

STA 4273H: Statistical Machine Learning

Russ Salakhutdinov

Department of Statistics

rsalakhu@utstat.toronto.edu

<http://www.utstat.utoronto.ca/~rsalakhu/>

Sidney Smith Hall, Room 6002

Lecture 10

Gaussian Processes

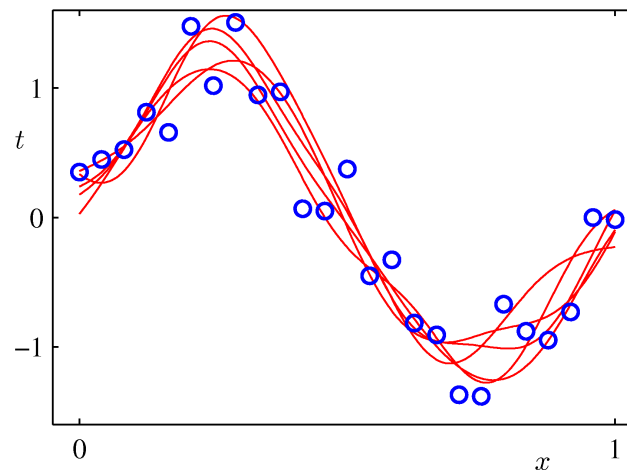
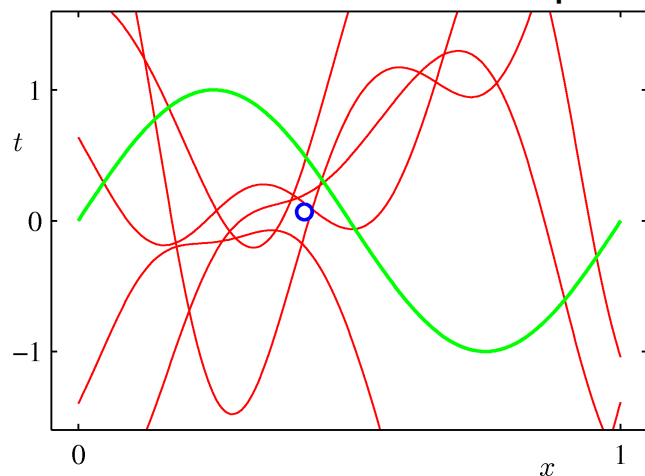
- So far, we have considered **linear regression models** of the form:

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x})$$

where \mathbf{w} is a vector of parameters and $\phi(\mathbf{x})$ is a vector of fixed nonlinear basis functions.

- A prior distribution over \mathbf{w} induces a **prior distribution over functions** $f(\mathbf{x}, \mathbf{w})$.
- Given a training dataset, we compute the posterior distribution over \mathbf{w} , which induces a **posterior distribution** over functions $f(\mathbf{x}, \mathbf{w})$.

Samples from the posterior



Gaussian Processes

- In the Gaussian process viewpoint, we define a **prior probability distributions over functions directly**.
 - May seem difficult: How can we define a distribution over the uncountably infinite space of functions?
 - **Insight**: for a finite training set, we only need to consider the values of the functions at discrete set of input values x_n .
 - Hence in practice, we work in a **finite space**.
-
- Many related models: In geostatistics literature, GP regression is known as kriging. See also a recent book on GPs by Rasmussen & Williams (2006).

Linear Regression Revisited

- Consider the following linear model, defined in terms of M linear combinations of fixed basis functions:

$$f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \phi(\mathbf{x})$$

- We place a Gaussian prior over model parameters:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \alpha^{-1} \mathbf{I})$$

- For any given fixed value of \mathbf{w} , we have a corresponding linear function. A probability distribution over \mathbf{w} defines a **probability distribution over functions**.

- Given a dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, we will denote the values of the function as $\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_N)]^T$.

- Hence:

$$\mathbf{f} = \Phi \mathbf{w}.$$

N by M Design matrix

M by 1 vector of model parameters

Linear Regression Revisited

$$\mathbf{f} = \Phi \mathbf{w}, \quad p(\mathbf{w}) = \mathcal{N}(\mathbf{w} | \mathbf{0}, \alpha^{-1} \mathbf{I})$$

- Observe that \mathbf{y} is a linear combination of Gaussian random variables, and hence is **itself Gaussian**:

$$\mathbb{E}[\mathbf{f}] = \Phi \mathbb{E}[\mathbf{w}] = \mathbf{0}$$

$$\text{cov}(\mathbf{f}) = \mathbb{E}[\mathbf{f}\mathbf{f}^T] = \Phi \mathbb{E}[\mathbf{w}\mathbf{w}^T] \Phi^T = \frac{1}{\alpha} \Phi \Phi^T = \mathbf{K}$$

Here, \mathbf{K} is known as the **Gramm matrix** with elements:

$$\mathbf{K}_{nm} = k(\mathbf{x}_n, \mathbf{x}_m) = \frac{1}{\alpha} \phi(\mathbf{x}_n)^T \phi(\mathbf{x}_m),$$

where $k(\mathbf{x}, \mathbf{x}')$ is the **kernel function**.

- This model provides a **particular example of a Gaussian process**.

Gaussian Process

- A Gaussian process (GP) is a **random function** $f: \mathbf{X} \rightarrow \mathbb{R}$, such that for any **finite set** of input points $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$,

$$\begin{bmatrix} f(\mathbf{x}_1) \\ \vdots \\ f(\mathbf{x}_N) \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} m(\mathbf{x}_1) \\ \vdots \\ m(\mathbf{x}_N) \end{bmatrix}, \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \cdots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \cdots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix} \right)$$

where the parameters are the **mean function** $m(\mathbf{x})$ and **covariance kernel** $k(\mathbf{x}, \mathbf{x}')$.

- Note that a **random function is a stochastic process**. It is a collection of random variables $\{f(\mathbf{x})\}_{\mathbf{x} \in \mathcal{X}}$, one for each possible value \mathbf{x} (see Rasmussen and Williams, 2006).
- **Key point about Gaussian Processes:** Given a dataset $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, the marginal distribution over $[f(\mathbf{x}_1), f(\mathbf{x}_2), \dots, f(\mathbf{x}_N)]$ is completely specified by the second-order statistics: the mean and covariance.

Gaussian Process

- In many applications, we will have no prior knowledge about the mean function $f(\mathbf{x})$. By symmetry, **we take it be zero**.
- The specification of a Gaussian Process is then completed by **specifying the covariance function**, evaluated at any two input points \mathbf{x}_n and \mathbf{x}_m :

$$\mathbb{E}[f(\mathbf{x}_n)f(\mathbf{x}_m)] = k(\mathbf{x}_n, \mathbf{x}_m).$$

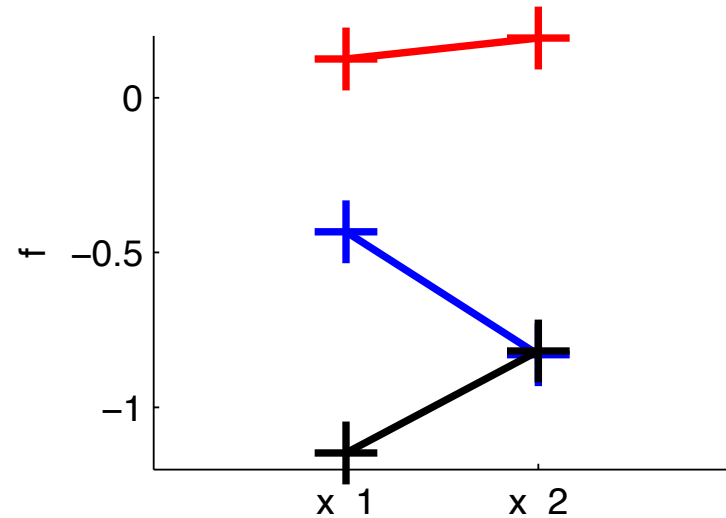
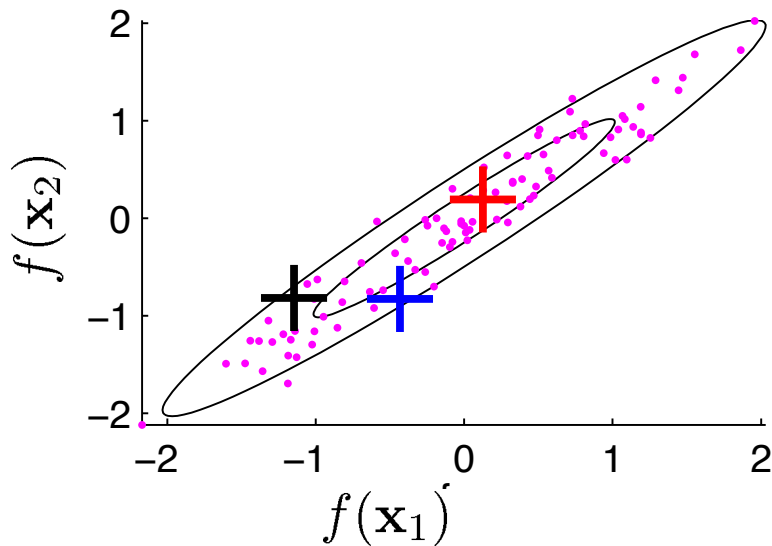
- One commonly used covariance function is squared exponential:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \exp\left(-\frac{\theta}{2}\|\mathbf{x}_n - \mathbf{x}_m\|^2\right)$$

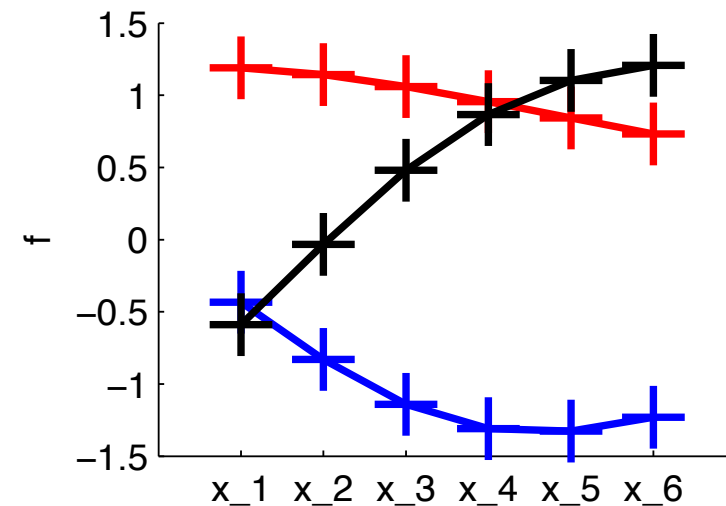
- Covariance (kernel) function is typically chosen to express the property that, for **inputs \mathbf{x}_n and \mathbf{x}_m that are similar**, the corresponding values $f(\mathbf{x}_n)$ and $f(\mathbf{x}_m)$ will **be more strongly correlated** than for dissimilar points.

