CSC 411 Lecture 20: Gaussian Processes

Mengye Ren and Matthew MacKay

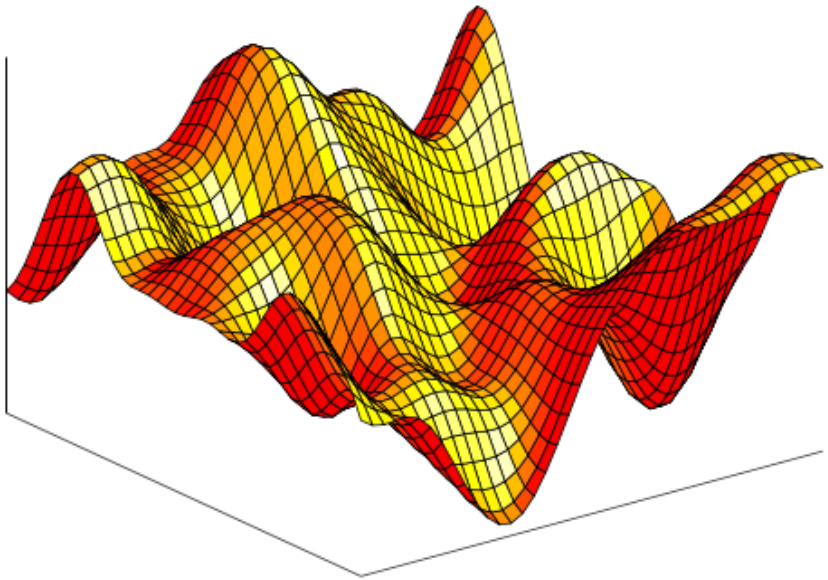
University of Toronto

Overview

- Last lecture: Bayesian linear regression, a parametric model
- This lecture: Gaussian processes, a nonparametric model
  - Define a distribution directly over functions (i.e., a **stochastic process**)
  - Derive as a generalization of Bayesian linear regression, with possibly infinite-dimensional feature mappings
  - Based on the Kernel Trick, one of the most important ideas in machine learning
  - Conceptually cleaner, since we can specify priors directly over functions. This lets us easily incorporate assumptions like smoothness, periodicity, etc., which are hard to encode as priors over regression weights.

Towards Gaussian Processes

- **Gaussian Processes** are distributions over functions.
- They're actually a simpler and more intuitive way to think about regression, once you're used to them.



— GPML

Towards Gaussian Processes

- Linear regression:
$$y(\mathbf{x}) = \mathbf{w}^\top \phi(\mathbf{x})$$
- In Bayesian linear regression, $\mathbf{w}$ is sampled from the prior $\mathcal{N}(\mathbf{m}, \mathbf{S})$. This defines a distribution over functions y
- Suppose we observe a dataset $\mathcal{D} = \{(\mathbf{x}^{(n)}, t^{(n)})\}_{n=1}^N$
- Goal: find the posterior predictive distribution of $t^{(N+1)}$ given a new datapoint $\mathbf{x}^{(N+1)}$
- Last lecture: first find the posterior over the weights, $p(\mathbf{w} \mid \mathcal{D})$, then marginalize over weights to find $p(t^{(N+1)} \mid \mathbf{x}^{(N+1)}, \mathcal{D})$
- We'll show how this can be done without directly considering the model parameters $\mathbf{w}$, which will lead us to GPs

Towards Gaussian Processes

- Let $\mathbf{y} = (y^{(1)}, \ldots, y^{(N+1)})^\top$ denote the vector of function values at $(\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N+1)})^\top$.
- Recall:

$$y^{(n)} = \mathbf{w}^\top \boldsymbol{\phi}(\mathbf{x}^{(n)}), \quad \mathbf{w} \sim \mathcal{N}(\mathbf{m}, \mathbf{S})$$

- By the linear transformation rules for Gaussian random variables, the distribution of $\mathbf{y}$ is a Gaussian with

$$\mathbb{E}[y^{(n)}] = \mathbf{m}^\top \boldsymbol{\phi}(\mathbf{x}^{(n)})$$
$$\text{Cov}(y^{(n)}, y^{(m)}) = \boldsymbol{\phi}(\mathbf{x}^{(n)})^\top \mathbf{S} \boldsymbol{\phi}(\mathbf{x}^{(m)})$$

- In vectorized form, $\mathbf{y} = \boldsymbol{\Phi}\mathbf{w}$. $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{y}, \boldsymbol{\Sigma}_\mathbf{y})$ with

$$\boldsymbol{\mu}_\mathbf{y} = \mathbb{E}[\mathbf{y}] = \boldsymbol{\Phi}\mathbf{m}$$
$$\boldsymbol{\Sigma}_\mathbf{y} = \text{Cov}(\mathbf{y}) = \boldsymbol{\Phi}\mathbf{S}\boldsymbol{\Phi}^\top$$

Towards Gaussian Processes

$$\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{y}}, \boldsymbol{\Sigma}_{\mathbf{y}})$$
$$\boldsymbol{\mu}_{\mathbf{y}} = \mathbb{E}[\mathbf{y}] = \boldsymbol{\Phi}\mathbf{m}$$
$$\boldsymbol{\Sigma}_{\mathbf{y}} = \text{Cov}(\mathbf{y}) = \boldsymbol{\Phi}\mathbf{S}\boldsymbol{\Phi}^{\top}$$

- What is this saying? This is our prior on function values $y^{(n)} = y(\mathbf{x}^{(n)})$ before we observe any targets $t^{(n)}$. It comes from our prior on $\mathbf{w}$
- Notice this distribution depends on the inputs $\mathbf{x}^{(n)}$- we need to know where we're evaluating the function!

Towards Gaussian Processes

- Recall that in Bayesian linear regression, we assume noisy Gaussian observations of the underlying function.

$$t^{(n)} \sim \mathcal{N}(y^{(n)}, \sigma^2)$$

- In vectorized form, if $\mathbf{t} = (t^{(1)}, \ldots, t^{(N+1)})^\top$ then

$$\mathbf{t} \sim \mathcal{N}(\mathbf{y}, \sigma^2 \mathbf{I})$$

- Since $\mathbf{y}$ is jointly Gaussian, so are the observations $\mathbf{t}$:

$$\mathbb{E}[t^{(n)}] = \mathbb{E}[y^{(n)}]$$

$$\mathrm{Cov}(t^{(n)}, t^{(m)}) = \begin{cases} \mathrm{Var}(y^{(n)}) + \sigma^2 & \text{if } n = m \\ \mathrm{Cov}(y^{(n)}, y^{(m)}) & \text{if } n \neq m \end{cases}$$

- In vectorized form, $\mathbf{t} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{t}, \boldsymbol{\Sigma}_\mathbf{t})$, with

$$\boldsymbol{\mu}_\mathbf{t} = \boldsymbol{\mu}_\mathbf{y}$$

$$\boldsymbol{\Sigma}_\mathbf{t} = \boldsymbol{\Sigma}_\mathbf{y} + \sigma^2 \mathbf{I}$$

Towards Gaussian Processes

In vectorized form, $\mathbf{t} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{t}, \boldsymbol{\Sigma}_\mathbf{t})$, with

$$\boldsymbol{\mu}_\mathbf{t} = \boldsymbol{\mu}_\mathbf{y}$$
$$\boldsymbol{\Sigma}_\mathbf{t} = \boldsymbol{\Sigma}_\mathbf{y} + \sigma^2 \mathbf{I}$$

- What is this saying? This is our prior on observed targets $t^{(n)}$ before we observe them. It comes from our prior on the function values $\mathbf{y}$

