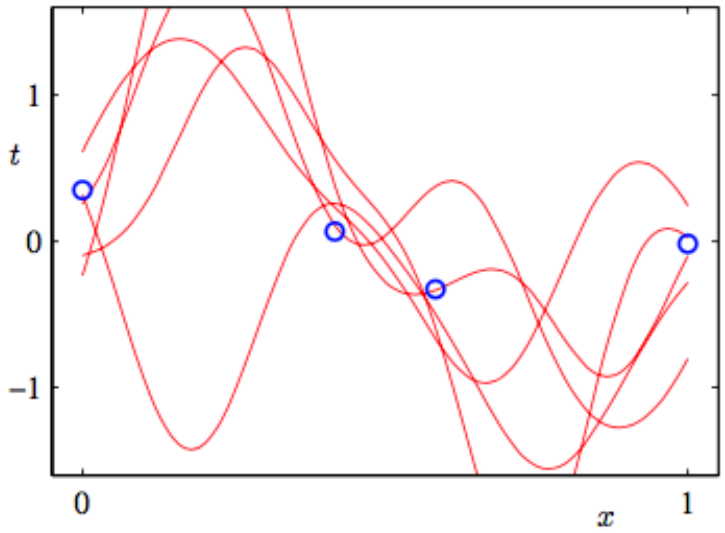
- Basis functions (i.e. feature maps) are great in one dimension, but don't scale to high-dimensional spaces.
- Recall that the second-to-last layer of an MLP can be thought of as a feature map:



- It is possible to train a Bayesian neural network, where we define a prior over all the weights for all layers, and make predictions using Bayesian parameter estimation.
  - The algorithms are complicated, and beyond the scope of this class.
  - A simple approximation which sometimes works: first train the MLP the usual way, and then do Bayesian linear regression with the learned features.

Bayesian Optimization

- Now let's apply all of this to black-box optimization. The technique we'll cover is called Bayesian optimization.
- The actual function we're trying to optimize (e.g. validation error as a function of hyperparameters) is really complicated. Let's approximate it with a simple function, called the surrogate function.
- After we've queried a certian number of points, we can condition on these to infer the posterior over the surrogate function using Bayesian linear regression.



Bayesian Optimization

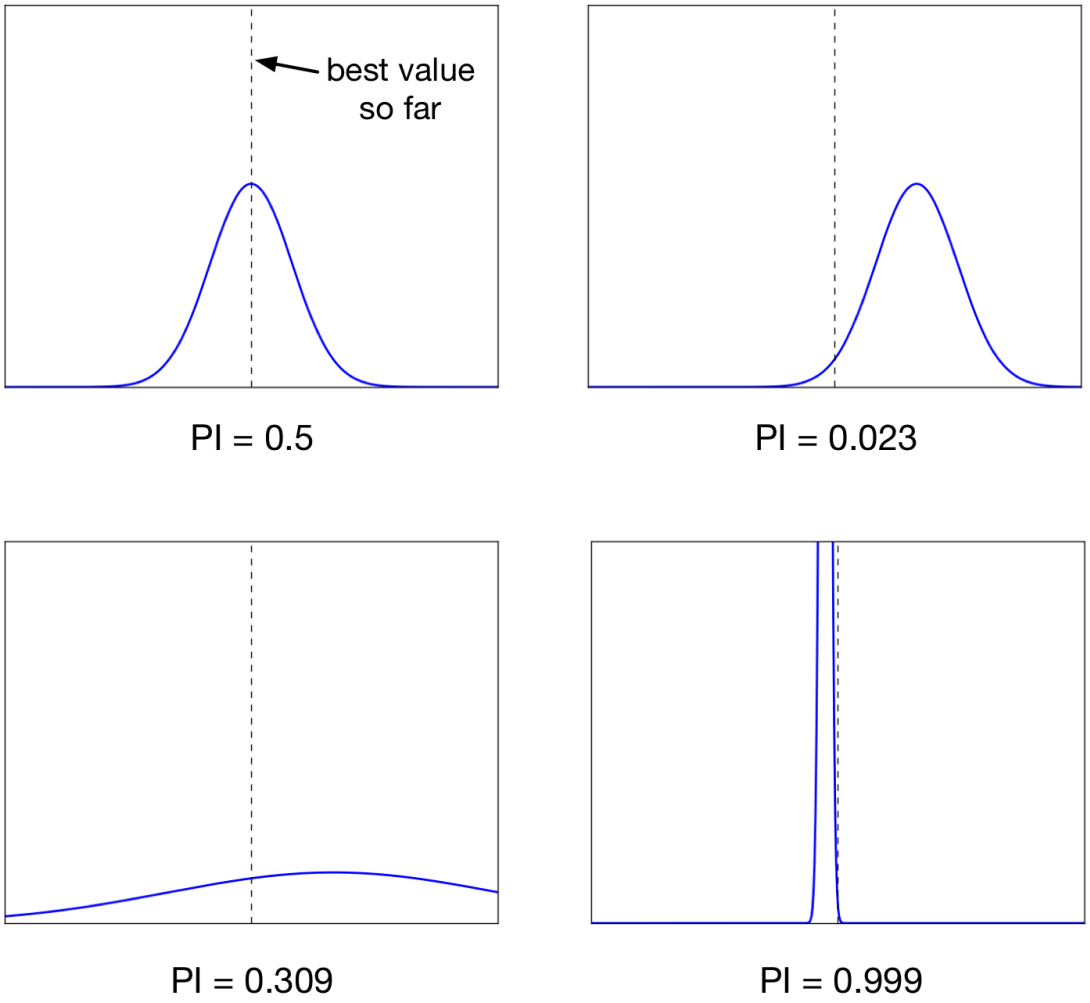
- To choose the next point to query, we must define an acquisition function, which tells us how promising a candidate it is.
- What's wrong with the following acquisition functions:
  - posterior mean: $-\mathbb{E}[f(\boldsymbol{\theta})]$
  - posterior variance: $\mathrm{Var}(f(\boldsymbol{\theta}))$
- Desiderata:
  - high for points we expect to be good
  - high for points we're uncertain about
  - low for points we've already tried
- Candidate 1: probability of improvement (PI)