Visualizing Draws from GPs

- Visualizing draws from 2-D Gaussian:

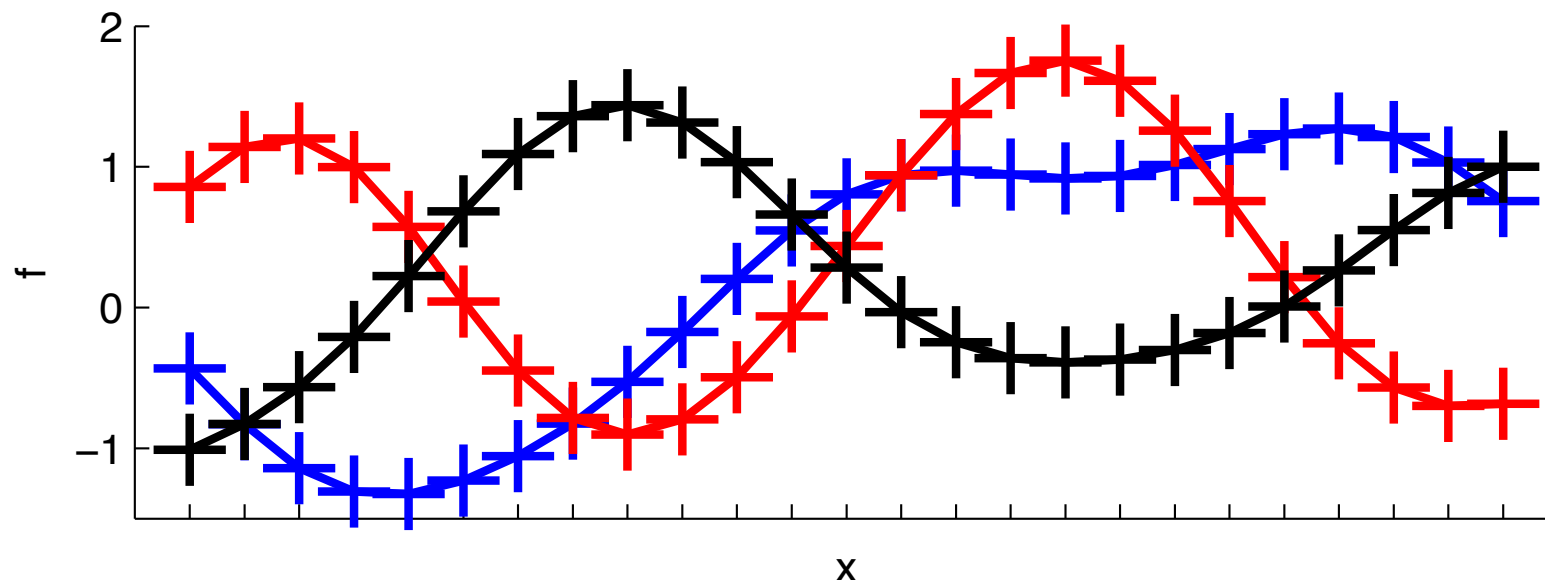


- Three draws from a 6-D Gaussian:



Visualizing Draws from GPs

- Three draws from 25-D Gaussian

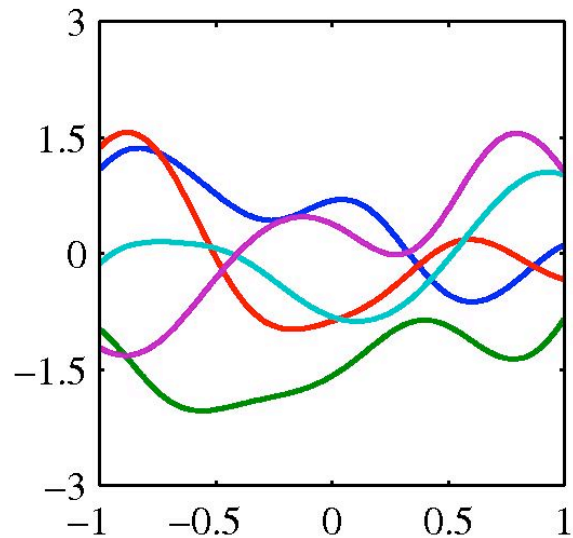


- To generate these, the mean was set to zero: `zeros(25,1)`
- The covariance was set using a covariance function: $\Sigma_{nm} = k(\mathbf{x}_n, \mathbf{x}_m)$.
- The x 's are the positions that are planted the ticks on the axis.

We can visualize draws from a GP iterative sampling $f(x_n) \mid f(x_1), \dots, f(x_{n-1})$ on a sequence of input points x_1, x_2, \dots, x_n .

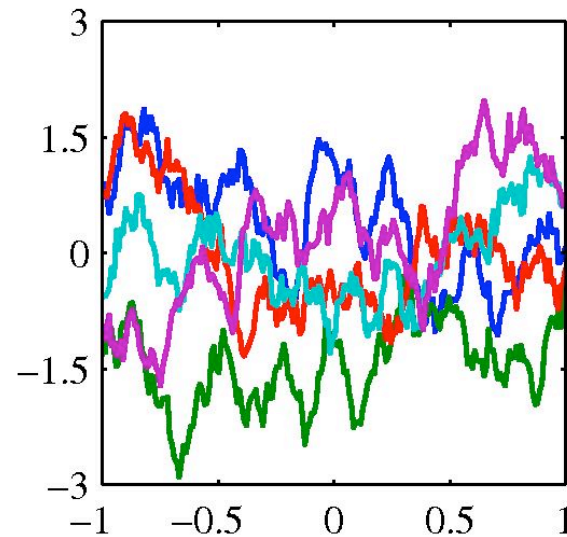
Samples from GPs

Squared-exponential kernel



$$k(x_n, x_m) = \exp\left(-\frac{\theta}{2}(x_n - x_m)^2\right)$$

Exponential kernel



$$k(x_n, x_m) = \exp(-\theta|x_n - x_m|)$$

- Ornstein-Uhlenbeck process that describes Brownian motion.

GPs for Regression

- We need to account for **noise on the observed target values**:

$$t_n = f_n + \epsilon_n,$$

where $f_n = f(\mathbf{x}_n)$, and ϵ_n is an **independent random noise variable**. We will assume Gaussian noise:

$$p(t_n | f_n) = \mathcal{N}(t_n | f_n, \beta^{-1}).$$

- Given a dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, and corresponding target values $\mathbf{t} = \{t_1, t_2, \dots, t_N\}$, the conditional takes form:

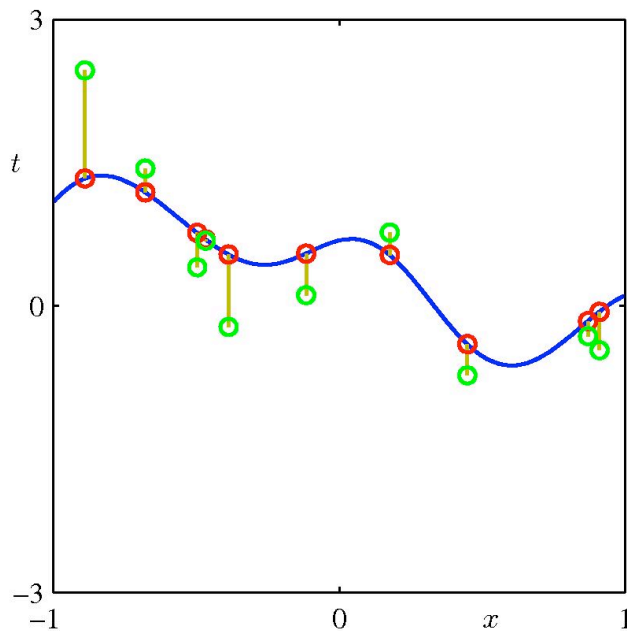
$$p(\mathbf{t} | \mathbf{f}) = \mathcal{N}(\mathbf{t} | \mathbf{f}, \beta^{-1} \mathbf{I}_N).$$

- From the definitions of a Gaussian process, **the marginal distribution $p(\mathbf{f})$** is given by the Gaussian:

$$p(\mathbf{f}) = \mathcal{N}(\mathbf{f} | \mathbf{0}, \mathbf{K}).$$

Illustration

- Illustration of sampling of targets $\{t_n\}$ from a Gaussian process.