Towards Gaussian Processes

- Let $\mathbf{t}_N = (t^{(1)}, \ldots, t^{(N)})^\top$. We just saw $\mathbf{t} = \begin{pmatrix} \mathbf{t}_N \\ t^{(N+1)} \end{pmatrix}$ is jointly Gaussian with distribution $\mathcal{N}(\boldsymbol{\mu}_\mathbf{t}, \boldsymbol{\Sigma}_\mathbf{t})$.
- We can divide up these parameters into blocks corresponding to the first N elements and the $(N+1)^{\text{th}}$ element:

$$\boldsymbol{\mu}_\mathbf{t} = \begin{pmatrix} \boldsymbol{\mu}_N \\ \mu_{N+1} \end{pmatrix}, \quad \boldsymbol{\Sigma}_\mathbf{t} = \begin{pmatrix} \boldsymbol{\Sigma}_N & \mathbf{s} \\ \mathbf{s}^\top & \sigma_{N+1} \end{pmatrix}$$

- Remember our goal: Find the posterior predictive distribution of $t^{(N+1)}$, given that we've observed $\mathbf{t}_N$
- The predictive distribution is a special case of the conditioning formula for a multivariate Gaussian:

$$t^{(N+1)} \mid \mathbf{t}_N \sim \mathcal{N}(\mu_{\text{pred}}, \sigma_{\text{pred}})$$
$$\mu_{\text{pred}} = \mu_{N+1} + \mathbf{s}^\top \boldsymbol{\Sigma}_N^{-1} (\mathbf{t}_N - \boldsymbol{\mu}_N)$$
$$\sigma_{\text{pred}} = \sigma_{N+1} - \mathbf{s}^\top \boldsymbol{\Sigma}_N^{-1} \mathbf{s}$$

Towards Gaussian Processes

- The marginal likelihood is just $p(\mathcal{D}) = p(\mathbf{t}_N)$. We can derive this from the joint $p(\mathbf{t}_N, t^{(N+1)}) = p(\mathbf{t}) = \mathcal{N}(\boldsymbol{\mu}_\mathbf{t}, \boldsymbol{\Sigma}_\mathbf{t})$.
- We have:

$$\boldsymbol{\mu}_\mathbf{t} = \begin{pmatrix} \boldsymbol{\mu}_N \\ \mu_{N+1} \end{pmatrix}, \quad \boldsymbol{\Sigma}_\mathbf{t} = \begin{pmatrix} \boldsymbol{\Sigma}_N & \mathbf{s} \\ \mathbf{s}^\top & \sigma_{N+1} \end{pmatrix}$$

- Thus, $p(\mathbf{t}_N)$ is the PDF of a multivariate Gaussian:

$$\begin{aligned} p(\mathcal{D}) &= \mathcal{N}(\mathbf{t}_N; \boldsymbol{\mu}_N, \boldsymbol{\Sigma}_N) \\ &= \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}_N|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{t}_N - \boldsymbol{\mu}_N)^\top \boldsymbol{\Sigma}_N^{-1}(\mathbf{t}_N - \boldsymbol{\mu}_N)\right) \end{aligned}$$

Towards Gaussian Processes

- To summarize:
  1. We started with a prior on the weights $\mathbf{w}$: $\mathcal{N}(\mathbf{m}, \mathbf{S})$
  2. This gave us a distribution on the function values $\mathbf{y}$: $\mathcal{N}(\boldsymbol{\mu}_\mathbf{y}, \boldsymbol{\Sigma}_\mathbf{y})$ with

$$\boldsymbol{\mu}_\mathbf{y} = \boldsymbol{\Phi}\mathbf{m}, \quad \boldsymbol{\Sigma}_\mathbf{y} = \boldsymbol{\Phi}\mathbf{S}\boldsymbol{\Phi}^\top$$

  3. Targets are noisy, leading to a distribution on targets $\mathbf{t}$: $\mathcal{N}(\boldsymbol{\mu}_\mathbf{t}, \boldsymbol{\Sigma}_\mathbf{t})$

$$\boldsymbol{\mu}_\mathbf{t} = \boldsymbol{\mu}_\mathbf{y}, \quad \boldsymbol{\Sigma}_\mathbf{t} = \boldsymbol{\Sigma}_\mathbf{y} + \sigma^2\mathbf{I}$$

  4. We conditioned on the observed targets $\mathbf{t}_N$ to find the posterior predictive distribution for the new target $t^{(N+1)}$, using the Gaussian conditioning formula

- $\boldsymbol{\mu}_\mathbf{y}$ and $\boldsymbol{\Sigma}_\mathbf{y}$ are functions of the prior's parameters $\mathbf{m}, \mathbf{S}$ and inputs $\mathbf{x}^{(n)}$
- But after $\boldsymbol{\mu}_\mathbf{y}$ and $\boldsymbol{\Sigma}_\mathbf{y}$ are specified, we can forget about $\mathbf{w}$
- What if we ignored weights from the start and let $\boldsymbol{\mu}_\mathbf{y}$ and $\boldsymbol{\Sigma}_\mathbf{y}$ be arbitrary functions of the inputs $\mathbf{x}^{(n)}$?

Gaussian Processes

$$\mathbf{y} = (y^{(1)}, \ldots, y^{(N)}) \text{ where } y^{(n)} = y(\mathbf{x}^{(n)})$$

- We need to specify
  - a mean function μ: $\mathbb{E}[y^{(n)}] = \mu(\mathbf{x}^{(n)})$
  - a covariance function k called a **kernel function**: $k(\mathbf{x}^{(n)}, \mathbf{x}^{(m)}) = \text{Cov}(y^{(n)}, y^{(n)})$
- We use these functions to specify a Gaussian prior over the function values $\mathbf{y}$:
  - Let $\mathbf{K_X}$ denote the kernel matrix. This is a matrix whose (i, j) entry is $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$, and is called the **Gram matrix**.
  - Let $\boldsymbol{\mu}_\mathbf{X}$ denote the mean vector, with the i^{th} entry given by $\mu(\mathbf{x}^{(i)})$

  Then the prior on function values $\mathbf{y}$ will be given by:

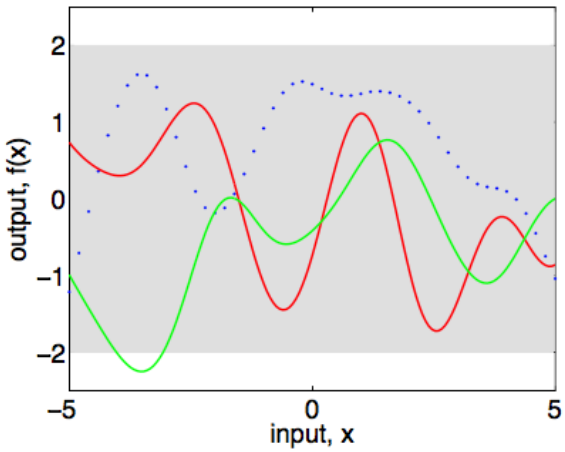
$$\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}_\mathbf{X}, \mathbf{K_X})$$

Gaussian Processes

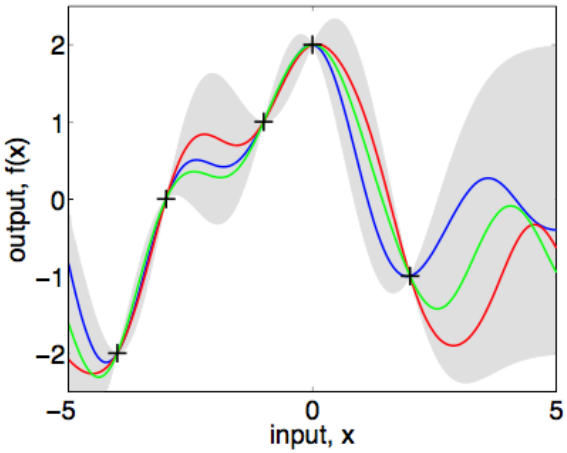
- Are there any restrictions necessary on the functions μ, k for this to be a valid distribution?
- If $k(x, x) < 0$ for some x, this means we're saying $\text{Var}(y(x)) < 0$, which can't happen
- It turns out we need that $\mathbf{K_X}$ be positive semidefinite for *any* $\mathbf{X}$. Other than that, k can be arbitrary.
- We can use any function μ that we want, but usually we choose it to be the constant zero function

Gaussian Processes

- We've just defined a distribution over *function values* at an arbitrary finite set of points.
- This can be extended to a distribution over *functions* using a kind of black magic called the Kolmogorov Extension Theorem. This distribution over functions is called a **Gaussian process (GP)**.
- We only ever need to compute with distributions over function values. The formulas from a few slides ago are all you need to do regression with GPs.
- But distributions over functions are conceptually cleaner.



