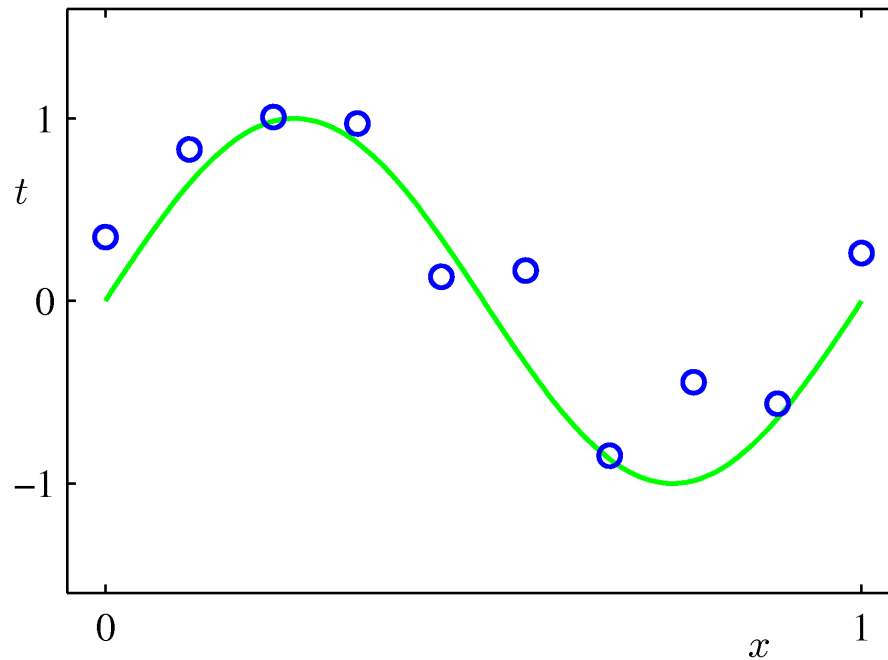


Introduction to Machine Learning

Lecture 3: Linear regression

Linear Basis Function Models (1)

- Example: Polynomial Curve Fitting



$$y(x, \mathbf{w}) = w_0 + w_1x + w_2x^2 + \dots + w_Mx^M = \sum_{j=0}^M w_jx^j$$

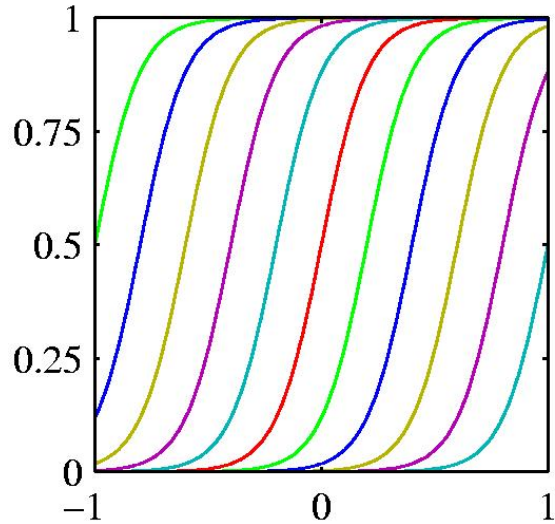
Linear Basis Function Models (2)

- Generally

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$$

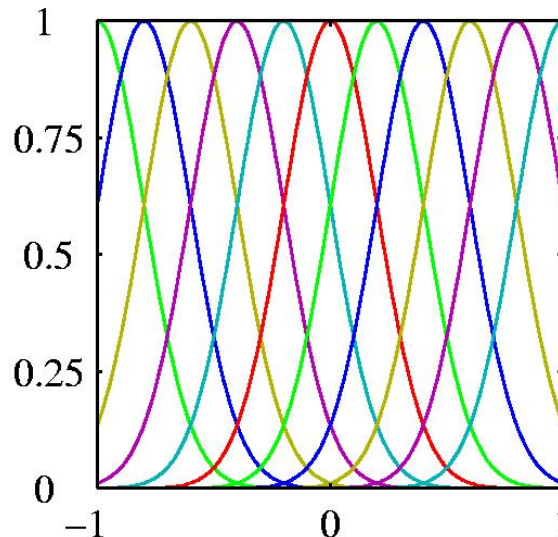
- where $\phi_j(\mathbf{x})$ are known as *basis functions*.
- Typically, $\phi_0(\mathbf{x}) = 1$, so that w_0 acts as a bias.
- In the simplest case, we use linear basis functions :
 $\phi_d(\mathbf{x}) = x_d$.

Some types of basis function in 1-D



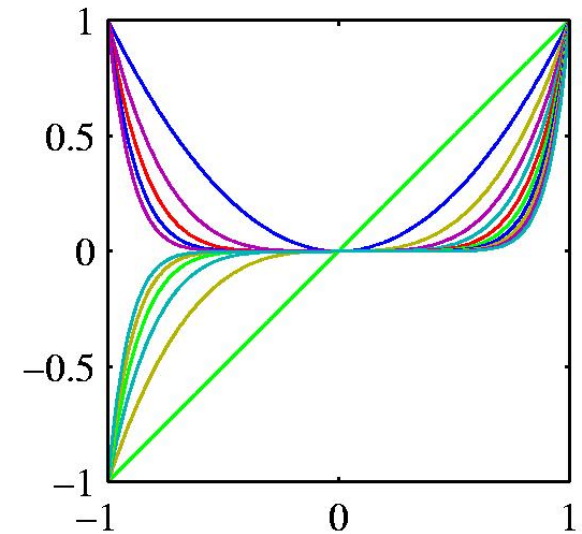
Sigmoids

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$$
$$\sigma(a) = \frac{1}{1 + \exp(-a)}.$$