- The blue curve shows a sample from a GP prior:

$$f \sim \mathcal{GP}$$

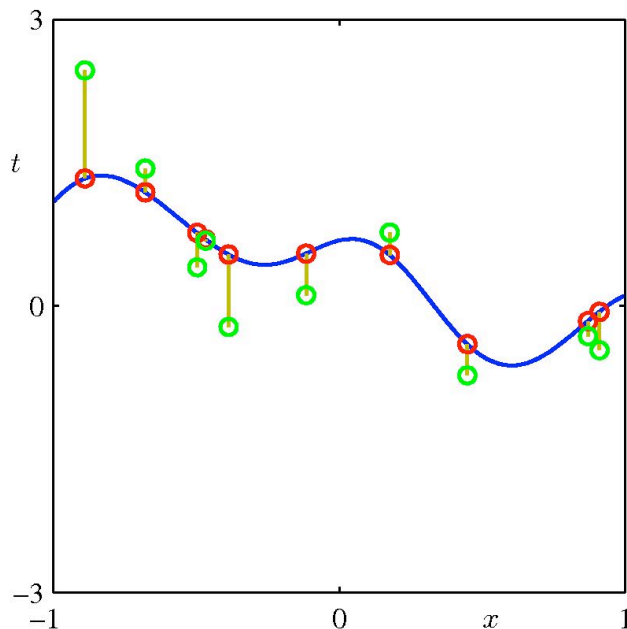
- The red points show the values of f_n , obtained by evaluating the function at a set of input values $\{x_n\}$.

- The green points show the corresponding values of $\{t_n\}$:

$$p(t_n | f_n) = \mathcal{N}(t_n | f_n, \beta^{-1}).$$

Marginal Distribution

- The **marginal distribution** $p(\mathbf{t})$, conditioned on the set of inputs \mathbf{X} , can be obtained by integrating over \mathbf{f} :



$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{f})p(\mathbf{f})d\mathbf{f} = \mathcal{N}(\mathbf{t}|\mathbf{0}, \mathbf{C}),$$

where the covariance matrix is given by:

$$C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{nm}.$$

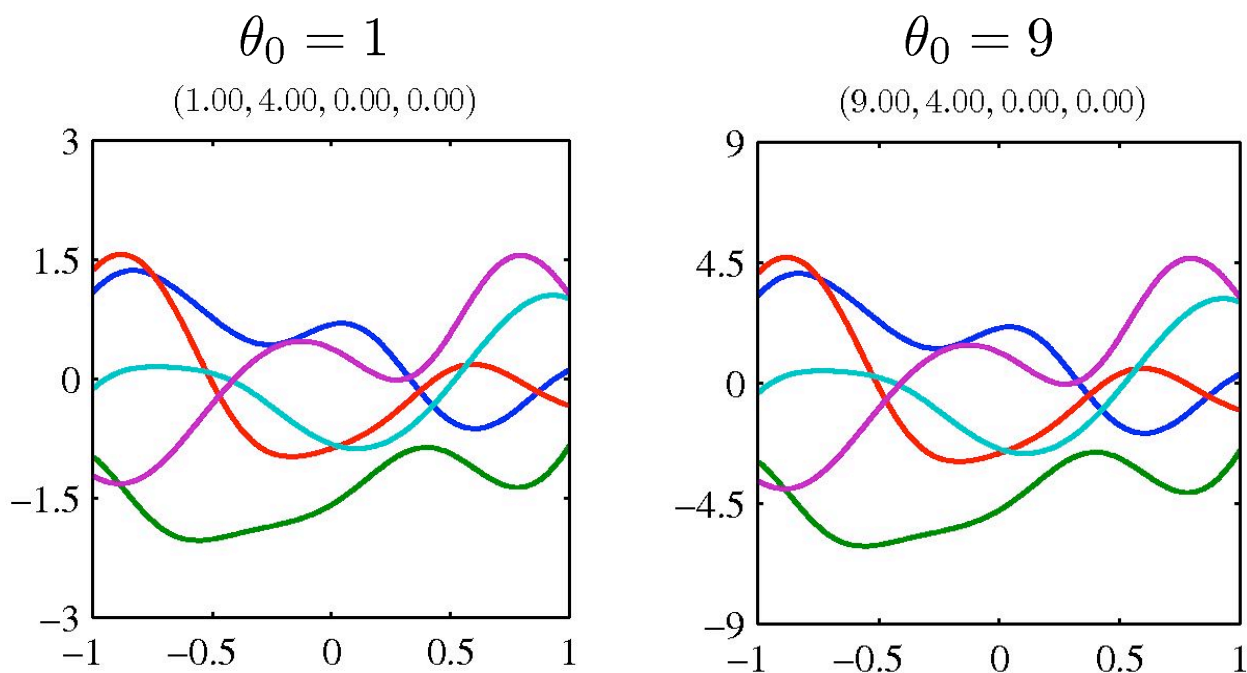
- The **two Gaussian sources of randomness**, one associated with $\mathbf{f}(\mathbf{x})$ and the other with noise, are independent, and so their covariances add.

Covariance Function

- One widely used covariance (kernel) function for GP regression is given by the **squared-exponential** plus constant and linear terms:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \|\mathbf{x}_n - \mathbf{x}_m\|^2\right) + \theta_2 + \theta_3 \mathbf{x}_n^T \mathbf{x}_m$$

- Note that the last term corresponds to a parametric model that is a **linear function of the input variables**.



Covariance Function

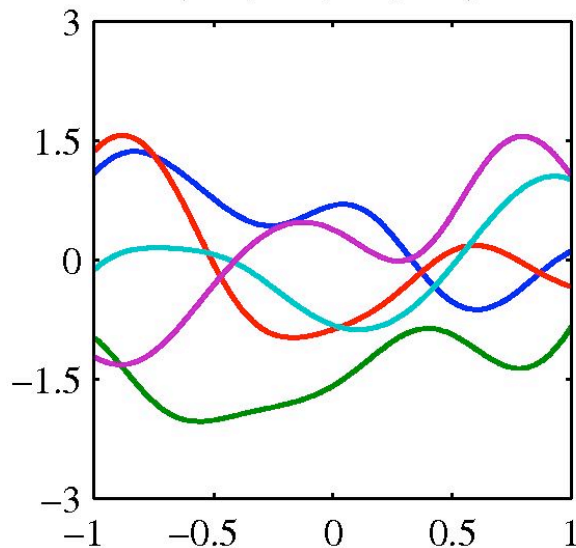
- One widely used covariance (kernel) function for GP regression is given by the **squared-exponential** plus constant and linear terms:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \|\mathbf{x}_n - \mathbf{x}_m\|^2\right) + \theta_2 + \theta_3 \mathbf{x}_n^T \mathbf{x}_m$$

- Note that the last term corresponds to a parametric model that is a **linear function of the input variables**.

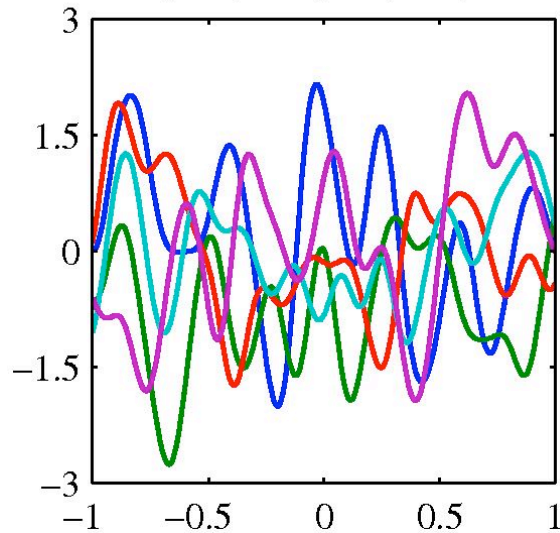
$$\theta_1 = 1$$

(1.00, 4.00, 0.00, 0.00)



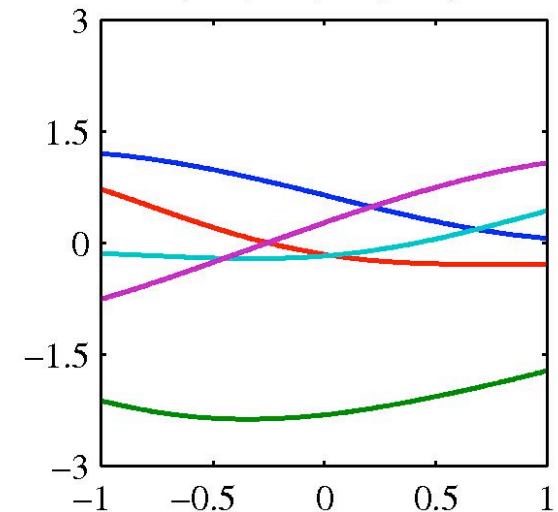
$$\theta_1 = 64$$

(1.00, 64.00, 0.00, 0.00)



$$\theta_1 = 0.25$$

(1.00, 0.25, 0.00, 0.00)



Covariance Function

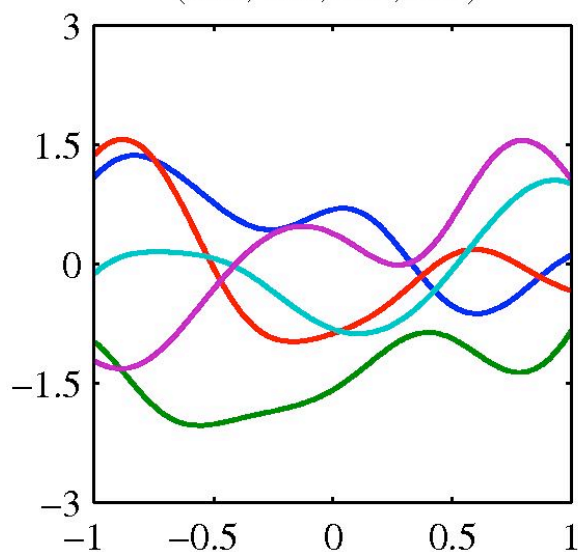
- One widely used covariance (kernel) function for GP regression is given by the **squared-exponential** plus constant and linear terms:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \|\mathbf{x}_n - \mathbf{x}_m\|^2\right) + \theta_2 + \theta_3 \mathbf{x}_n^T \mathbf{x}_m$$

- Note that the last term corresponds to a parametric model that is a **linear function of the input variables**.