(a), prior



(b), posterior

- How do you think these plots were generated?

Kernel Trick

- This is an instance of a more general trick called the **Kernel Trick**.
- Many algorithms (e.g. linear regression, logistic regression, SVMs) can be written in terms of dot products between feature vectors, $\phi(\mathbf{x})^\top \phi(\mathbf{x}')$.
- **Mercer's Theorem** (informal): For a given kernel function k, if $\mathbf{K}_\mathbf{X}$ is positive semidefinite for *any* $\mathbf{X}$ then $k(\mathbf{x}, \mathbf{x}')$ is given by an inner product in some (possibly infinite-dimensional) feature space
- Whenever an algorithm is given in terms of $\phi(\mathbf{x})^\top \phi(\mathbf{x}')$, we can replace it with $k(\mathbf{x}, \mathbf{x}')$ for a valid kernel k! This is called **kernelizing** the algorithm

Kernel Trick

- A **kernel** implements an inner product between feature vectors often much more efficiently than the explicit dot product.
- For instance, the following feature vector is quadratic in size:

$$\phi(\mathbf{x}) = (1, \sqrt{2}x_1, ..., \sqrt{2}x_d, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, ...\sqrt{2}x_{d-1}x_d, x_1^2, ..., x_d^2)$$

- But the **quadratic kernel** can compute the inner product in linear time:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}') = 1 + \sum_{i=1}^{d} 2x_i x_i' + \sum_{i,j=1}^{d} x_i x_j x_i' x_j' = (1 + \mathbf{x}^\top \mathbf{x}')^2$$

Kernel Trick

- We rarely think about the underlying feature space explicitly. Instead, we build kernels directly.
- Useful composition rules for kernels (to be proved in Homework 7):
  - A constant function $k(\mathbf{x}, \mathbf{x}') = \alpha$ is a kernel.
  - If k_1 and k_2 are kernels and $a, b \geq 0$, then $ak_1 + bk_2$ is a kernel.
  - If k_1 and k_2 are kernels, then the product $k(\mathbf{x}, \mathbf{x}') = k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}')$ is a kernel. (Interesting and surprising fact!)
- Before neural nets took over, kernel SVMs were probably the best-performing general-purpose classification algorithm.

Kernel Trick: Computational Cost

- The kernel trick lets us implicitly use very high-dimensional (even infinite-dimensional) feature spaces, but this comes at a cost.
- **Bayesian linear regression:**
  - Computational cost comes from deriving posterior in weight space: $\mathbf{w}|\mathcal{D} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

$$\boldsymbol{\mu} = \sigma^{-2}\boldsymbol{\Sigma}\boldsymbol{\Phi}^\top \mathbf{t}$$
$$\boldsymbol{\Sigma}^{-1} = \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \mathbf{S}^{-1}$$

  - Need to compute the inverse of a $D \times D$ matrix, which is an $\mathcal{O}(D^3)$ operation. (D is the number of features.)
- **GP regression:**

$$\mu_{\text{pred}} = \mu_{N+1} + \mathbf{s}^\top \boldsymbol{\Sigma}_N^{-1}(\mathbf{t}_N - \boldsymbol{\mu}_N)$$
$$\sigma_{\text{pred}} = \sigma_{N+1} - \mathbf{s}^\top \boldsymbol{\Sigma}_N^{-1}\mathbf{s}$$

  - Need to invert an $N \times N$ matrix! (N is the number of training examples.)

Kernel Trick: Computational Cost

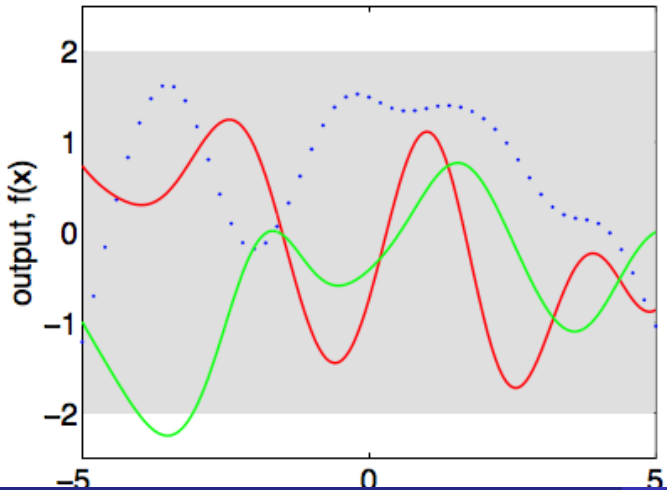
- This $\mathcal{O}(N^3)$ cost is typical of kernel methods. Most exact kernel methods don't scale to more than a few thousand data points.
- Kernel SVMs can be scaled further, since you can show you only need to consider the kernel over the support vectors, not the entire training set. (This is part of why they were so useful.)
- Scaling GP methods to large datasets is an active (and fascinating) research area.

GP Kernels

- One way to define a kernel function is to give an explicit feature map ϕ and define $k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$
- But we have lots of other options. Here's a useful one, called the **squared-exp**, or **Gaussian**, or **radial basis function (RBF)** kernel:

$$k_{\text{SE}}(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2}\right)$$

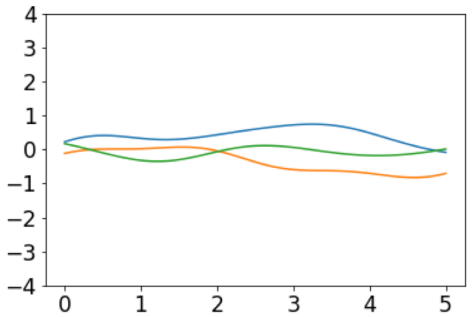
- More accurately, this is a **kernel family** with **hyperparameters** σ and ℓ.
- It gives a distribution over smooth functions:



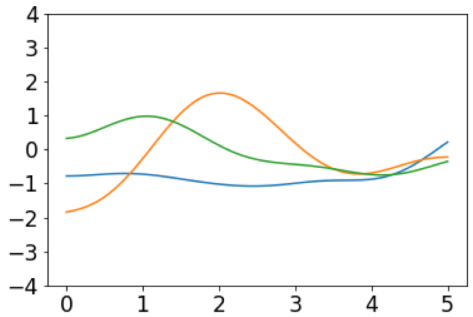
GP Kernels

$$k_{\mathrm{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right)$$

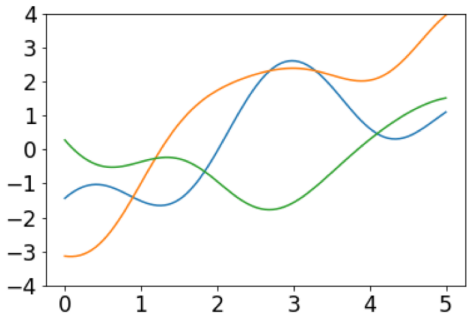
- The hyperparameters determine key properties of the function.
- Varying the **output variance** σ^2:



$\sigma^2 = 0.3$

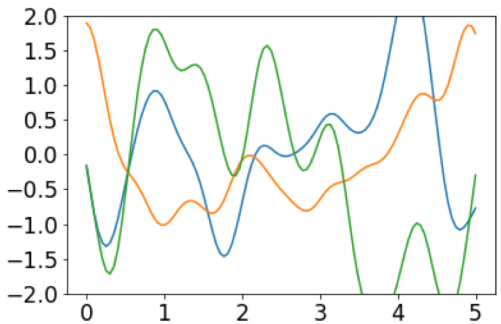


$\sigma^2 = 1$

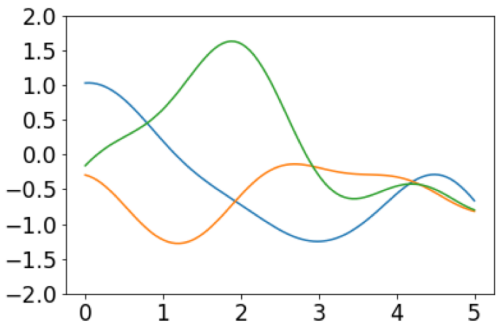


$\sigma^2 = 3$

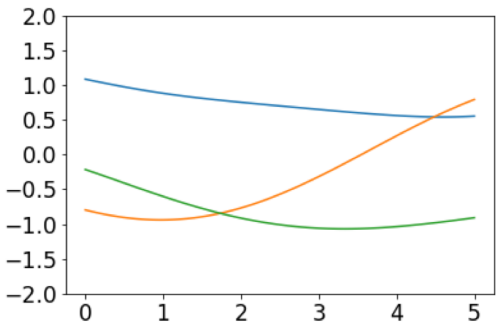
- Varying the **lengthscale** ℓ:



$\ell = 0.3$



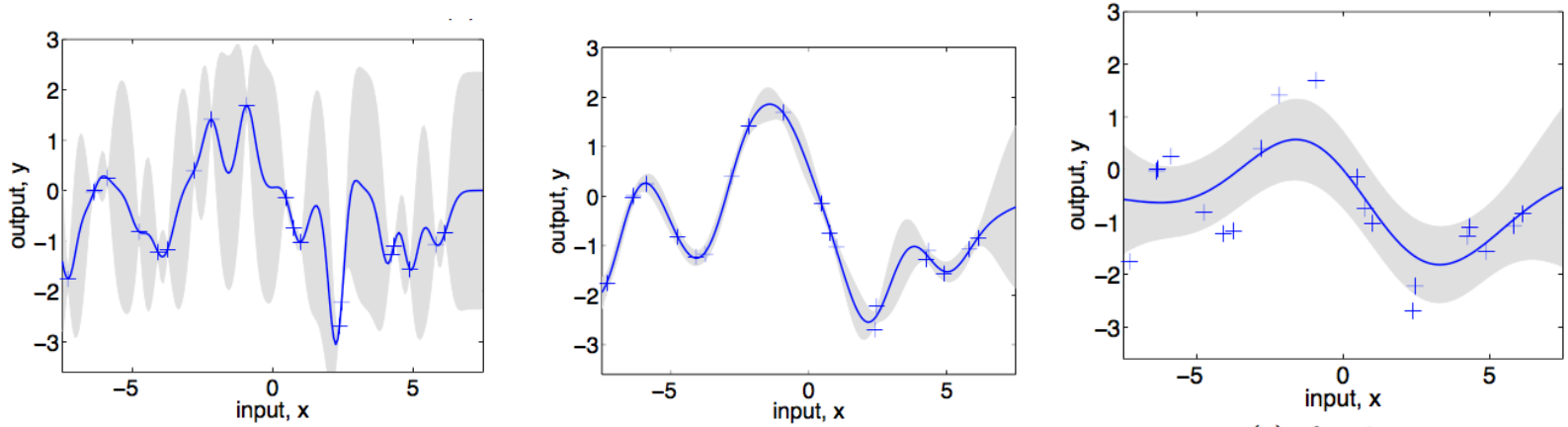
$\ell = 1$



$\ell = 3$

GP Kernels

- The choice of hyperparameters heavily influences the predictions:



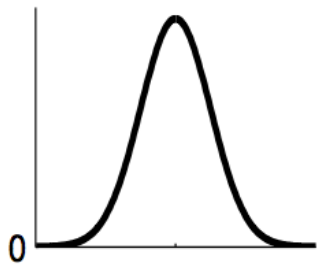
(b), $\ell = 0.3$ (a), $\ell = 1$ (c), $\ell = 3$

- In practice, it's very important to tune the hyperparameters (e.g. by maximizing the marginal likelihood).

GP Kernels

$$k_{\text{SE}}(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right)$$

- The squared-exp kernel is **stationary** because it only depends on $x - x'$. Most kernels we use in practice are stationary.
- We can visualize the function $k(0, x)$:



Now for a more interesting use of Bayesian decision theory...

Bayesian Optimization

- **Black-box optimization**: we want to minimize a function $y(\mathbf{x})$, but we only get to query function values (i.e. no gradients!)
  - Each query is expensive, so we want to do as few as possible
  - Canonical example: minimize the validation error of an ML algorithm with respect to its hyperparameters
- **Bayesian Optimization**: we're uncertain what the true function y is. Represent this uncertainty with a probability distribution over functions $p(y)$ e.g. a GP
- After we've queried points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}$ and built a dataset $\mathcal{D} = \{(\mathbf{x}^{(n)}, y^{(n)})\}_{n=1}^{N}$, we can infer the posterior $p(y|\mathcal{D})$
- Choose next point $\mathbf{x}^{(N+1)}$ to evaluate to balance:
  - Exploitation: choose a point $\mathbf{x}$ we're relatively certain to improve on best value we've seen so far
    - e.g. if b is best value so far choose $\mathbf{x}$ s.t. $p(y(\mathbf{x}) < b|\mathcal{D})$ is large
  - Exploration: choose a point we're unsure will improve on best value so far, but will reduce uncertainty in $p(y)$

Bayesian Optimization

- To choose the next point to query, define an **acquisition function** $a(\mathbf{x}; \mathcal{D})$, which tells us how promising a candidate $\mathbf{x}$ is given our current dataset $\mathcal{D}$
  - Then choose next point to maximize a
- What's wrong with the following acquisition functions:
  - posterior mean: $a(\mathbf{x}; \mathcal{D}) = -\mathbb{E}[y(\mathbf{x})|\mathcal{D}]$
  - posterior variance: $a(\mathbf{x}; \mathcal{D}) = \mathrm{Var}(y(\mathbf{x})|\mathcal{D})$
- Desiderata:
  - high for points we expect to be good (exploitation)
  - high for points we're uncertain about (exploration)
  - low for points we've already tried
- Candidate 1: **probability of improvement (PI)**

$$\mathrm{PI}(\mathbf{x}; \mathcal{D}) = \Pr(y(\mathbf{x}) < b - \epsilon | \mathcal{D}),$$