Gaussians

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



Polynomials

$$\phi_j(x) = x^j.$$

Sigmoid and Gaussian basis functions can also be used in multilayer neural networks, but neural networks **learn** the parameters of the basis functions. **This is much more powerful but also much harder and messier.**

Two types of linear model that are equivalent with respect to learning

bias
↓

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + \dots = \mathbf{w}^T \mathbf{x}$$

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 \phi_1(\mathbf{x}) + w_2 \phi_2(\mathbf{x}) + \dots = \mathbf{w}^T \Phi(\mathbf{x})$$

- The first and second model has the same number of adaptive coefficients as the number of basis functions +1.
- Once we have replaced the data by the outputs of the basis functions, fitting the second model is exactly the same problem as fitting the first model (unless we use the kernel trick)
 - So its silly to clutter up the math with basis functions

Minimizing squared error

$$y = \mathbf{w}^T \mathbf{x}$$

$$error = \sum_n (t_n - \mathbf{w}^T \mathbf{x}_n)^2$$

$$\mathbf{w}^* = \boxed{(\mathbf{X}^T \mathbf{X})^{-1}} \mathbf{X}^T \boxed{\mathbf{t}}$$

↑
optimal weights

↑
inverse of the covariance matrix of input vectors

↑
the transposed design matrix has one input vector per column

←
vector of target values

Maximum Likelihood and Least Squares (1)

- Assume observations from a deterministic function with added Gaussian noise:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \quad \text{where} \quad p(\epsilon|\beta) = \mathcal{N}(\epsilon|0, \beta^{-1})$$

- or,

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}).$$

- Given observed inputs, $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ and targets $\mathbf{t} = [t_1, \dots, t_N]^T$, we obtain the likelihood function

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(t_n|\mathbf{w}^T \phi(\mathbf{x}_n), \beta^{-1}).$$

Maximum Likelihood and Least Squares (2)

- Taking the logarithm, we get

$$\begin{aligned}\ln p(\mathbf{t}|\mathbf{w}, \beta) &= \sum_{n=1}^N \ln \mathcal{N}(t_n | \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}) \\ &= \frac{N}{2} \ln \beta - \frac{N}{2} \ln(2\pi) - \beta E_D(\mathbf{w})\end{aligned}$$

- Where the sum-of-squares error is

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

Maximum Likelihood and Least Squares (3)

- Computing the gradient and setting it to zero yields

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{w}, \beta) = \beta \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\} \phi(\mathbf{x}_n)^T = \mathbf{0}.$$

- Solving for \mathbf{w} ,

$$\mathbf{w}_{\text{ML}} = \left(\Phi^T \Phi \right)^{-1} \Phi^T \mathbf{t}$$

The Moore-Penrose pseudo-inverse, Φ^\dagger .

- where

$$\Phi = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}.$$

Maximum Likelihood and Least Squares (4)

Maximizing with respect to the bias, w_0 , alone,

$$\begin{aligned}w_0 &= \bar{t} - \sum_{j=1}^{M-1} w_j \overline{\phi_j} \\&= \underbrace{\frac{1}{N} \sum_{n=1}^N t_n}_{\bar{t}} - \sum_{j=1}^{M-1} w_j \underbrace{\frac{1}{N} \sum_{n=1}^N \phi_j(\mathbf{x}_n)}_{\overline{\phi_j}}.\end{aligned}$$

We can also maximize with respect to β , giving

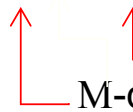
$$\frac{1}{\beta_{\text{ML}}} = \frac{1}{N} \sum_{n=1}^N \{t_n - \mathbf{w}_{\text{ML}}^T \boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

Geometry of Least Squares

Consider

$$\mathbf{y} = \Phi \mathbf{w}_{\text{ML}} = [\varphi_1, \dots, \varphi_M] \mathbf{w}_{\text{ML}}.$$

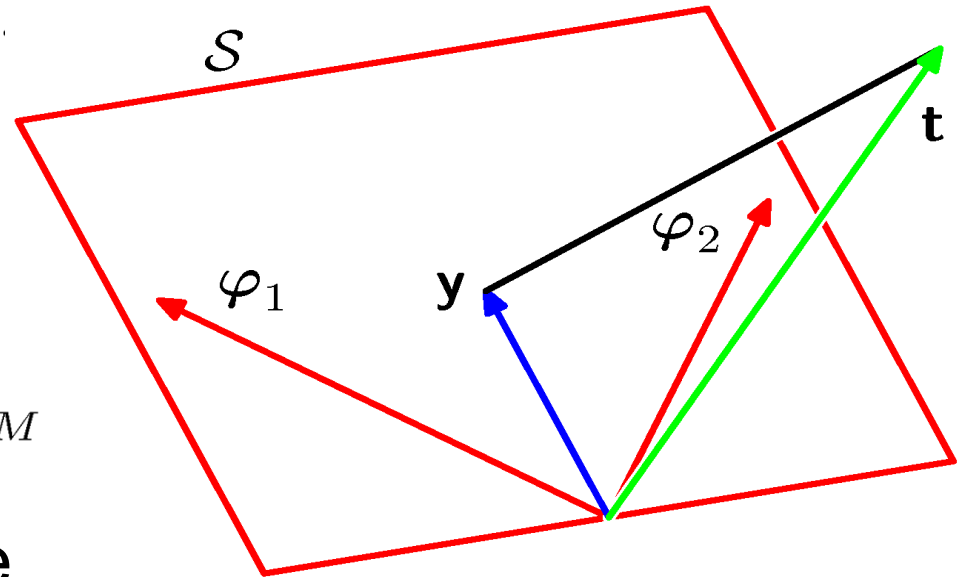
$$\mathbf{y} \in \mathcal{S} \subseteq \mathcal{T} \quad \mathbf{t} \in \mathcal{T}$$