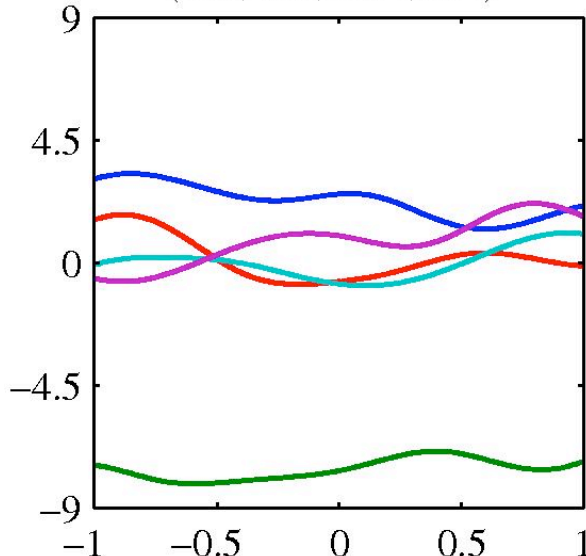
$$\theta_2 = 0, \theta_3 = 0$$

(1.00, 4.00, 0.00, 0.00)



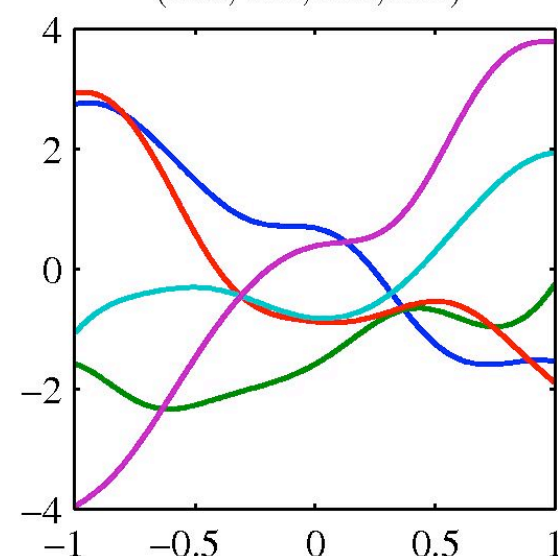
$$\theta_2 = 10, \theta_3 = 0$$

(1.00, 4.00, 10.00, 0.00)



$$\theta_2 = 0, \theta_3 = 5$$

(1.00, 4.00, 0.00, 5.00)



Prediction

- Suppose we are given a dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, with target values $\mathbf{t} = \{t_1, t_2, \dots, t_N\}$.
- Our goal is predict t_{N+1} for a new input vector \mathbf{x}_{N+1} .
- Note that the joint distribution over \mathbf{t} and t_{N+1} is given by:

$$P\left(\begin{bmatrix} \mathbf{t} \\ t_{N+1} \end{bmatrix}\right) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^T & c \end{bmatrix}\right)$$

where \mathbf{C}_N is the **N by N matrix** with elements:

$$C_N(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1} \delta_{nm}.$$

c is the **scalar**:

$$c = k(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) + \beta^{-1}$$

and \mathbf{k} is the **N by 1 vector** with elements $k(\mathbf{x}_n, \mathbf{x}_{N+1})$.

Prediction

- Suppose we are given a dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, with target values $\mathbf{t} = \{t_1, t_2, \dots, t_N\}$.
- Our goal is predict t_{N+1} for a new input vector \mathbf{x}_{N+1} .
- Note that the joint distribution over \mathbf{t} and t_{N+1} is given by:

$$P\left(\begin{bmatrix} \mathbf{t} \\ t_{N+1} \end{bmatrix}\right) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{C}_N & \mathbf{k} \\ \mathbf{k}^T & c \end{bmatrix}\right)$$

- Hence the conditional distribution is Gaussian:


$$P(t_{N+1}|\mathbf{t}) = \mathcal{N}(m(\mathbf{x}_{N+1}), \sigma^2(\mathbf{x}_{N+1}))$$

with

$$m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{t}$$

$$\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{k}$$

Key results that define
GP regression



Positive: hence the
reduction in uncertainty




Illustration 1

- We are given **one training point, t_1** and **one test point**. Conditioned on t_1 (blue line), we obtain predictive distribution $p(t_2 | t_1)$ (green curve)

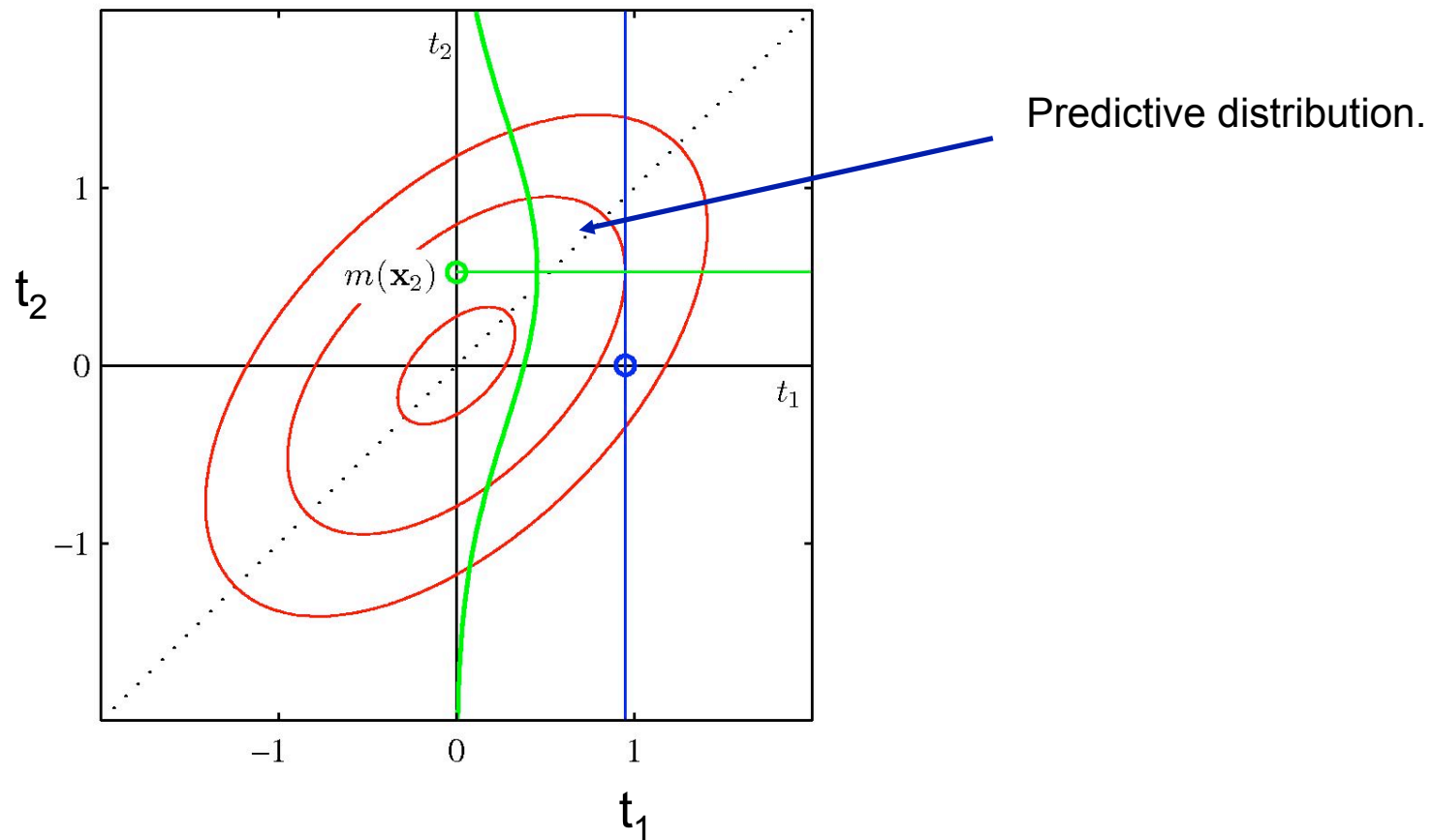
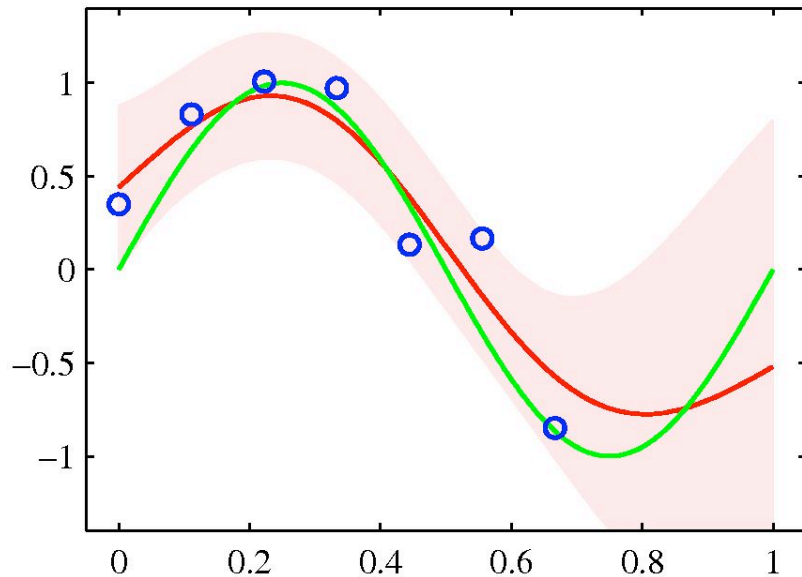


Illustration 2

- Illustration of GP regression applied to the sinusoidal data set.



