CSC2515 Winter 2015
Introduction to Machine Learning

Lecture 2: Linear regression

All lecture slides will be available as .pdf on the course website:
http://www.cs.toronto.edu/~urtasun/courses/CSC2515/CSC2515_Winter15.html

Many of the figures are provided by Chris Bishop
from his textbook: ”Pattern Recognition and Machine Learning”

Admin Details

- Permanent tutorial time/place:
  - Tuesday 11-12, GB248

- Do I have the appropriate background?
  - Linear algebra: vector/matrix manipulations, properties
  - Calculus: partial derivatives
  - Probability: common distributions; Bayes Rule
  - Statistics: mean/median/mode; maximum likelihood

- Am I interested in this course?
  - Many people in the waiting list

- Talk about Projects

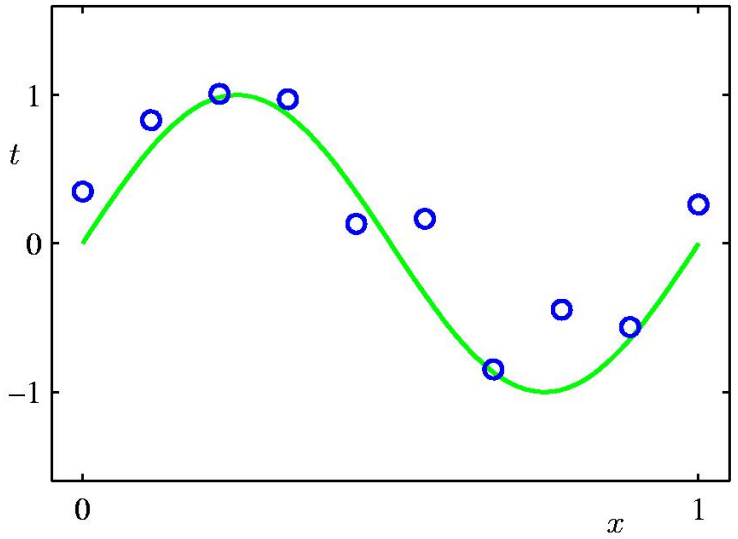
Outline

- Linear regression problem
  - continuous outputs
  - simple model
- Introduce key concepts:
  - loss functions
  - generalization
  - optimization
  - model complexity
  - regularization

A simple example: 1-D regression

The green curve is the true function (which is not a polynomial) – not known

from Bishop



The data points are uniform in x but have noise in t.

$$t(x) = f(x) + \varepsilon$$

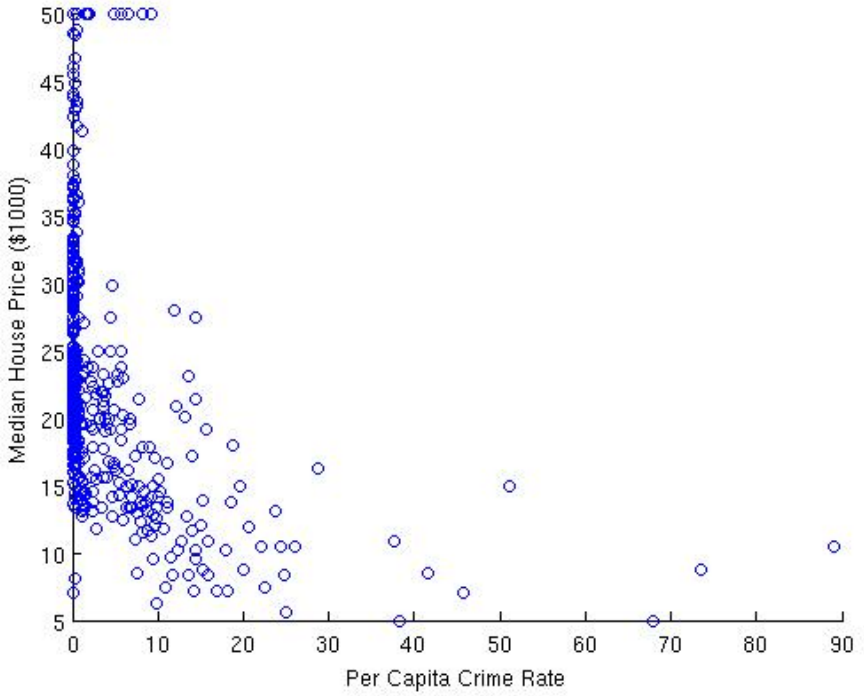
Aim: fit a curve to these points

Key questions:

- How do we parametrize the model (the curve)?
- What loss (objective) function should we use to judge fit?
- How do we optimize fit to unseen test data (generaliz., prediction)?