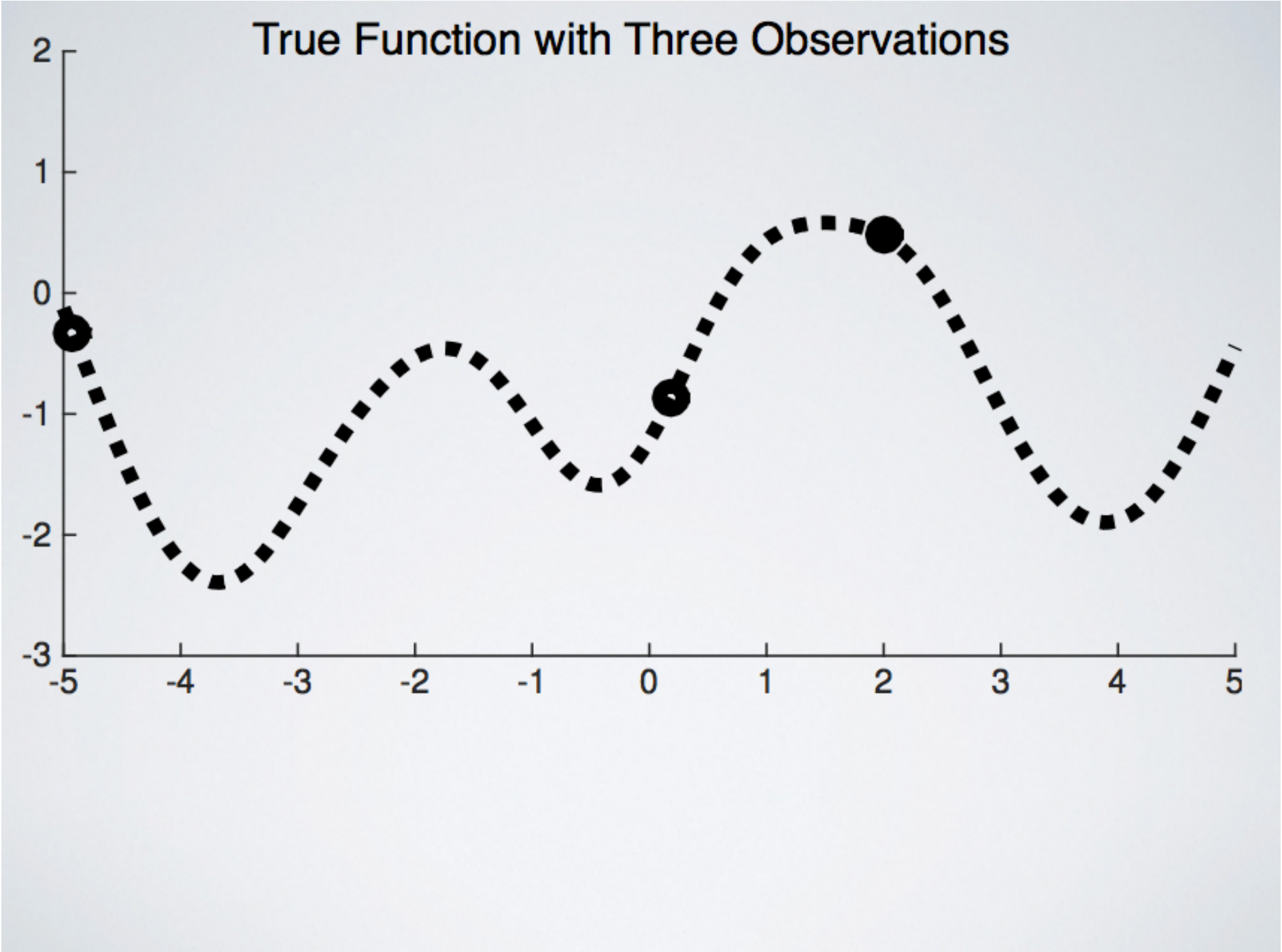
where b is the best value so far, and ϵ is small.