- **The green curve** shows the true function.
- **The blue data points** are samples from the true function plus some additive Gaussian noise
- **The red curve shows** the mean of the GP predictive distribution, with shaded region corresponding to +/- 2 standard deviations.

- **Restriction on the kernel function:** The covariance matrix:

$$C(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1} \delta_{nm}.$$

must be positive definite.

Mean of Predictive Distribution

- Note that the **mean of the predictive distribution**

$$m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{t}$$

can be written as a function of \mathbf{x}_{N+1} :

Linear combination

$$m(\mathbf{x}_{N+1}) = \sum_{n=1}^N a_n k(\mathbf{x}_n, \mathbf{x}_{N+1})$$

a_n is the n^{th} component of $\mathbf{C}_N^{-1} \mathbf{t}$

- Also, note that **the mean and variance** of the predictive distribution both depend on \mathbf{x}_{N+1} .

$$m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{t}$$

$$\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{k}$$

Remember: \mathbf{k} is the N by 1 vector with elements $k(\mathbf{x}_n, \mathbf{x}_{N+1})$.

Computational Complexity

- The central computation in using GPs will involve the inversion of an N by N matrix \mathbf{C}_N , which is of order $O(N^3)$:

$$m(\mathbf{x}_{N+1}) = \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{t}$$
$$\sigma^2(\mathbf{x}_{N+1}) = c - \mathbf{k}^T \mathbf{C}_N^{-1} \mathbf{k}$$

- By contrast, in the basis function model, we have to invert a matrix \mathbf{S}_N of size M by M (where M is the number of basis functions).
- If the number of M basis functions is smaller than the number N of data points, then it will be computationally more efficient to work in the basis function framework (see the first few slides)
- The advantage of GPs is that we can consider covariance functions that can only be expressed in terms of an infinite number of basis functions.

Learning the Hyperparameters

- The predictions of a GP regression model will depend on the **choice of the covariance function**.
- Instead of fixing the covariance function, we may prefer to use a parametric family of functions and **infer the parameter values from data**.
- These parameters may govern the length scale of the correlations or the precision of the noise model and **correspond to the hyperparameters in a standard parametric model**.

hyperparameters

The diagram shows the word "hyperparameters" at the top center. Four blue arrows point downwards from it to the parameters θ_0 , θ_1 , θ_2 , and θ_3 in the equation below. θ_1 is specifically associated with the exponent of the exponential function.

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \|\mathbf{x}_n - \mathbf{x}_m\|^2\right) + \theta_2 + \theta_3 \mathbf{x}_n^T \mathbf{x}_m$$

- How can we infer the values of these parameters?

Learning the Hyperparameters

- We can compute the **marginal likelihood function**:

$$p(\mathbf{t}|\theta) = \int p(\mathbf{t}|\mathbf{f}, \theta)p(\mathbf{f}|\theta)d\mathbf{f} = \mathcal{N}(\mathbf{t}|\mathbf{0}, \mathbf{C}_N),$$

Hyperparameters of the GP model

- One option is to **maximize the log of the marginal likelihood** with respect to θ .

$$\ln p(\mathbf{t}|\theta) = -\frac{1}{2} \ln |\mathbf{C}_N| - \frac{1}{2} \mathbf{t}^T \mathbf{C}_N^{-1} \mathbf{t} - \frac{N}{2} \ln(2\pi).$$

- This corresponds to the **type II maximum likelihood**, or **empirical Bayes**:
- The maximization can be performed using **gradient-based optimization techniques**, such as conjugate gradients. The gradients take form:

$$\frac{\partial}{\partial \theta_i} \ln p(\mathbf{t}|\theta) = -\frac{1}{2} \text{Tr} \left(\mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta_i} \right) + \frac{1}{2} \mathbf{t}^T \mathbf{C}_N^{-1} \frac{\partial \mathbf{C}_N}{\partial \theta_i} \mathbf{C}_N^{-1} \mathbf{t}.$$

Learning the Hyperparameters

$$\ln p(\mathbf{t}|\theta) = -\frac{1}{2} \ln |\mathbf{C}_N| - \frac{1}{2} \mathbf{t}^T \mathbf{C}_N^{-1} \mathbf{t} - \frac{N}{2} \ln(2\pi).$$

- Because $\ln p(\mathbf{t}|\theta)$ will be a **nonconvex function**, it will have **multiple maxima**.
- In the **fully Bayesian approach**, we can introduce a prior $p(\theta)$ and infer the posterior $p(\theta | \mathbf{t})$.
- In general, the posterior will not have a closed form solution, so we must resort of approximations (typically MCMC).
- **Noise**: We have assumed that the additive noise, governed by β , **is constant**.

$$p(t_n | f_n) = \mathcal{N}(t_n | f_n, \beta^{-1}).$$

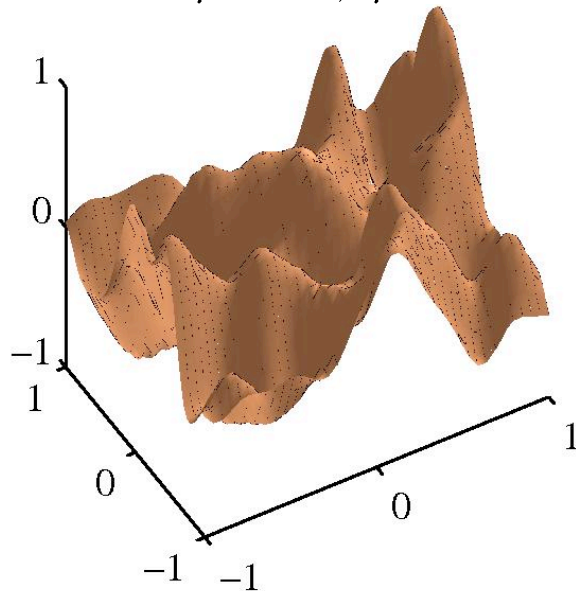
- For some models, known as **heteroscedastic**, the noise variance itself will depend on \mathbf{x} (e.g. by introducing another GP that will model $\log \beta(\mathbf{x})$).

Automatic Relevance Determination

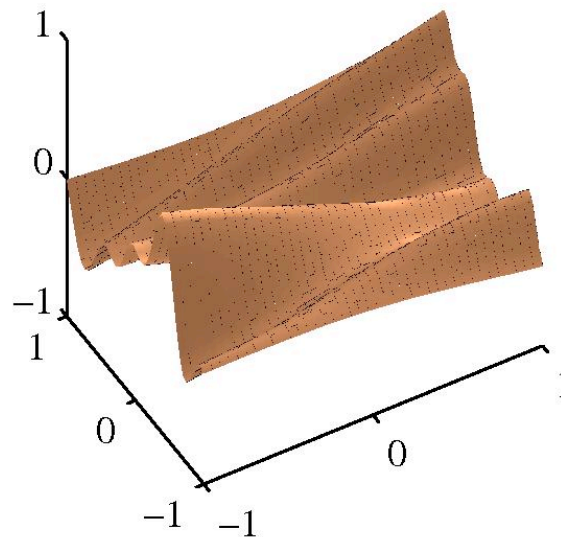
- How can we detect inputs variables that have **very little effect on the predictive distribution** (irrelevant inputs).
- Consider a GP with 2-D input space $\mathbf{x} = (x_1, x_2)$ with the following covariance function:

$$k(\mathbf{x}, \mathbf{x}') = \theta_0 \exp \left(-\frac{\theta_1}{2} \sum_{i=1}^2 \eta_i (x_i - x'_i)^2 \right)$$

$$\eta_1 = 1, \eta_2 = 1$$



$$\eta_1 = 1, \eta_2 = 0.01$$



- As η_i becomes small, the function becomes insensitive to the corresponding value of x_i (input x_i becomes less relevant).

Automatic Relevance Determination

- The ARD framework can be easily incorporated into **exponential-quadratic kernel**:

$$k(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp \left(-\frac{1}{2} \sum_{i=1}^D \eta_i (x_{ni} - x_{mi})^2 \right) + \theta_2 + \theta_3 \mathbf{x}_n^T \mathbf{x}_m$$

Control relevance of input dimension i , where D is the dimensionality of the input space

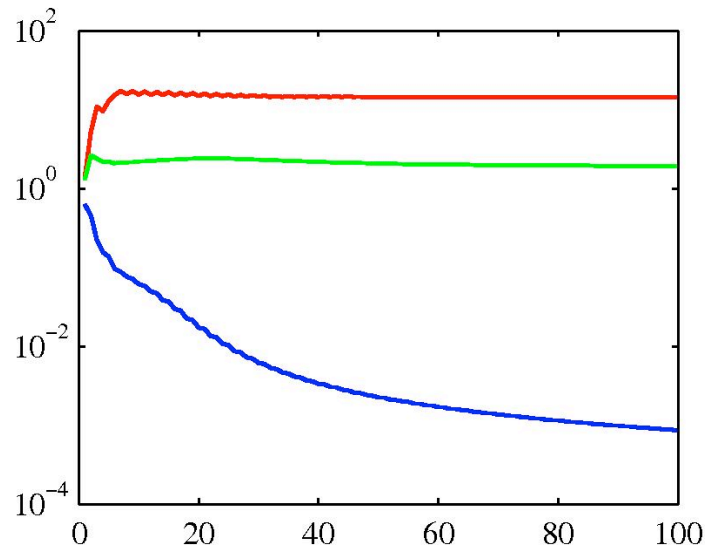
- We can optimize these parameters by performing **type II maximum likelihood** (by optimizing marginal log-likelihood)

$$\ln p(\mathbf{t}|\theta) = -\frac{1}{2} \ln |\mathbf{C}_N| - \frac{1}{2} \mathbf{t}^T \mathbf{C}_N^{-1} \mathbf{t} - \frac{N}{2} \ln(2\pi).$$

- The relative importance of different inputs can be **inferred from data**.