 M-dimensional
N-dimensional

\mathcal{S} is spanned by

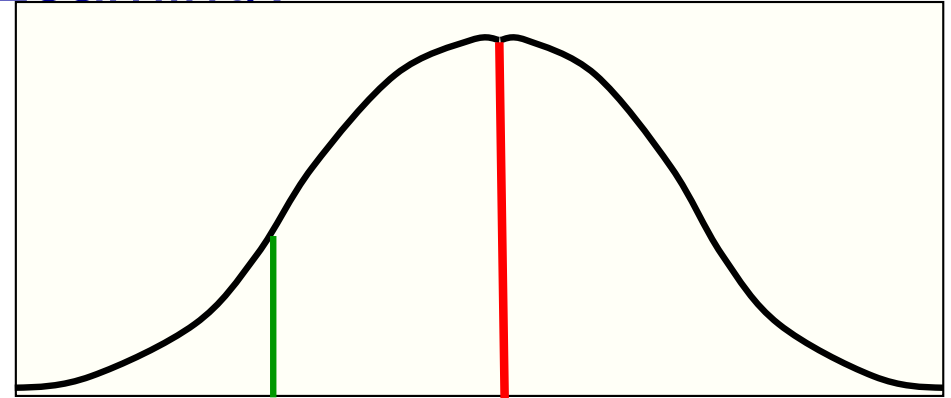
$$\varphi_1, \dots, \varphi_M$$

\mathbf{w}_{ML} minimizes the distance between \mathbf{t} and its orthogonal projection on \mathcal{S} , i.e. \mathbf{y} .



When is minimizing the squared error equivalent to Maximum Likelihood Learning?

Minimizing the squared residuals is equivalent to maximizing the log probability of the correct answer under a Gaussian centered at the model's guess.



t = correct
answer

y = model's estimate
of most probable
value

$$y_n = y(\mathbf{x}_n, \mathbf{w})$$

$$p(t_n | y_n) = p(y_n + \text{noise} = t_n | \mathbf{x}_n, \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t_n - y_n)^2}{2\sigma^2}}$$

$$-\log p(t_n | y_n) = \log \sqrt{2\pi} + \log \sigma + \frac{(t_n - y_n)^2}{2\sigma^2}$$

can be ignored
if sigma is fixed

can be ignored if
sigma is same for
every case

Multiple outputs

- If there are multiple outputs we can often treat the learning problem as a set of independent problems, one per output.
 - Not true if the output noise is correlated and changes from case to case.
- Even though they are independent problems we can save work by only multiplying the input vectors by the inverse covariance of the input components once. For output k:

$$\mathbf{w}_k^* = \boxed{(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T} \mathbf{t}_k$$

↑
does not
depend on a

Sequential Estimation (FROM lecture 2)

Contribution of the N^{th} data point, \mathbf{x}_N

$$\begin{aligned}\boldsymbol{\mu}_{\text{ML}}^{(N)} &= \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n \\&= \frac{1}{N} \mathbf{x}_N + \frac{1}{N} \sum_{n=1}^{N-1} \mathbf{x}_n \\&= \frac{1}{N} \mathbf{x}_N + \frac{N-1}{N} \boldsymbol{\mu}_{\text{ML}}^{(N-1)} \\&= \underbrace{\boldsymbol{\mu}_{\text{ML}}^{(N-1)}}_{\text{old estimate}} + \underbrace{\frac{1}{N} (\mathbf{x}_N - \boldsymbol{\mu}_{\text{ML}}^{(N-1)})}_{\text{correction given } \mathbf{x}_N}\end{aligned}$$

Correction weight

Least mean squares: An alternative approach for big datasets

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} - \eta \nabla E_{n(\tau)}$$

↑ weights after seeing training case tau+1

↑ learning rate

↑ squared error derivatives w.r.t. the weights on the training case at time tau.

- This is called “**online**” **learning**. It can be more efficient if the dataset is very redundant and it is simple to implement in hardware.
 - It is called **stochastic gradient descent** if the training cases are picked randomly.
 - Care must be taken with the learning rate to prevent divergent oscillations, and the rate must decrease with tau to get a good fit.