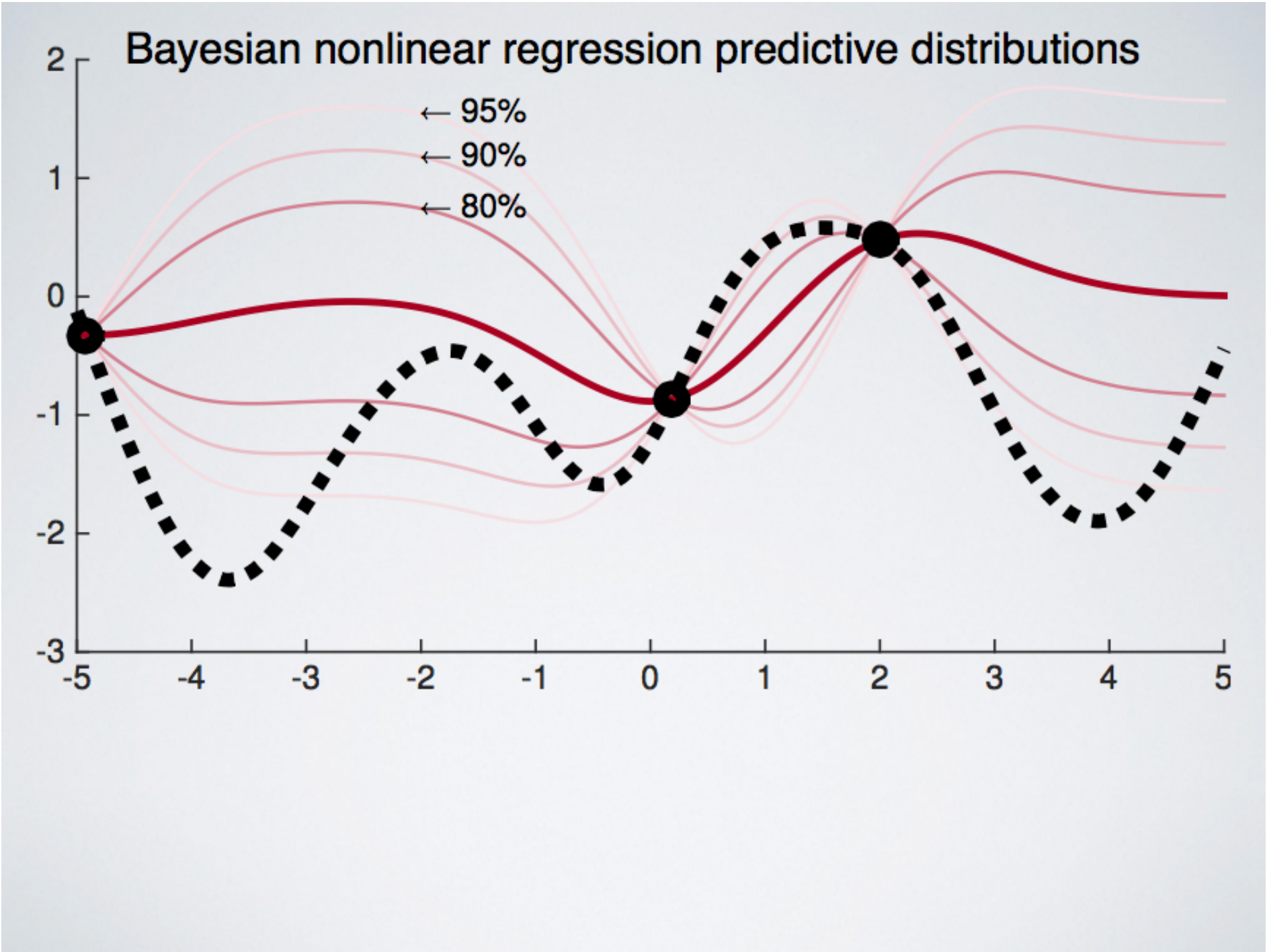
Bayesian Optimization

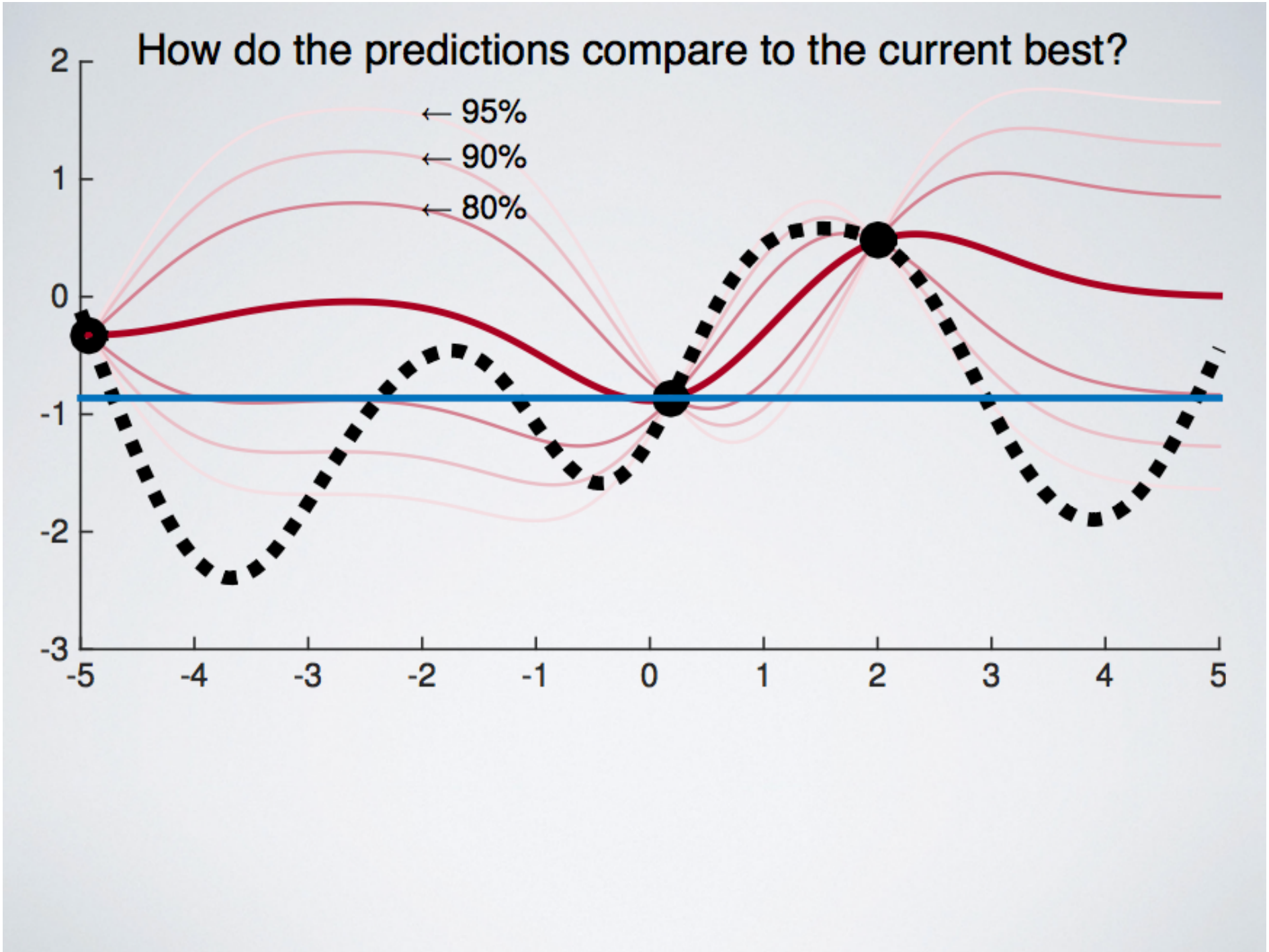
- The problem with Probability of Improvement (PI): it queries points it is highly confident will have a small imporvement
  - Usually these are right next to ones we've already evaluated
- A better choice: **Expected Improvement (EI)**

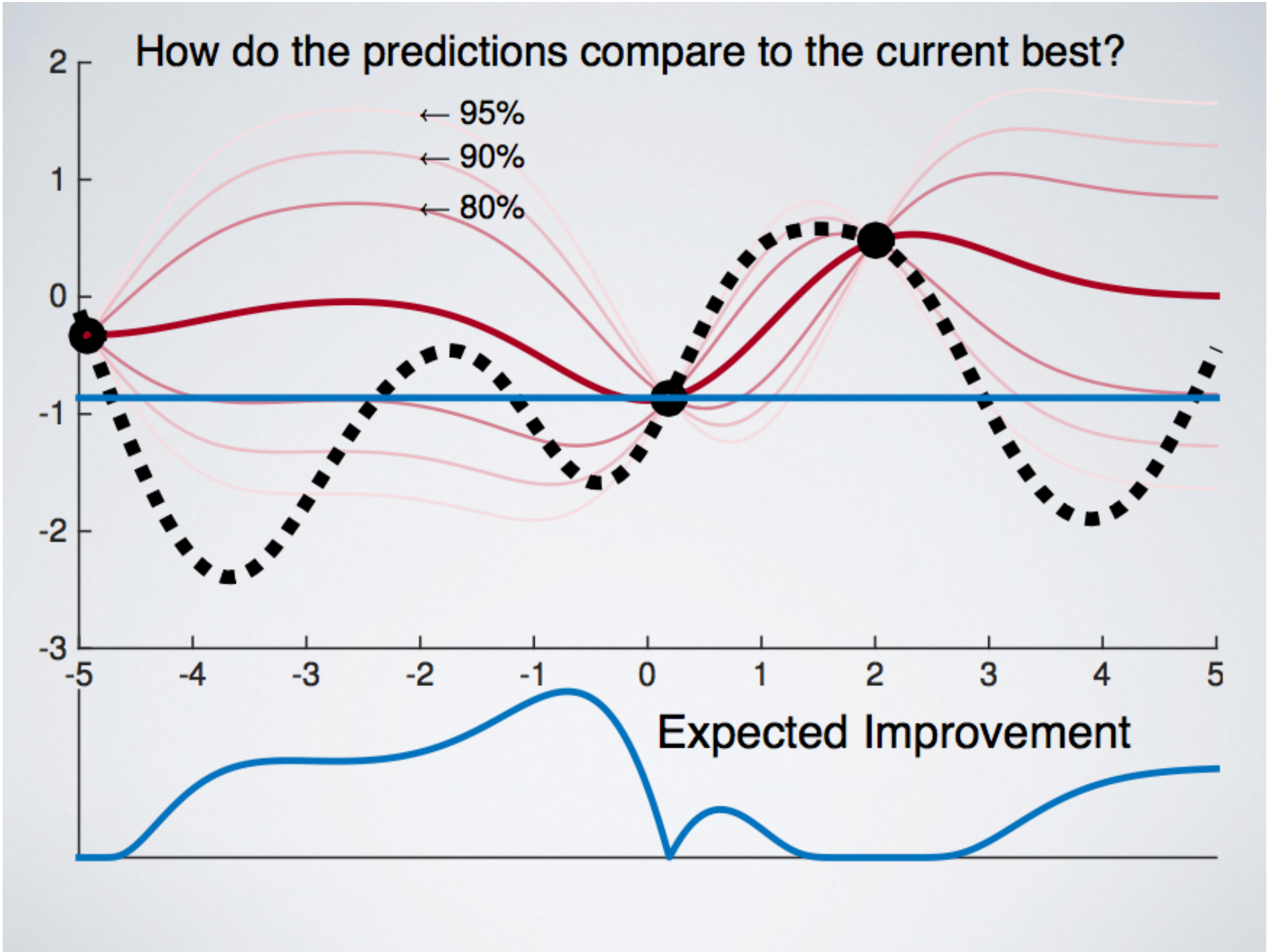
$$\mathrm{EI}(\mathbf{x}; \mathcal{D}) = \mathbb{E}[\max(b - y(\mathbf{x}), 0) | \mathcal{D}]$$

  - The idea: if the new value is much better, we win by a lot; if it's much worse, we haven't lost anything.