Illustration

- **Example:** We have a dataset with 3-D inputs (x_1, x_2, x_3) . The target variables t_n are sampled as follows:
 - Sample 100 values of x_1 from a Gaussian, **evaluate the function** $\sin(2\pi x_1)$, and add Gaussian noise.
 - Let $x_2 = x_1$, and **add Gaussian noise**.
 - Sample 100 values of x_3 from an **independent Gaussian distribution**.
- Hence x_1 is a good predictor of t , x_2 is a more noisy predictor of t , and x_3 has only chance correlation with t .



- Plot displays η_1 (red), η_2 (green), and η_3 (blue) as a function of the number of iterations when optimizing the marginal likelihood.

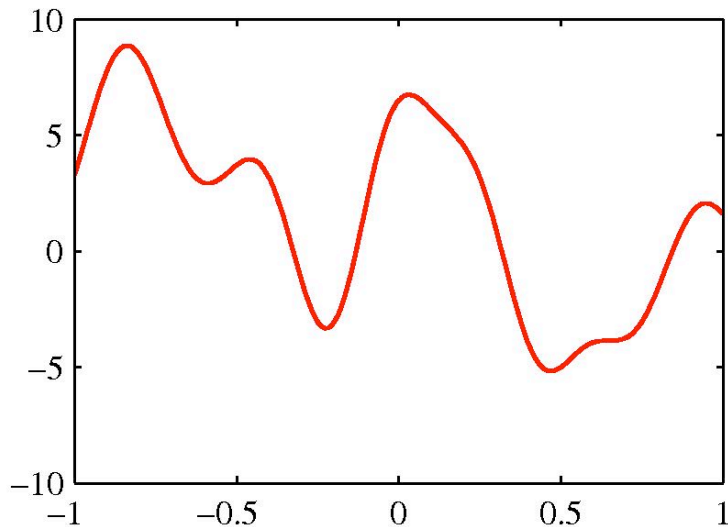
Classification with GPs

- Consider a two-class problem with targets $t \in \{0,1\}$.
- Define a Gaussian process over a function $f(\mathbf{x})$.
- Transform the function using sigmoid function:

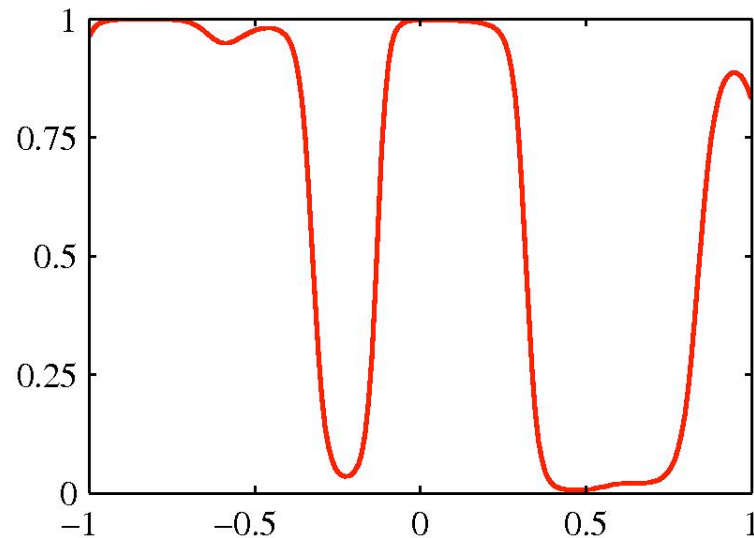
$$y(\mathbf{x}) = \sigma(f(\mathbf{x})) = \frac{1}{1 + \exp(-f(\mathbf{x}))}$$

- Hence $y(\mathbf{x}) \in (0,1)$.

$f \sim \mathcal{GP}$



Transformed sample using sigmoid function

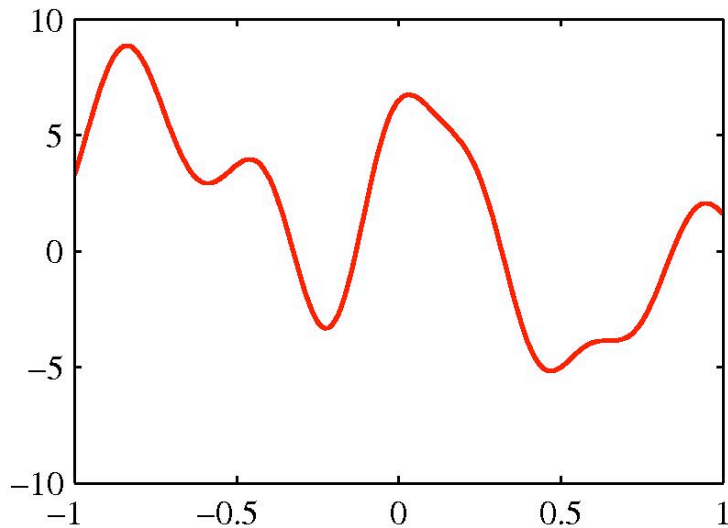


Classification with GPs

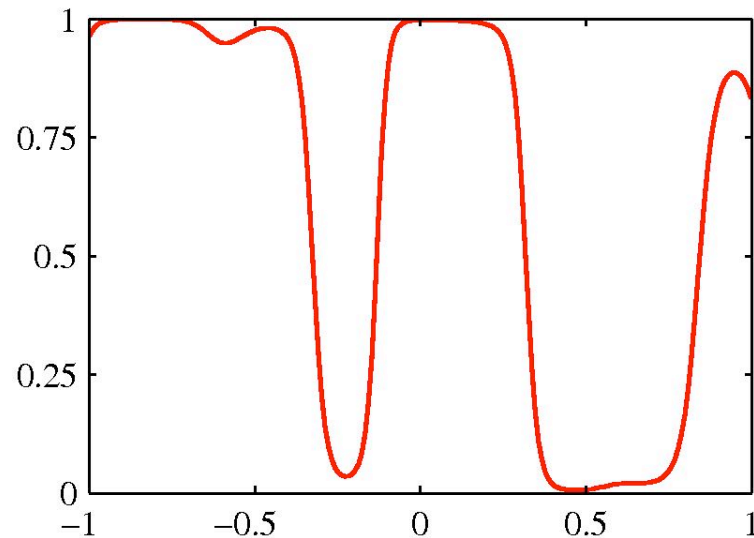
- After transformation, we obtain a **non-Gaussian stochastic process over functions** $y(\mathbf{x})$.
- The probability distribution over t is given by the **Bernoulli distribution**:

$$p(t|f) = \sigma(f)^t (1 - \sigma(f))^{1-t}.$$

$f \sim GP$



Transformed sample using sigmoid function



Classification with GPs

- Suppose we are given a dataset $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$, with target values $\mathbf{t}_N = \{t_1, t_2, \dots, t_N\}$.
- Our goal is predict t_{N+1} for a new input vector \mathbf{x}_{N+1}
- Predictive distribution is given by:

$$p(t_{N+1} = 1 | \mathbf{t}_N) = \int p(t_{N+1} = 1 | f_{N+1}) p(f_{N+1} | \mathbf{t}_N) df_{N+1}$$

given by $\sigma(f(\mathbf{x}_{N+1}))$

Posterior is also intractable.

- This integral is **analytically intractable**. Can resort to MCMC by approximately **sampling from the posterior**, and performing Monte Carlo integration:

$$p(t_{N+1} = 1 | \mathbf{t}) = \frac{1}{M} \sum_m p(t_{N+1} = 1 | f_{N+1}^{(m)})$$

where $f_{N+1}^{(m)} \sim p(f_{N+1} | \mathbf{t}_N)$

Approximations

- Another option:

$$p(t_{N+1} = 1 | \mathbf{t}_N) = \int p(t_{N+1} = 1 | f_{N+1}) p(f_{N+1} | \mathbf{t}_N) df_{N+1}$$



Gaussian approximation

- Use approximate formula for the convolution of a logistic sigmoid and a Gaussian distribution.
- Three different approaches to obtaining a Gaussian approximation:
 - Variational Inference
 - Expectation Propagation
 - Laplace Approximation

Laplace Approximation

- We seek to obtain a **Gaussian approximation to the posterior**. Using Bayes rule we have:

$$p(f_{N+1} | \mathbf{t}_N) = \int p(f_{N+1}, \mathbf{f}_N | \mathbf{t}_N) d\mathbf{f}_N = \int p(f_{N+1} | \mathbf{f}_N) p(\mathbf{f}_N | \mathbf{t}_N) d\mathbf{f}_N$$

Easy to compute:
Gaussian.