Regularized least squares

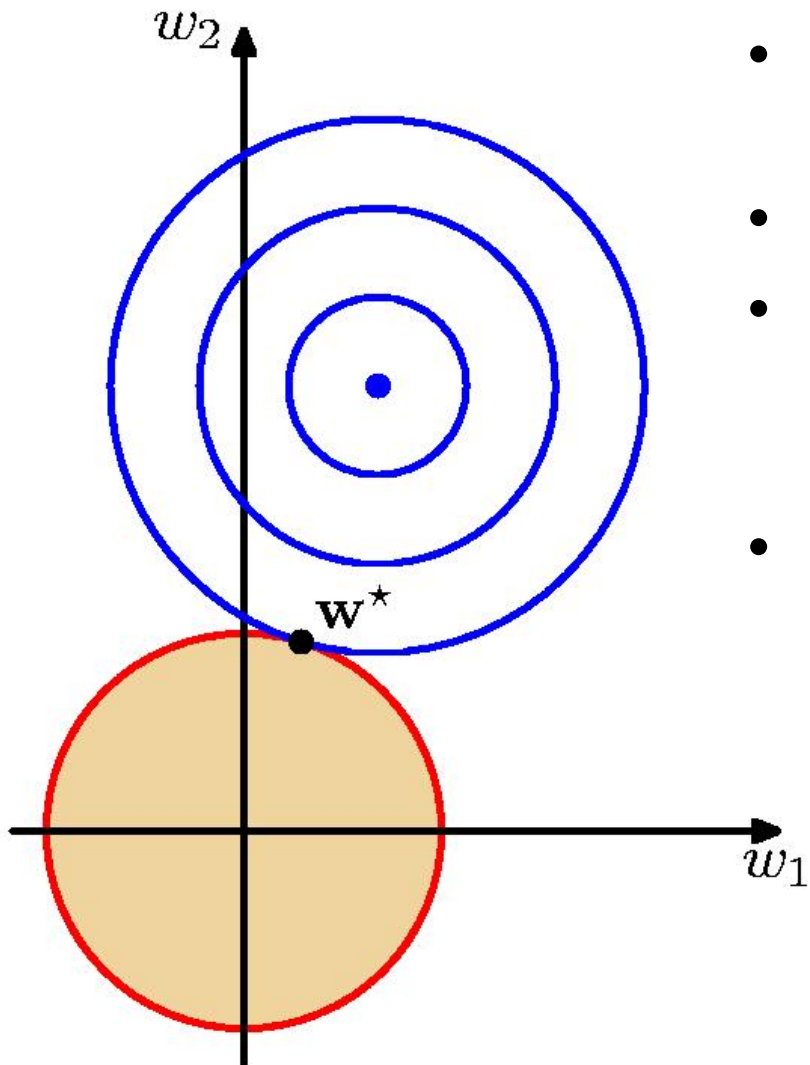
$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

The penalty on the squared weights is mathematically compatible with the squared error function, so we get a nice closed form for the optimal weights with this regularizer:

$$\mathbf{w}^* = (\lambda \mathbf{I} + \mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$$

↑
identity matrix

A picture of the effect of the regularizer



- The overall cost function is the sum of two parabolic bowls.
- The sum is also a parabolic bowl.
- The combined minimum lies on the line between the minimum of the squared error and the origin.
- The regularizer just **shrinks** the weights.

A problem with the regularizer

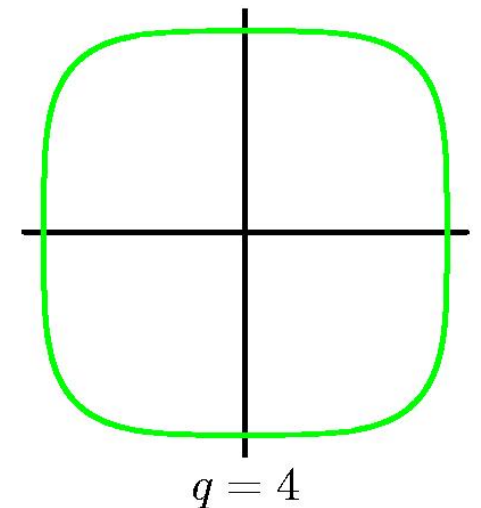
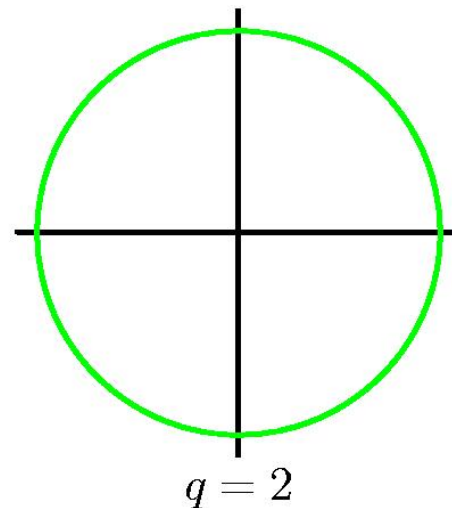
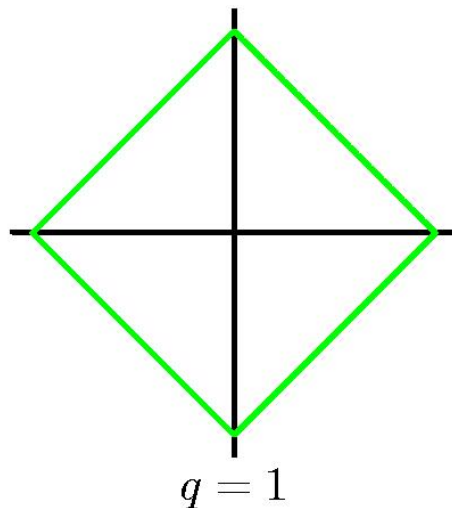
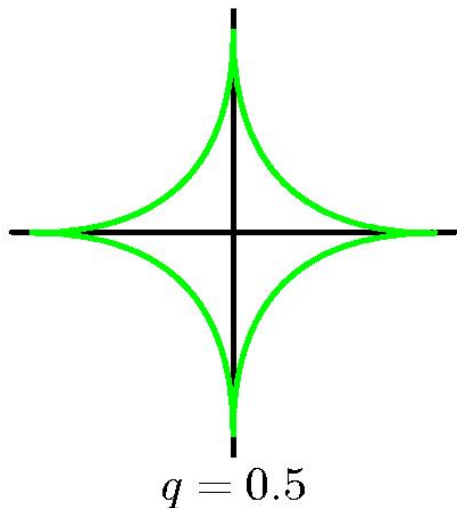
- We would like the solution we find to be independent of the units we use to measure the components of the input vector.
- If different components have different units (e.g. age and height), we have a problem.
 - If we measure age in months and height in meters, the relative values of the two weights are very different than if we use years and millimeters. So the squared penalty has very different effects.
- One way to avoid the units problem: Whiten the data so that the input components all have unit variance and no covariance. This stops the regularizer from being applied to the whitening matrix.