$$\mathrm{PI} = \Pr(f(\theta) < \gamma - \epsilon),$$

where γ is the best value so far, and ϵ is small.

Bayesian Optimization

Examples:



- Plots show the posterior predictive distribution for $f(\theta)$.

Bayesian Optimization

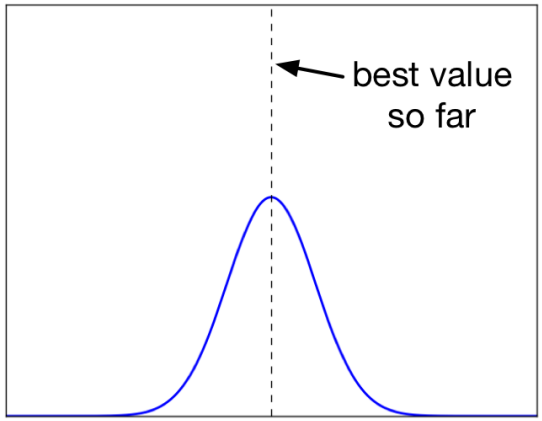
- The problem with Probability of Improvement (PI): it queries points it is highly confident will have a small imporvement
  - Usually these are right next to ones we've already evaluated
- A better choice: Expected Improvement (EI)

$$\mathrm{EI} = \mathbb{E}[\max(\gamma - f(\boldsymbol{\theta}), 0)]$$

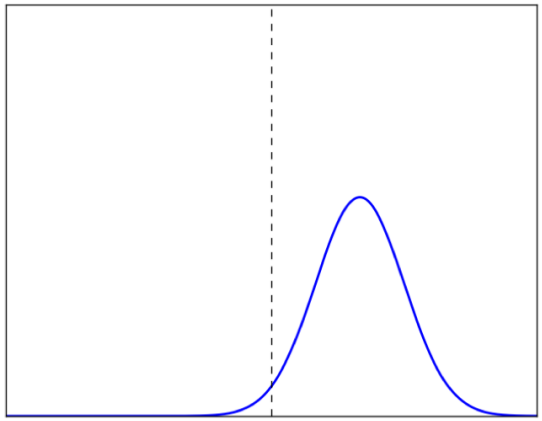
  - The idea: if the new value is much better, we win by a lot; if it's much worse, we haven't lost anything.
  - There is an explicit formula for this if the posterior predictive distribution is Gaussian.