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

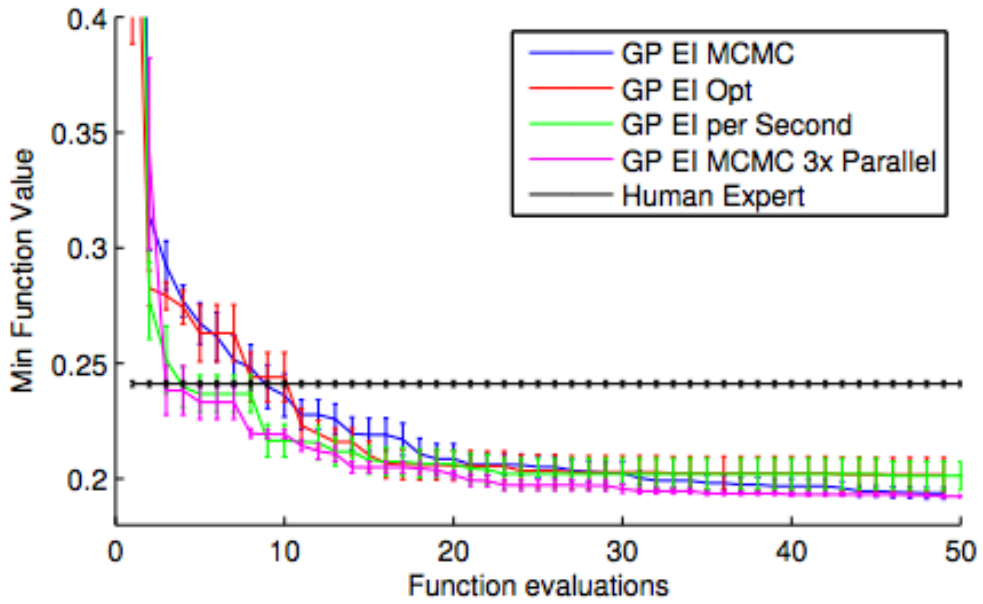

How do the predictions compare to the current best?
← 95%
← 90%
← 80%
2
1
0
-1
-2
-3
-5
-4
-3
-2
-1
0
1
2
3
4
5
Expected Improvement

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.
- In practice, it's typically done with Gaussian processes
  - 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

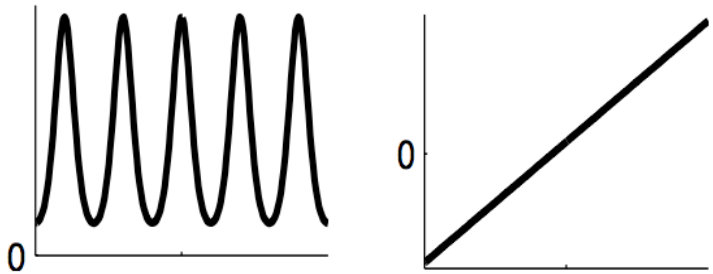
- BayesOpt can often beat hand-tuned configurations in a relatively small number of steps.
- Results on optimizing hyperparameters (layer-specific learning rates, weight decay, and a few other parameters) for a CIFAR-10 conv net:

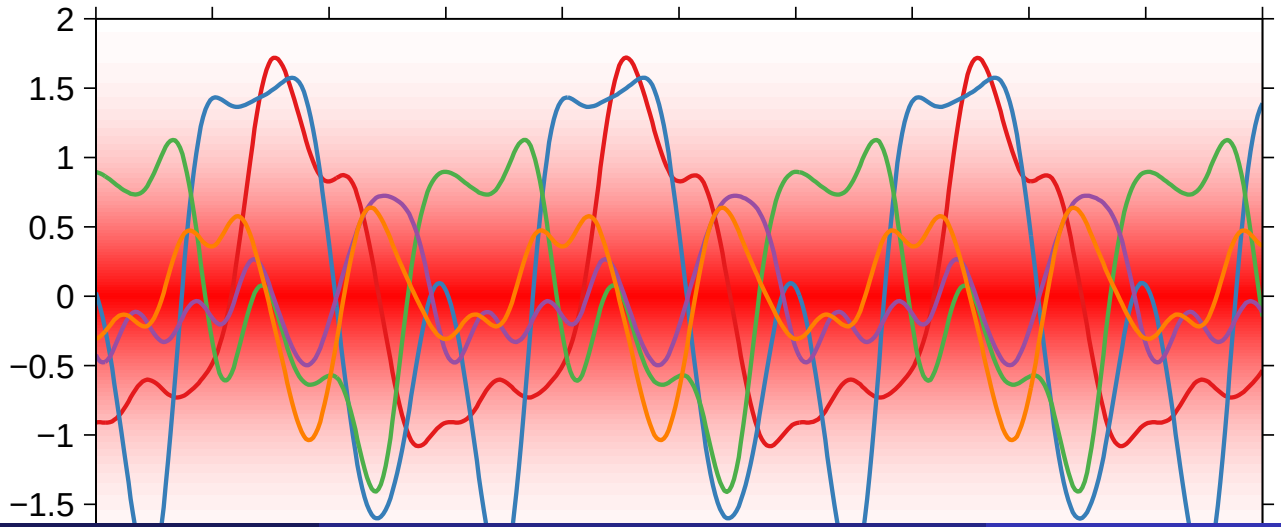


- Each function evaluation takes about an hour
- Human expert = Alex Krizhevsky, the creator of AlexNet

GP Kernels (optional)

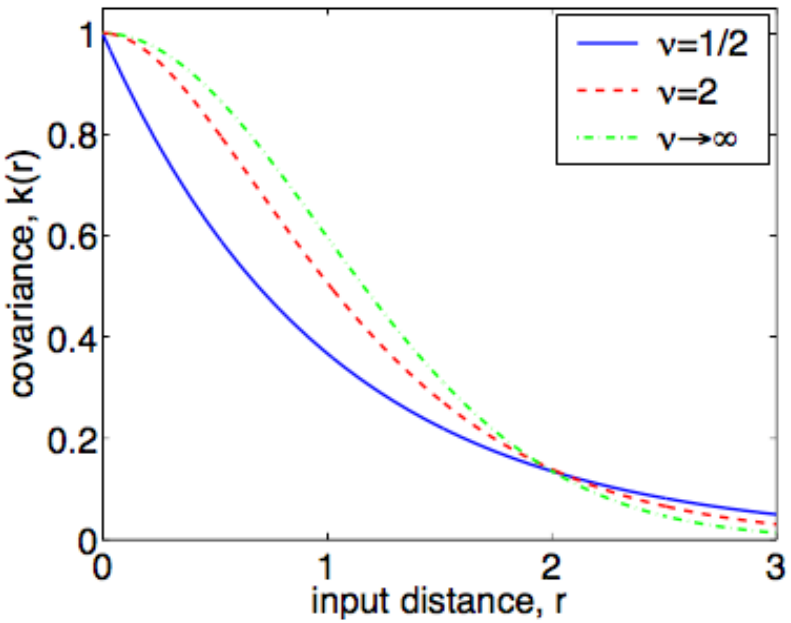
- The periodic kernel encodes for a probability distribution over periodic functions
- The linear kernel results in a probability distribution over linear functions



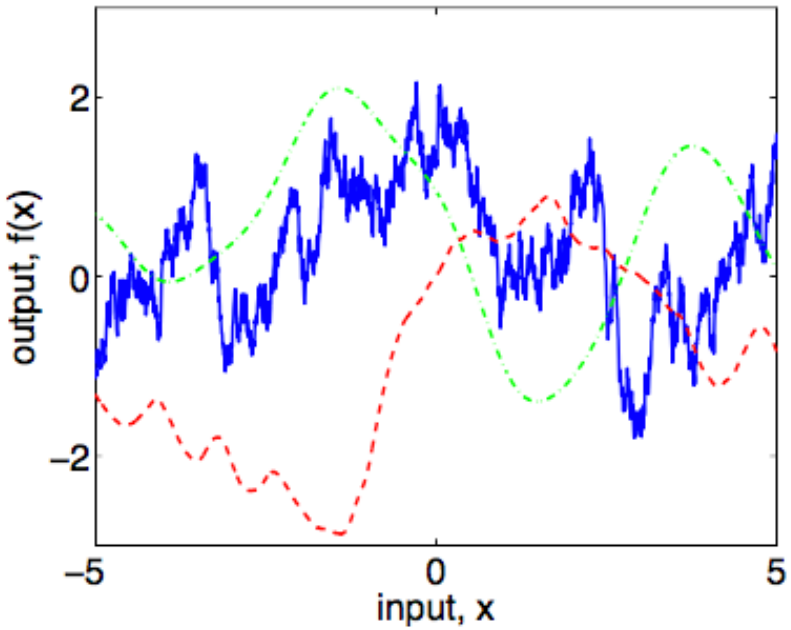


GP Kernels (optional)

- The Matern kernel is similar to the squared-exp kernel, but less smooth.
- See Chapter 4 of GPML for an explanation (advanced).
- Imagine trying to get this behavior by designing basis functions!