$$\mathbf{X}_{whitened}^T = (\mathbf{X}^T \mathbf{X})^{-\frac{1}{2}} \mathbf{X}^T$$

- But this can cause other problems when two input components are almost perfectly correlated.
- We really need a prior on the weight on each input component.

Other regularizers

- We do not need to use the squared error, provided we are willing to do more computation.
- Other powers of the weights can be used.

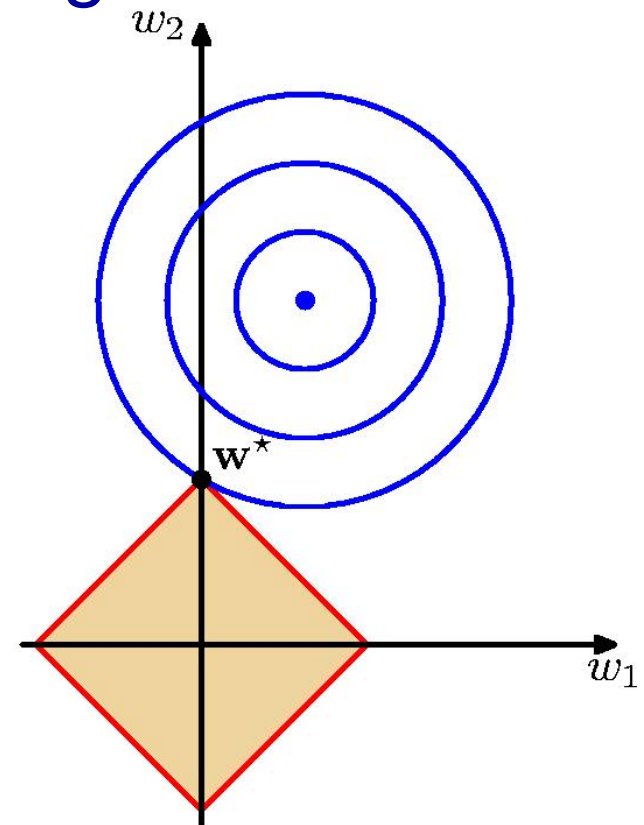
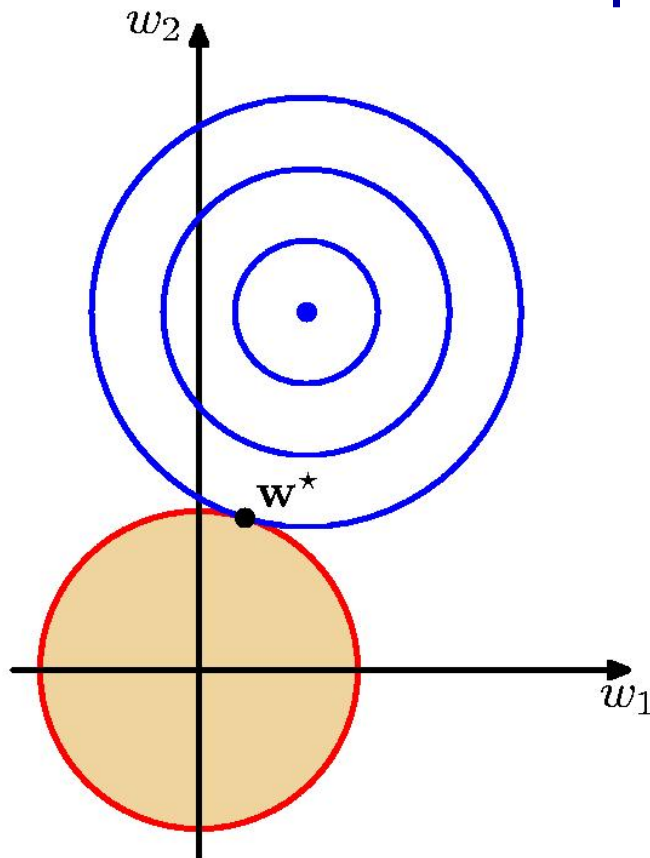


The lasso: penalizing the absolute values of the weights

$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 + \lambda \sum_i |\mathbf{w}_i|$$

- Finding the minimum requires quadratic programming but its still unique because the cost function is convex (a bowl plus an inverted pyramid)
- As lambda is increased, many of the weights go to exactly zero.
 - This is great for interpretation, and it is also pretty good for preventing overfitting.