Bayesian Optimization

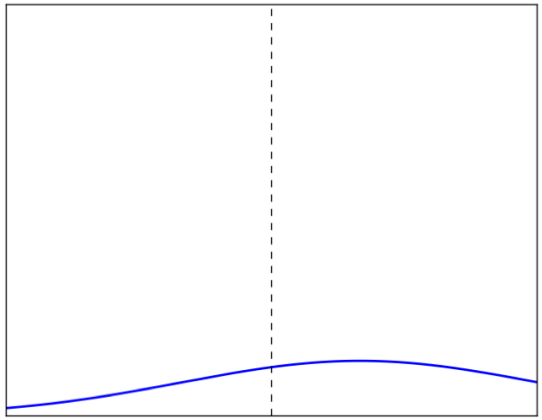
Examples:



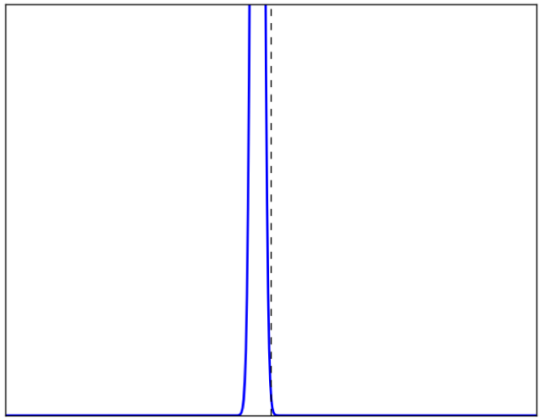
EI = 0.199