(a)



(b)

GP Kernels (optional)

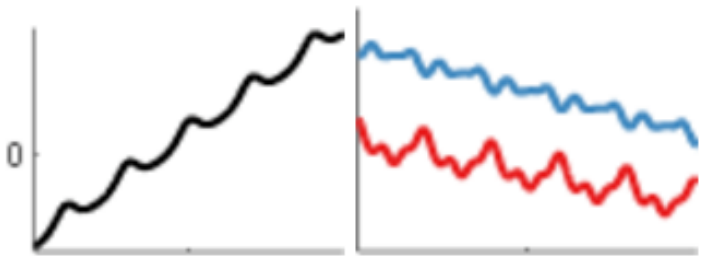
- We get exponentially more flexibility by combining kernels.
- The sum of two kernels is a kernel.
  - This is because valid covariance matrices (i.e. PSD matrices) are closed under addition.
- The sum of two kernels corresponds to the sum of functions.

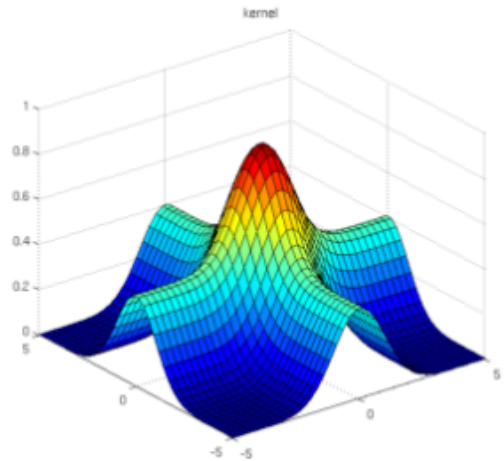
Additive kernel

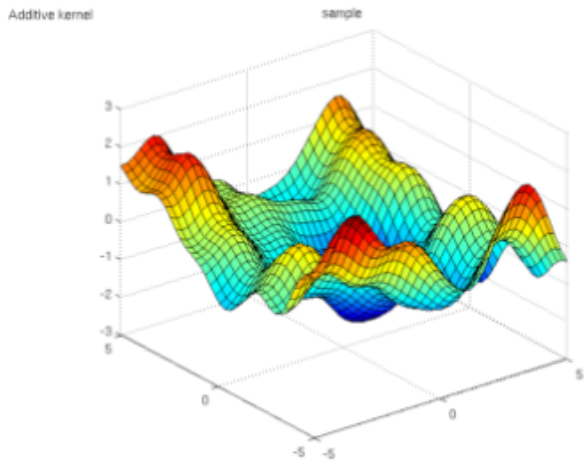
$$k(x, y, x', y') = k_1(x, x') + k_2(y, y')$$

Linear + Periodic

e.g. seasonal pattern w/ trend

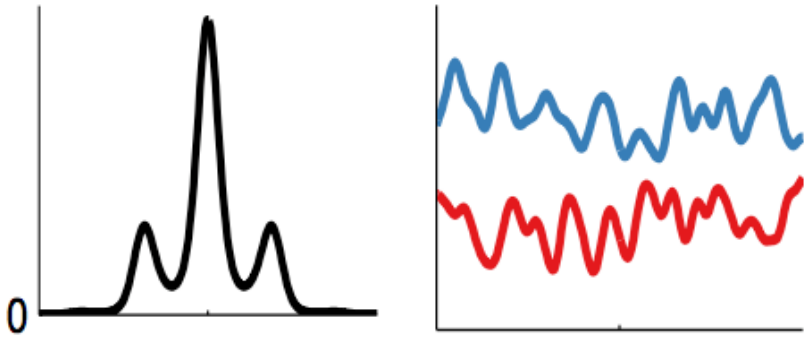






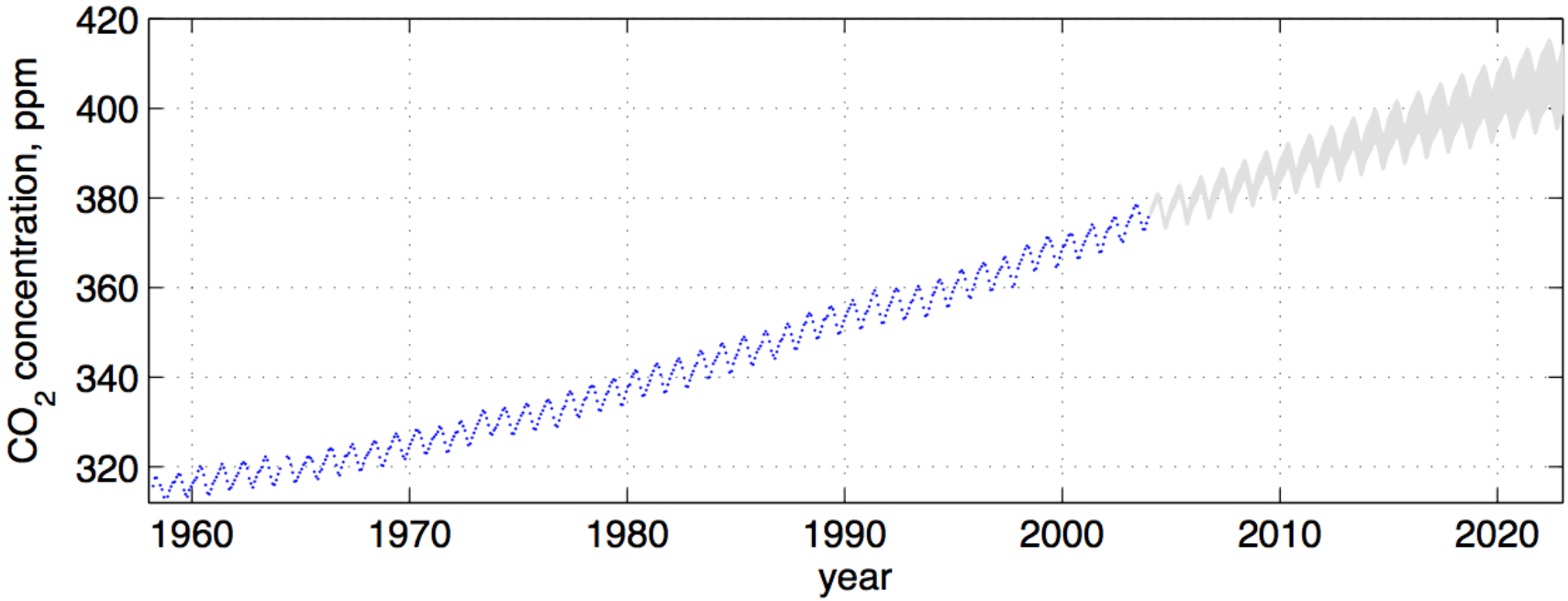
GP Kernels (optional)

- A kernel is like a similarity function on the input space. The sum of two kernels is like the OR of their similarity.
- Amazingly, the product of two kernels is a kernel. (Follows from the Schur Product Theorem.)
- The product of two kernels is like the AND of their similarity functions.
- Example: the product of a squared-exp kernel (spatial similarity) and a periodic kernel (similar location within cycle) gives a locally periodic function.



GP Kernels (optional)

- Modeling CO2 concentrations:
  trend + (changing) seasonal pattern + short-term variability + noise
- Encoding the structure allows sensible extrapolation.



Summary

- Bayesian approach to regression lets us determine uncertainty in our predictions.
- Gaussian processes are an elegant framework for doing Bayesian inference directly over functions.
- The choice of kernels gives us much more control over what sort of functions our prior would allow or favor.