Geometrical view of the lasso compared with a penalty on the squared weights



Notice **$w_1=0$** at the optimum

Minimizing the absolute error

$$\min_{\text{over } \mathbf{w}} \sum_n |t_n - \mathbf{w}^T \mathbf{x}_n|$$

- This minimization involves solving a linear programming problem.
- It corresponds to maximum likelihood estimation if the output noise is modeled by a Laplacian instead of a Gaussian.

$$p(t_n | y_n) = a e^{-a |t_n - y_n|}$$

$$-\log p(t_n | y_n) = -a |t_n - y_n| + \text{const}$$

The bias-variance trade-off

(a figment of the frequentists lack of imagination?)

- Imagine that the training set was drawn at random from a whole set of training sets.
- The squared loss can be decomposed into a “bias” term and a “variance” term.
 - Bias = systematic error in the model’s estimates
 - Variance = noise in the estimates caused by sampling noise in the training set.
- There is also an additional loss due to the fact that the target values are noisy.
 - We eliminate this extra, irreducible loss from the math by using the average target values (i.e. the unknown, noise-free values)

The bias-variance decomposition

model's estimate
for test case n
when trained on
dataset D

average
target
value for
test case n

The “bias” term is the squared error of the average, over all training datasets, of the estimates.

$$\left\langle \{y(\mathbf{x}_n; D) - \bar{t}_n\}^2 \right\rangle_D = \left\{ \left\langle y(\mathbf{x}_n; D) \right\rangle_D - \bar{t}_n \right\}^2 + \left\langle \{y(\mathbf{x}_n; D) - \langle y(\mathbf{x}_n; D) \rangle_D\}^2 \right\rangle_D$$

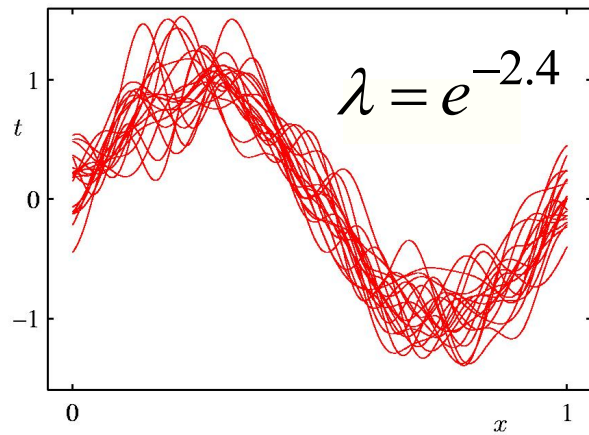
angle brackets are
physics notation
for expectation
over D

The “variance” term is the variance, over all training datasets, of the model's estimate.

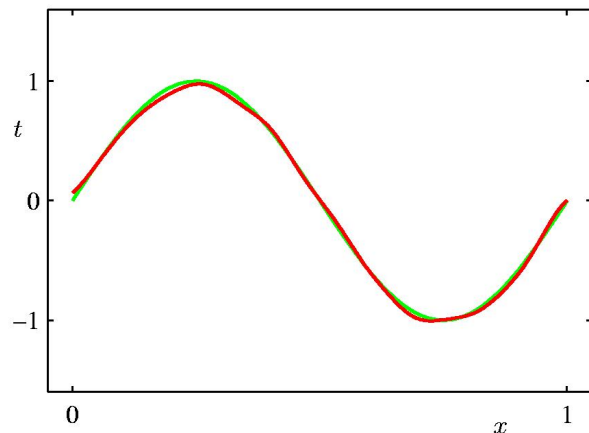
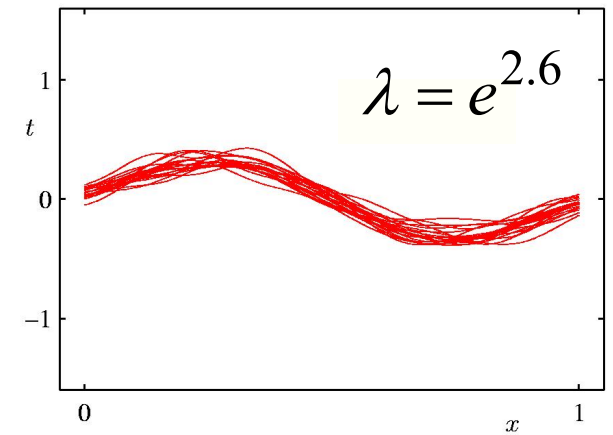
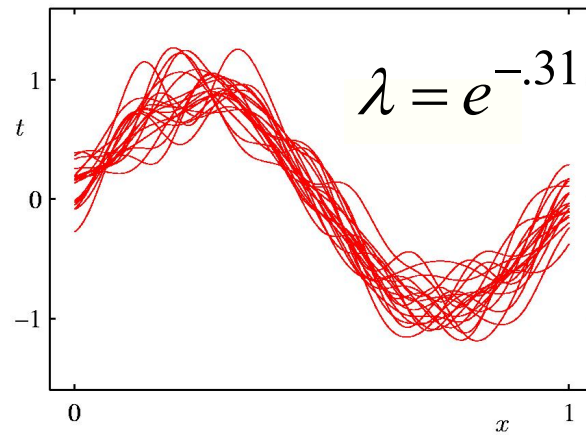
see Bishop page 149 for a derivation using a different notation