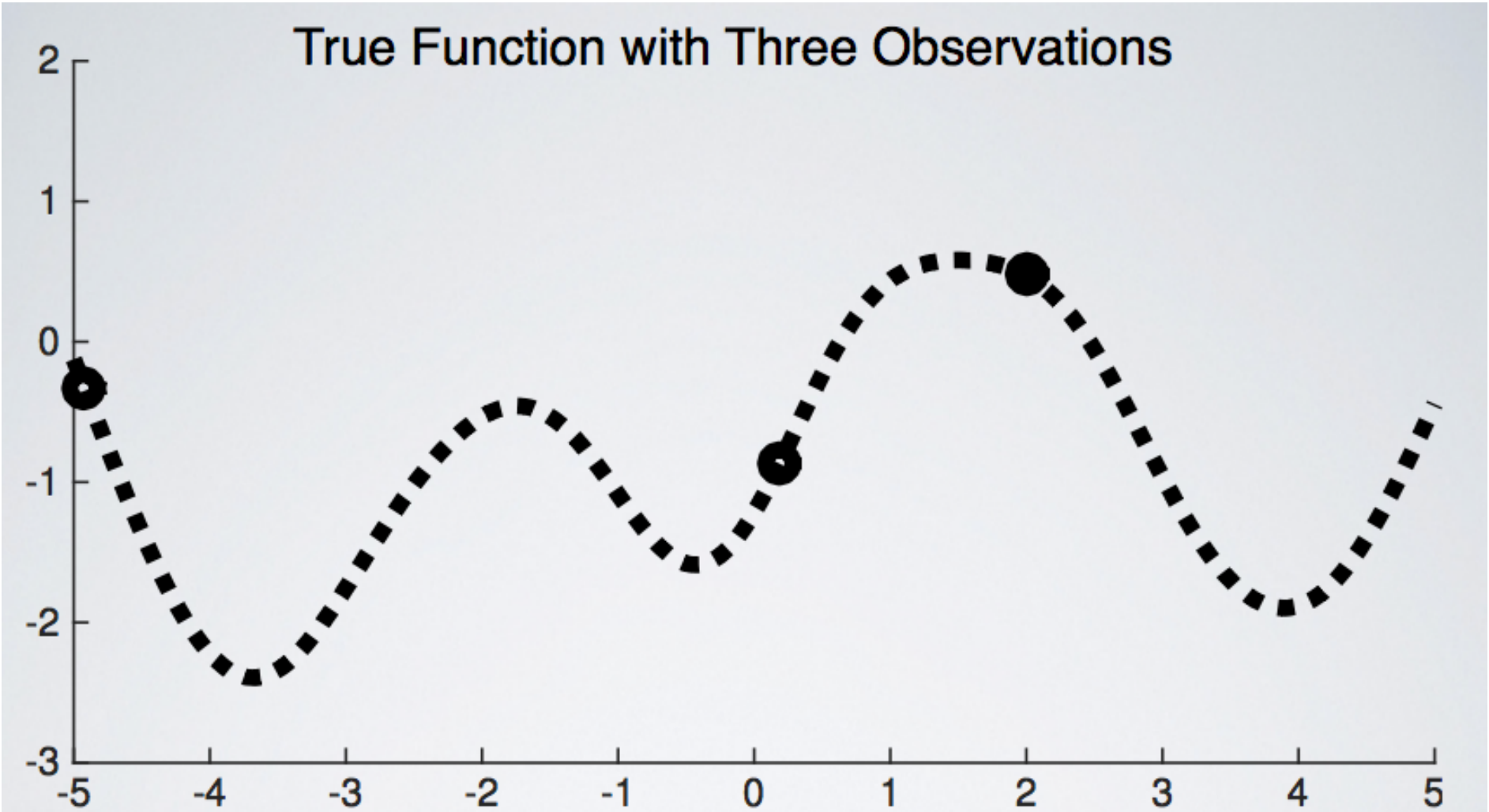
EI = 0.004

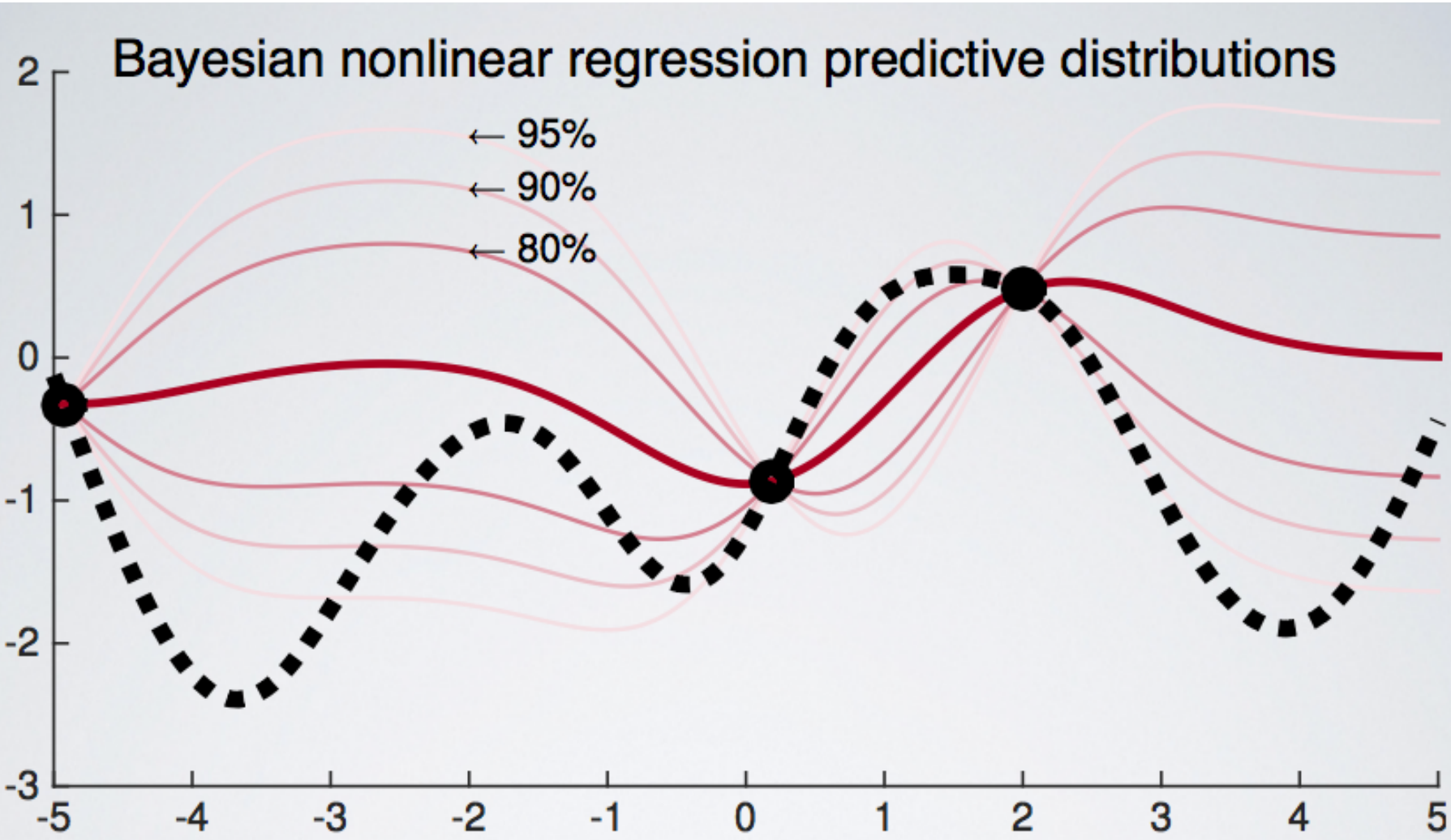


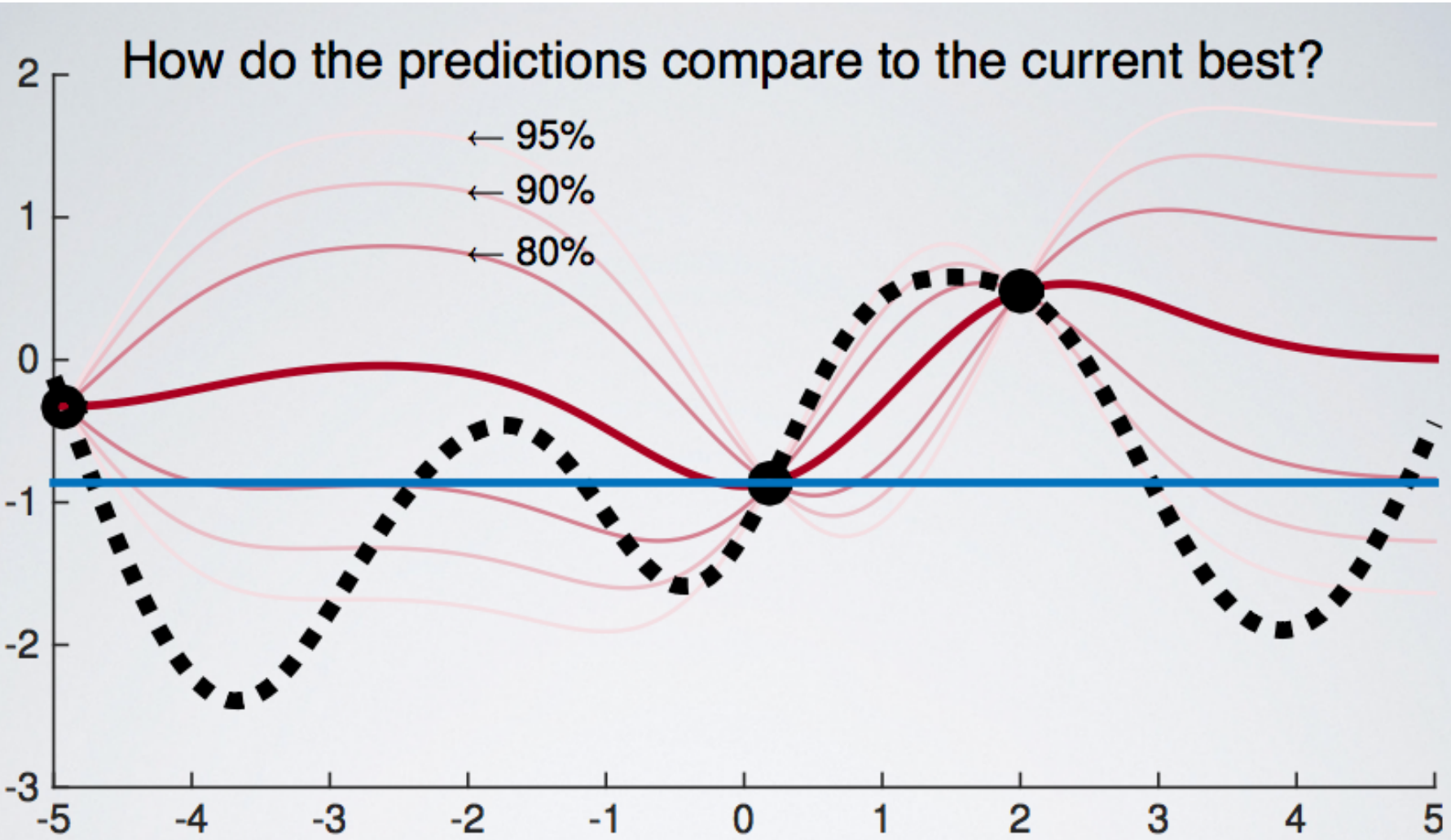