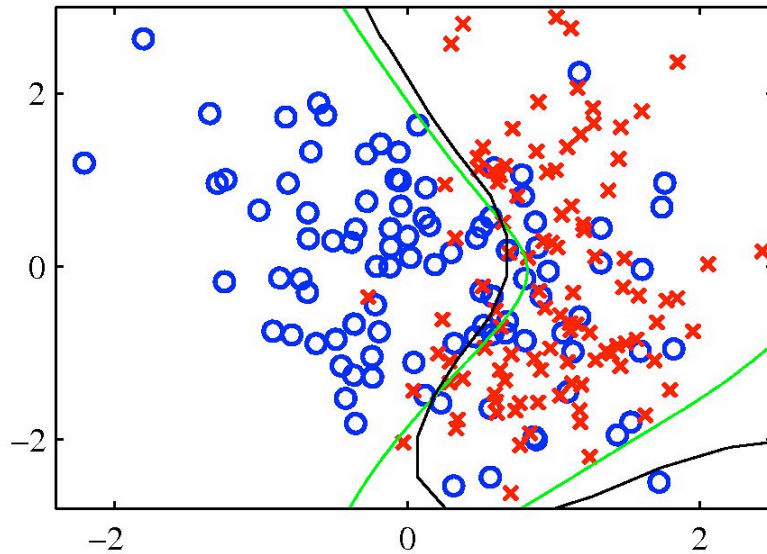
Laplace approximation

- Here $p(\mathbf{f}_N)$ is given by a **zero-mean GP with covariance matrix \mathbf{C}_N** , and the data term:

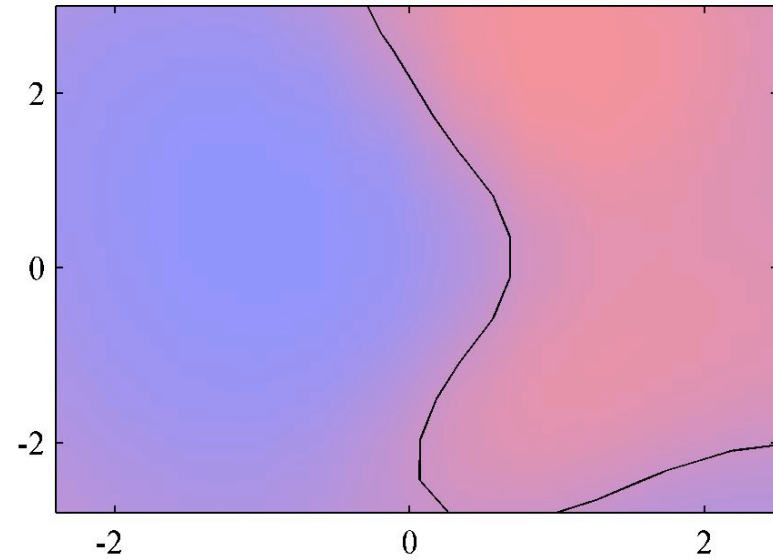
$$p(\mathbf{t}_N | \mathbf{f}_N) = \prod_{n=1}^N \sigma(f_n)^{t_n} (1 - \sigma(f_n))^{1-t_n}$$

- Obtain the **Laplace approximation** by Taylor expanding log of the posterior: $\log p(\mathbf{f}_N | \mathbf{t}_N)$.

Classification Results



Optimal decision boundary from the true distribution (green) and the decision boundary from GP classifier (black)



Predictive posterior probability together with GP decision boundary.

Combining Models

- In practice, it is often found that one can improve performance by combining multiple models, instead of using a single model.
- **Example:** We may train K different models and then **make predictions using the average of predictions made by each model.**
- Such combinations of models are called **committees**.
- One important variant of the committee method is called **boosting**.
- Another approach is to use different models in different regions of the input space.
- One widely used framework is known as a decision tree.
- One can take a probabilistic approach -- mixture of experts framework.
- The hope of “**meta-learning**” is that it can “supercharge” a mediocre learning algorithm into an excellent learning algorithm.

Model Averaging

- It is useful to distinguish between: **Bayesian model averaging** and **model combination**.

- Example: Consider a mixture of Gaussians:

$$p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k).$$

- Hence for i.i.d. data:

$$p(\mathbf{X}) = \prod_{n=1}^N \sum_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n).$$

- This is an example of model combination.
- Different data points **within the same dataset** can be generated from **different values of the latent variables** (or by different components).

Bayesian Model Averaging

- Suppose we have several different models, indexed by $h=1,\dots,H$, with prior probabilities $p(h)$.
- Example: one model can be a **mixture of Gaussians**, another one can be a **mixture of Cauchy** distributions.

- The marginal over the dataset is:

$$p(\mathbf{X}) = \sum_{h=1}^H p(\mathbf{X}|h)p(h).$$

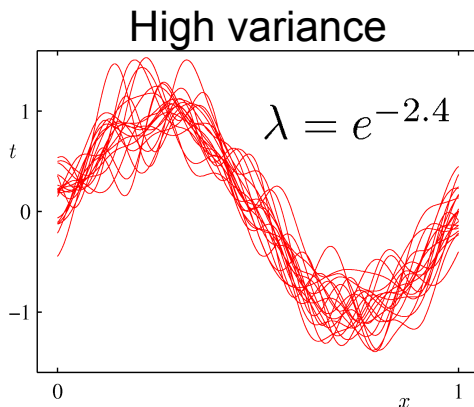
- This is an example of the **Bayesian model averaging**.
- **Interpretation**: Just one model is responsible for generating the whole dataset!
- The distribution over h reflects our uncertainty as to which model that is.
- As we observe more data, the uncertainty decreases, and the posterior $p(h|\mathbf{X})$ becomes focused on **just one model**.
- The same reasoning apply for the conditional distributions $p(t|x,\mathbf{X},\mathbf{T})$.

Committees

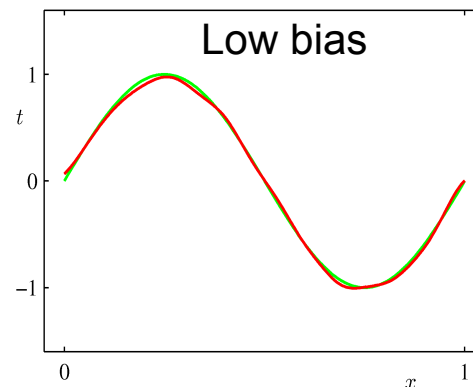
- Average the predictions of a set of individual models.
- Motivation: **Bias-variance trade-off**:
 - **bias**: difference between the model and the true function to be predicted.
 - **variance**: sensitivity of the model due to the given dataset.

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise}$$

Average predictions over all datasets differ from the optimal regression function.



Solutions for individual datasets vary around their averages -- how sensitive is the function to the particular choice of the dataset.



Intrinsic variability of the target values.

- When we average a set of low-bias models (e.g. higher-order polynomials), we obtain accurate predictions.

Bagging

- Bagging = Bootstrap aggregation.
- In practice, we only have one dataset: Need a way to introduce variability between different models.
- One idea: Generate **M bootstrap samples** from your original training set and train B separate models.
 - For regression, average predictions.
 - For regression, average class probabilities (or take the majority vote if only hard outputs available).
- The **size of each bootstrap sample is equal to the size of the original training set**, but they are drawn with replacement, so each one contains some duplicates of certain training points and leaves out other training points completely.

Variance Reduction by Averaging

- Suppose we M bootstrap datasets and train M models $y_m(\mathbf{x})$.
- The committee prediction is given by:

$$y_{COM} = \frac{1}{M} \sum_{m=1}^M y_m(\mathbf{x}).$$

- Assume that the **true function** is $h(\mathbf{x})$, hence

$$y_m(\mathbf{x}) = h(\mathbf{x}) + \epsilon_m(\mathbf{x}).$$

Expectation with respect to the distribution over the input vector \mathbf{x} .

- The average sum-of-squares error takes the form:

$$\mathbb{E}_{\mathbf{x}} [(y_m(\mathbf{x}) - h(\mathbf{x}))^2] = \mathbb{E}_{\mathbf{x}} [\epsilon_m(\mathbf{x})^2].$$

- The average error made by the models **acting individually** is therefore:

$$E_{AV} = \frac{1}{M} \sum_{m=1}^M \mathbb{E}_{\mathbf{x}} [\epsilon_m(\mathbf{x})^2].$$

Variance Reduction by Averaging

- The **committee prediction** is given by:

$$y_{COM} = \frac{1}{M} \sum_{m=1}^M y_m(\mathbf{x}).$$

- The expected error from the committee is given by:

$$\begin{aligned} E_{COM} &= \mathbb{E}_{\mathbf{x}} \left[\left(\frac{1}{M} \sum_{m=1}^M y_m(\mathbf{x}) - h(\mathbf{x}) \right)^2 \right] \\ &= \mathbb{E}_{\mathbf{x}} \left[\left(\frac{1}{M} \sum_{m=1}^M \epsilon_m(\mathbf{x}) \right)^2 \right]. \end{aligned}$$

- Assuming the errors are uncorrelated:

$$\begin{aligned} \mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})] &= 0 \\ \mathbb{E}_{\mathbf{x}}[\epsilon_m(\mathbf{x})\epsilon_k(\mathbf{x})] &= 0, \end{aligned} \quad \longrightarrow \quad E_{COM} = \frac{1}{M} E_{AV}.$$

Variance Reduction by Averaging

- Hence we have:

$$E_{COM} = \frac{1}{M} E_{AV}.$$

- This dramatic result suggests that the average error of a model **can be reduced by a factor of M** simply by averaging M versions of the models.
- Too good to be true!
- The above result depends on the key assumption that the **errors of the individual models are uncorrelated**.
- In practice, the errors will be **highly correlated** (remember, we are using bootstrap datasets).

Why do Committees Work?