How the regularization parameter affects the bias and variance terms

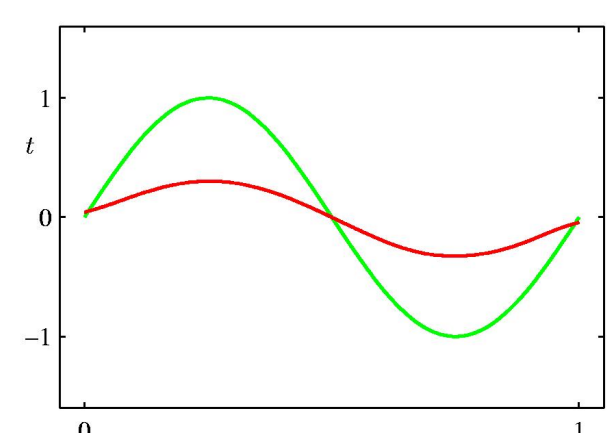
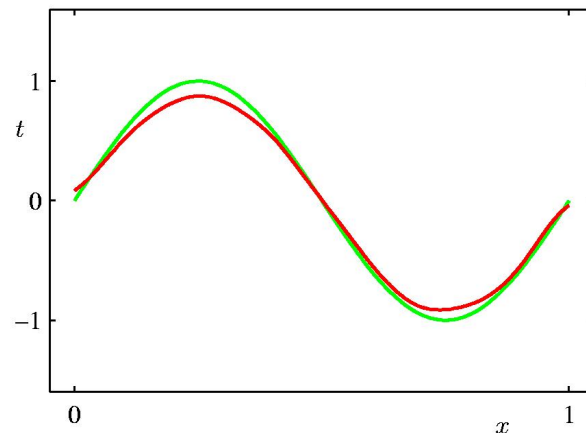
high variance



low variance

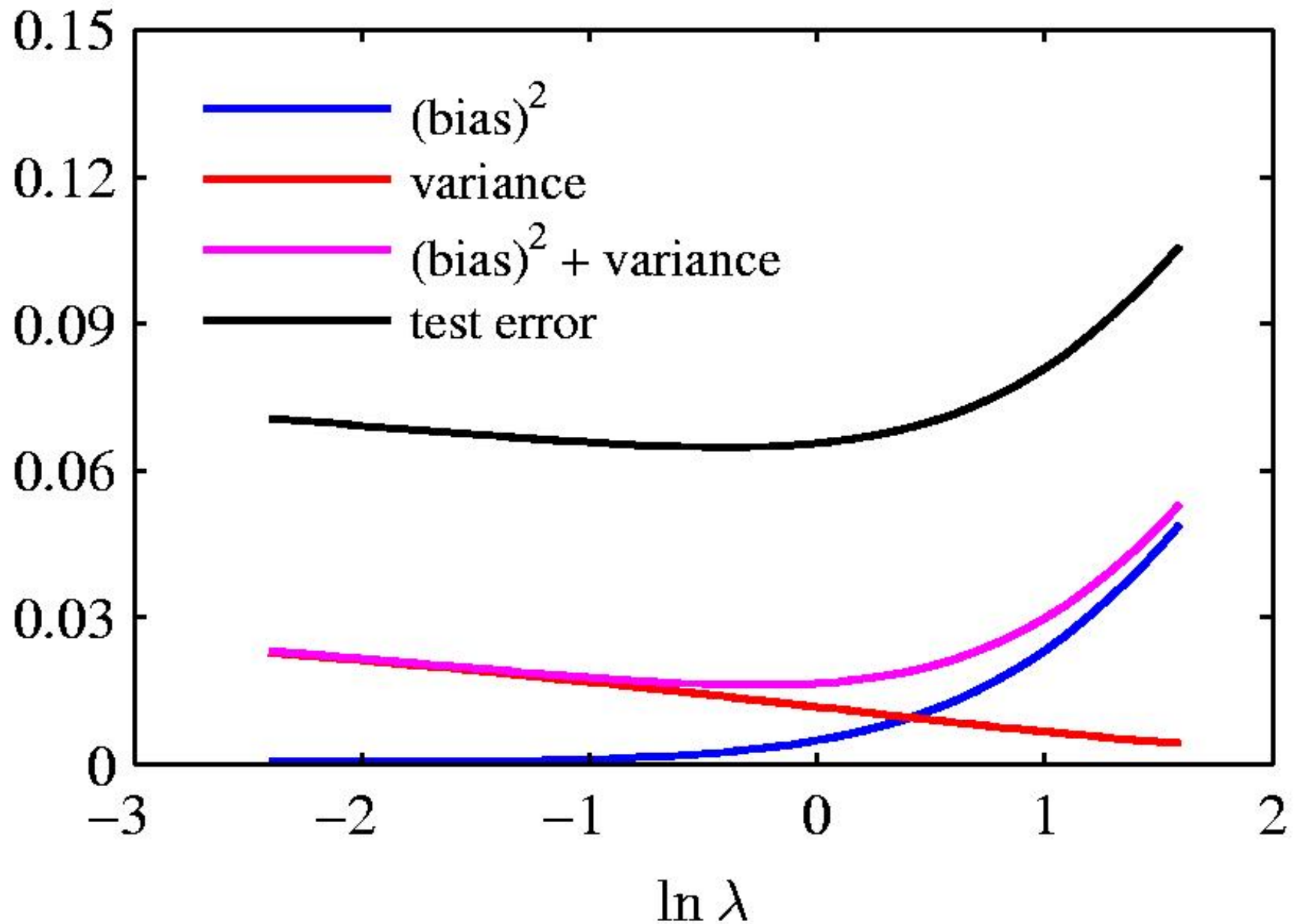


low bias



high bias

An example of the bias-variance trade-off



Beating the bias-variance trade-off

- We can reduce the variance term by averaging lots of models trained on different datasets.
 - This seems silly. If we had lots of different datasets it would be better to combine them into one big training set.
 - With more training data there will be much less variance.
- **Weird idea:** We can create different datasets by bootstrap sampling of our single training dataset.
 - This is called “bagging” and it works surprisingly well.
- But if we have enough computation its better to do the right Bayesian thing:
 - Combine the predictions of many models using the posterior probability of each parameter vector as the combination weight.