EI = 0.396

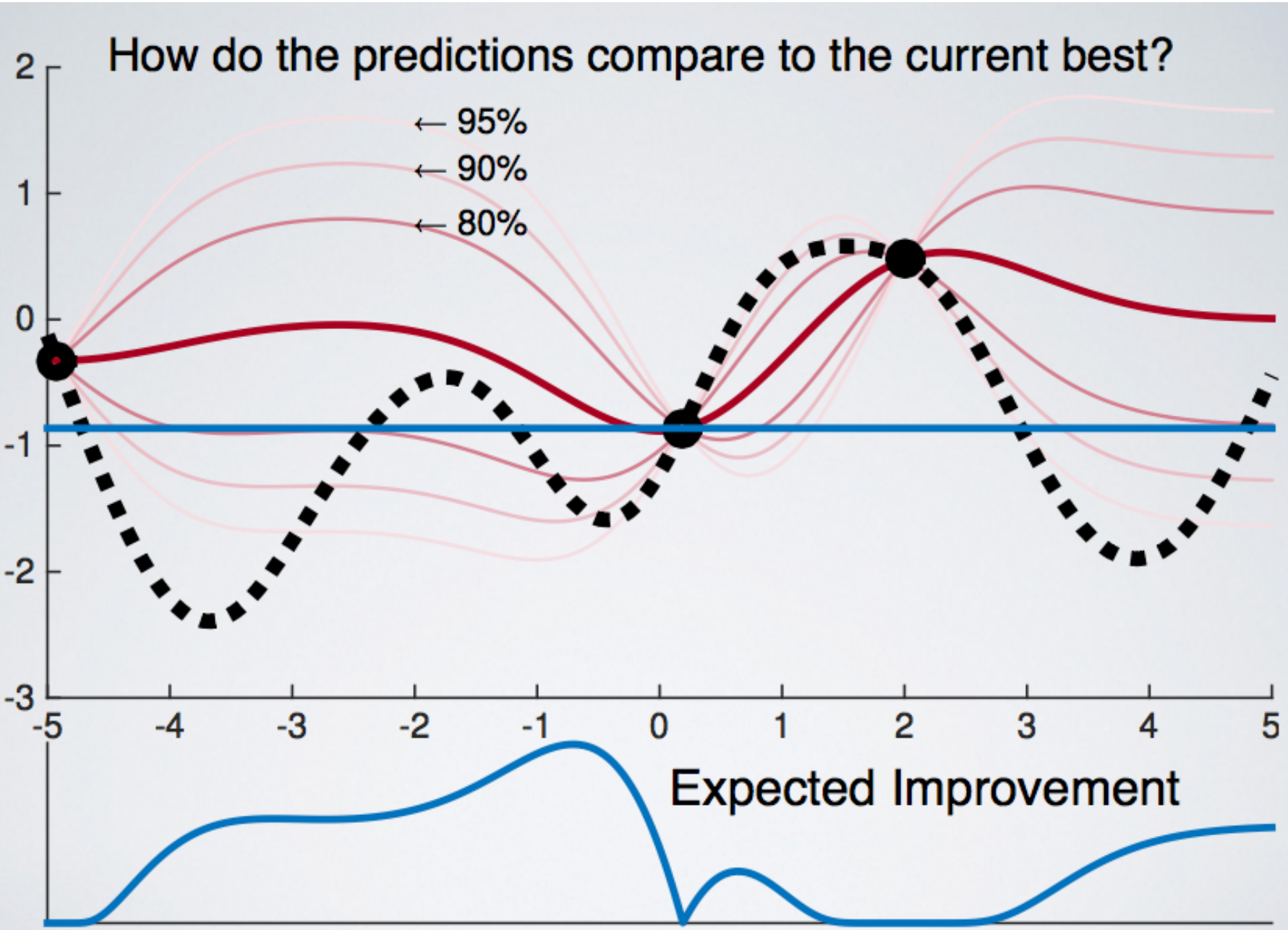


EI = 0.15


True Function with Three Observations
2
1
0
-1
-2
-3
-5
-4
-3
-2
-1
0
1
2
3
4
5


Bayesian nonlinear regression predictive distributions
← 95%
← 90%
← 80%
2
1
0
-1
-2
-3
-5
-4
-3
-2
-1
0
1
2
3
4
5