- All committee learning (often called meta-learning) is based on one of two observations:
 - **Variance Reduction**: If we had completely independent training sets it always helps to average together an ensemble of learners because this reduces variance without changing bias.
 - **Bias Reduction**: For many simple models, a weighted average of those models (in some space) has much greater capacity than a single model (e.g. hyperplane classifiers, single-layer networks). Averaging models can often reduce bias substantially by increasing capacity; we can keep variance low by only fitting one member of the mixture at a time.
- Either reduces variance substantially without affecting bias (bagging), or vice versa (boosting).

Finite Bagging Can Hurt

- Bagging helps when a learning algorithm is good on average but **unstable with respect to the training set**.
- But if we bag a stable learning algorithm, we can actually **make it worse**. (For example, if we have a Bayes optimal algorithm, and we bag it, we might leave out some training samples in every bootstrap, and so the optimal algorithm will never be able to see them.)
- Bagging almost always helps with regression, but even with unstable learners it can hurt in classification. If we bag a poor and unstable classifier we can make it horrible.
- **Example**: true class = A for all inputs.
Our learner guesses class A with probability 0.4 and class B with probability 0.6 regardless of the input. (Very unstable).
It has error 0.6. But if we bag it, it will have error 1.

Boosting

- Probably one of the **most influential ideas** in machine learning in the last decade.
- In the PAC framework, boosting is a way of converting a “**weak**” learning model (behaves slightly better than chance) into a “**strong**” learning mode (behaves arbitrarily close to perfect).
- Strong theoretical result, but also lead to a very powerful and practical algorithm which is used all the time in real world machine learning.
- Basic idea, for binary classification with $t_n = \pm 1$.

$$y_{boost} = \text{sign} \left(\sum_{m=1}^M \alpha_m y_m(\mathbf{x}) \right),$$

where $y_m(x)$ are models trained with **reweighted** datasets D_m , and the weights α_m are non-negative.

AdaBoost Algorithm

- Initialize the data weights $w_n = 1/N$.
- For $m=1, \dots, M$:
 - Fit a classifier $y_m(x)$ to the training data by minimizing the weighted error function:

$$J_m = \sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n),$$

where $I(y_m(\mathbf{x}_n) \neq t_n)$ is the indicator function and equals to one when $y_m(\mathbf{x}_n) \neq t_n$ and zero otherwise.

- Evaluate:

$$\alpha_m = \ln \frac{1 - \epsilon_m}{\epsilon_m}, \quad \epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}}.$$

Weighting coefficients.

weighted measures of the error rates.

AdaBoost Algorithm

- Initialize the data weights $w_n = 1/N$.
- For $m=1, \dots, M$:
 - Fit a classifier $y_m(\mathbf{x})$ to the training data by minimizing:

$$J_m = \sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n),$$

- Evaluate:

$$\alpha_m = \ln \frac{1 - \epsilon_m}{\epsilon_m}, \quad \epsilon_m = \frac{\sum_{n=1}^N w_n^{(m)} I(y_m(\mathbf{x}_n) \neq t_n)}{\sum_{n=1}^N w_n^{(m)}}.$$

- Update the data weights:

$$w_n^{(m+1)} = w_n^{(m)} \exp(\alpha_m I(y_m(\mathbf{x}_n) \neq t_n)).$$

- Make predictions using the final model:

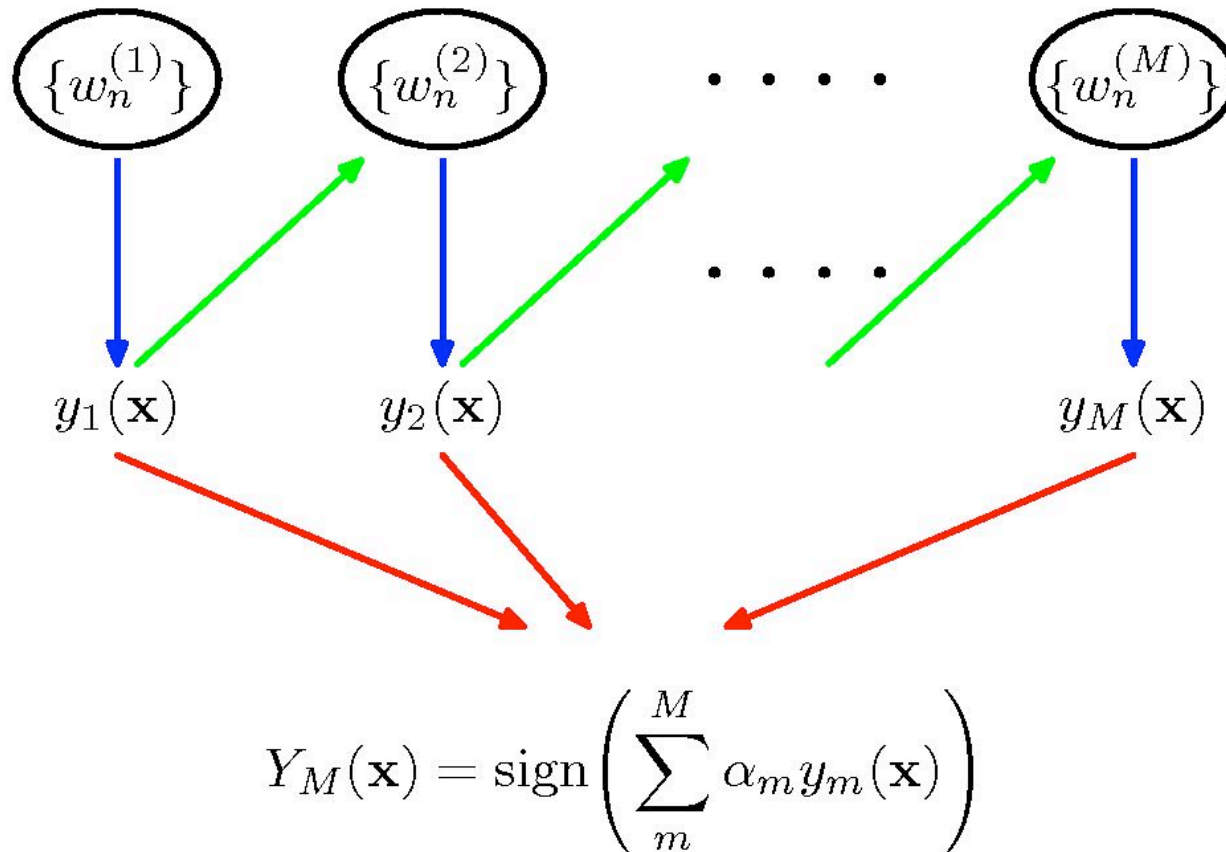
$$Y_M(\mathbf{x}) = \text{sign} \left(\sum_{m=1}^M \alpha_m y_m(\mathbf{x}) \right).$$

Some Intuitions

- The first classifier corresponds to the usual procedure for training a single classifier.
- At each round, boosting:
 - **increases the weight** on those examples the last classifier **got wrong**,
 - **decreases the weight** on those it **got right**.
- Over time, AdaBoost focuses on the examples that are **consistently difficult** and forgets about the ones that are consistently easy.
- The weight each intermediate classifier gets in the final ensemble depends on the error rate it achieved on its weighted training set at the time it was created.
- Hence the weighting coefficients α_m give greater weight to more accurate classifiers.

Some Intuitions

- Schematic illustration of AdaBoost:

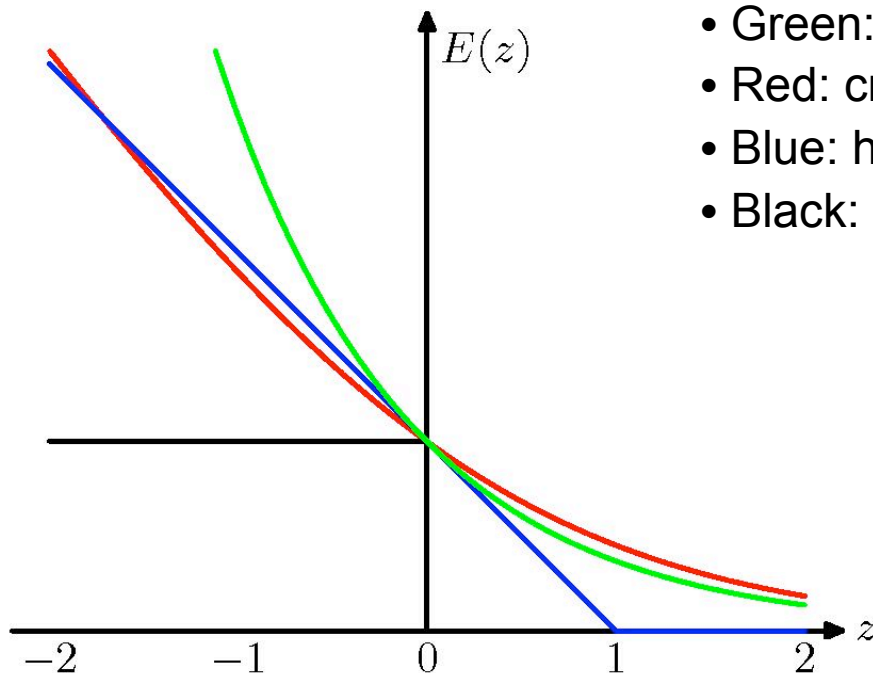


Exponential Loss

- One explanation, which helps a lot to understand how boosting really works, is that classification boosting is equivalent to **sequential minimization** of the following loss (error) function:

$$L(t, f(\mathbf{x})) = \exp(-tf(\mathbf{x})).$$

- This is called **exponential loss** and it is very similar to other kinds of loss, e.g. classification loss.



- Green: exponential
- Red: cross-entropy
- Blue: hinge loss
- Black: misclassifications error (0-1 loss)

Example

- Base learners are simple thresholds applied to one or another axis.