The Bayesian approach

- Consider a simple linear model that only has two parameters:

$$y(x, \mathbf{w}) = w_0 + w_1 x$$

- It is possible to display the full posterior distribution over the two-dimensional parameter space.
- The likelihood term is a Gaussian, so if we use a Gaussian prior the posterior will be Gaussian:
 - This is a conjugate prior. It means that the prior is just like having already observed some data.

$$p(\mathbf{t} | \mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^N \mathcal{N}(t_n | \mathbf{w}^T \mathbf{x}_n, \beta^{-1})$$

Gaussian
variance of output noise

← likelihood

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

conjugate prior
inverse variance of prior

$$-\ln p(\mathbf{w} | \mathbf{t}) = \frac{\beta}{2} \sum_{n=1}^N (t_n - \mathbf{w}^T \mathbf{x}_n)^2 + \frac{\alpha}{2} \mathbf{w}^T \mathbf{w} + \text{const}$$

The Bayesian interpretation of the regularization parameter:

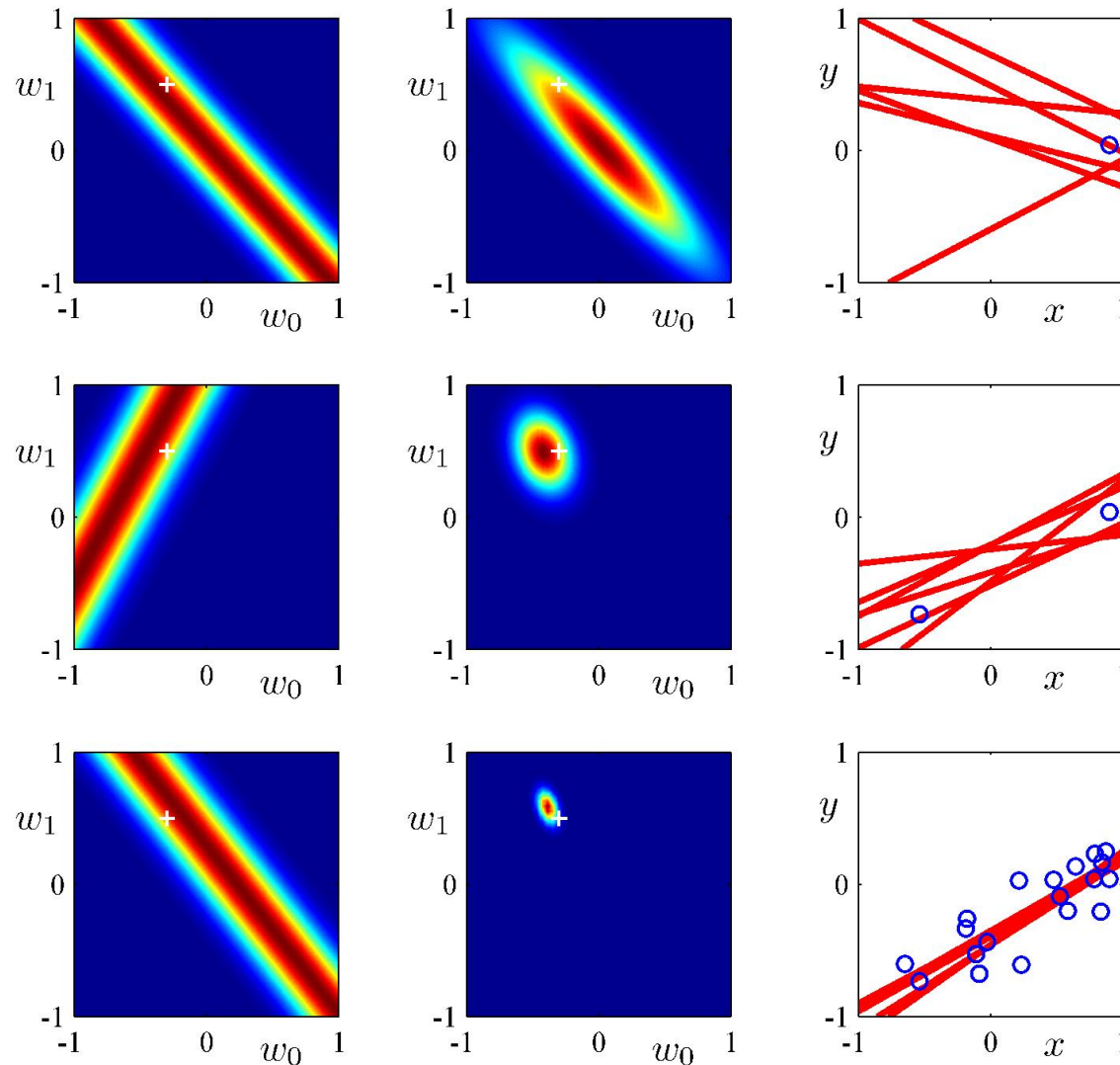
$$\lambda = \frac{\alpha}{\beta}$$

likelihood

prior/posterior

data space

With no data we sample lines from the prior.



With 20 data points, the prior has little effect