How do the predictions compare to the current best?
← 95%
← 90%
← 80%
2
1
0
-1
-2
-3
-5
-4
-3
-2
-1
0
1
2
3
4
5

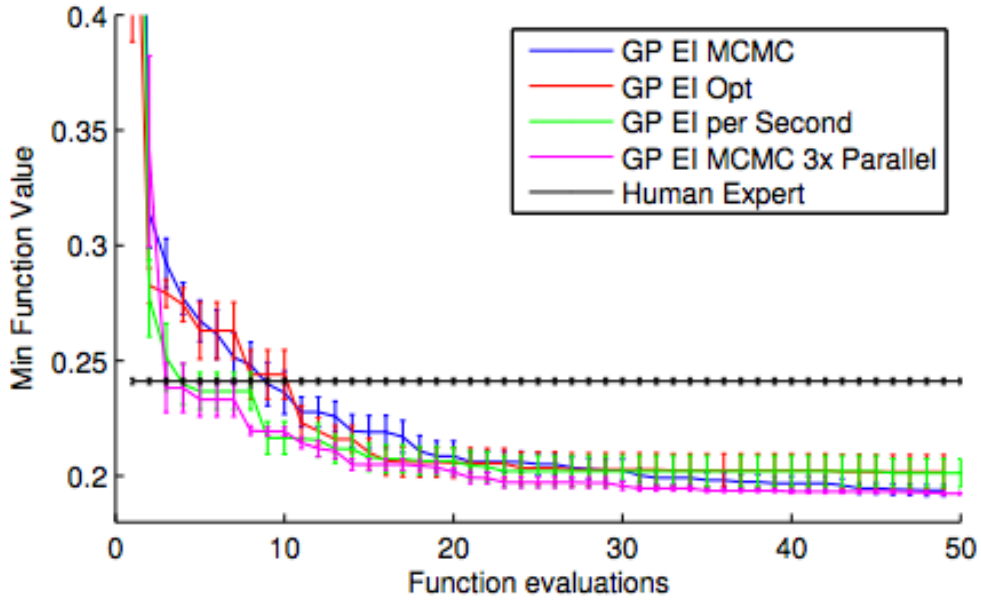

How do the predictions compare to the current best?
← 95%
← 90%
← 80%
2
1
0
-1
-2
-3
-5
-4
-3
-2
-1
0
1
2
3
4
5
Expected Improvement

Bayesian Optimization

- I showed one-dimensional visualizations, but the higher-dimensional case is conceptually no different.
  - Maximize the acquisition function using gradient descent
  - Use lots of random restarts, since it is riddled with local maxima
  - BayesOpt can be used to optimize tens of hyperparameters.
- I've described BayesOpt in terms of Bayesian linear regression with basis functions learned by a neural net.
  - In practice, it's typically done with a more advanced model called Gaussian processes, which you learn about in CSC 412.
  - But Bayesian linear regression is actually useful, since it scales better to large numbers of queries.
- One variation: some configurations can be much more expensive than others
  - Use another Bayesian regression model to estimate the computational cost, and query the point that maximizes expected improvement per second

Bayesian Optimization

- BayesOpt can often beat hand-tuned configurations in a relatively small number of steps.
- Results on optimizing hyperparameters (layer-specific learning rates, weight decay, and a few other parameters) for a CIFAR-10 conv net:



- Each function evaluation takes about an hour
- Human expert = Alex Krizhevsky, the creator of AlexNet

Bayesian Optimization

- Spearmint is an open-source BayesOpt software package that optimizes hyperparameters for you:

  `https://github.com/JasperSnoek/spearmint`

- Much of this talk was taken from the following two papers:
  - J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. NIPS, 2012.

    `http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms`
  - J. Snoek et al. Scalable Bayesian optimization using deep neural networks. ICML, 2015.

    `http://www.jmlr.org/proceedings/papers/v37/snoek15.pdf`