Example: Boston Housing data

- Estimate median house price in a neighborhood based on neighborhood statistics
- Look at first (of 13) attributes: per capita crime rate
- Use this to predict house prices in other neighborhoods



Represent the Data

Data described as pairs $D = ((x^{(1)},t^{(1)}), (x^{(2)},t^{(2)}),\ldots, (x^{(N)},t^{(N)}))$

- x is the input feature (per capita crime rate)
- t is the target output (median house price)

- Here t is continuous, so this is a regression problem

Could take first 300 examples as training set, remaining 206 as test set

- Use the training examples to construct hypothesis, or function approximator, that maps x to predicted y
- Evaluate hypothesis on test set

Noise

A simple model typically does not exactly fit the data – lack of fit can be considered noise

Sources of noise

- Imprecision in data attributes (input noise)
- Errors in data targets (mislabeling)
- Additional attributes not taken into account by data attributes, affect target values (latent variables)
- Model may be too simple to account for data targets

Least-squares Regression

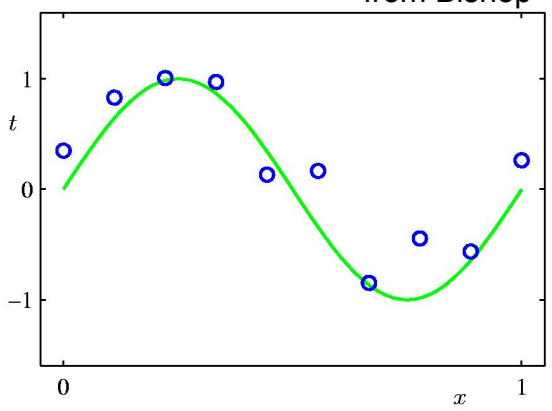
- Standard loss/cost/objective function measures the squared error in the prediction of t(x) from x.

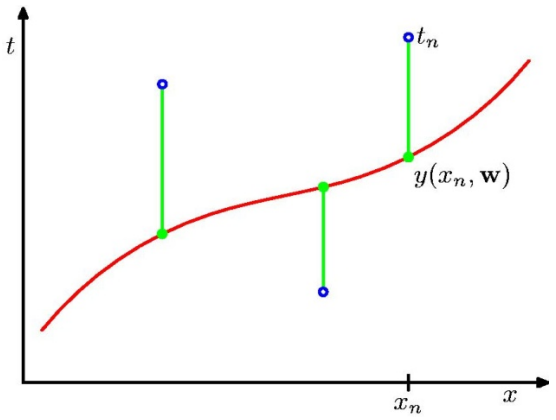
$$y(x) = w_0 + w\ x$$

$$J(\mathbf{w}) = \sum_{n=1}^{N} [t^{(n)} - y(x^{(n)})]^2$$

- The loss for the red hypothesis is the sum of the squared vertical errors.

from Bishop





Optimizing the Objective

- One straightforward method: initialize w randomly, repeatedly update based on gradient descent in J

$$w \leftarrow w - \lambda \frac{\partial J}{\partial w}$$

- Here λ is the learning rate

- For a single training case, this gives the LMS update rule:

$$w \leftarrow w + 2\lambda(t^{(n)} - y(x^{(n)}))x^{(n)}$$

- Note: as error approaches zero, so does update
- What about w_0?

Optimizing Across Training Set

Two ways to generalize this for all examples in training set:

1. Stochastic/online updates – update the parameters for each training case in turn, according to its own gradients

2. Batch updates: sum or average updates across every example i, then change the parameter values

$$w \leftarrow w + 2\lambda \sum_{n=1}^{N} (t^{(n)} - y(x^{(n)}))x^{(n)}$$

   - Underlying assumption: sample is independent and identically distributed (i.i.d.)

Non-iterative Least-squares Regression

An alternative optimization approach is non-iterative: take derivatives, set to zero, and solve for parameters.

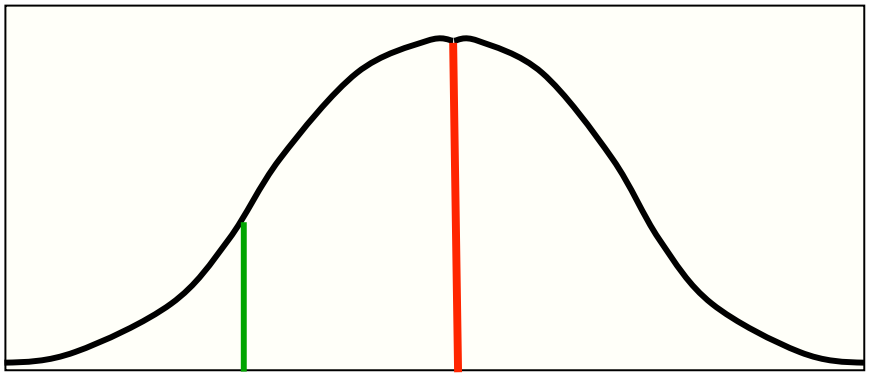
$$\frac{dJ(\mathbf{w})}{dw_0} = -2\sum_{n=1}^{N}[t^{(n)} - (w_0 + w\, x^{(n)})] = 0$$

$$w_0 = (\sum_{n=1}^{N} t^{(n)} - w\, x^{(n)}) / N = \bar{t} - w\bar{x}$$

$$w = \frac{\sum_n (t^{(n)} - \bar{t})(x^{(n)} - \bar{x})}{\sum_n (x^{(n)} - \bar{x})^2}$$

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 = the correct answer

y = model's estimate of most probable value

$$y^{(n)} = y(\mathbf{x}^{(n)}, \mathbf{w})$$

$$p(t^{(n)} \mid y^{(n)}) = p(y^{(n)} + noise = t^{(n)} \mid \mathbf{x}^{(n)}, \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(t^{(n)} - y^{(n)})^2}{2\sigma^2}}$$

$$-\log p(t^{(n)} \mid y^{(n)}) = \log\sqrt{2\pi} + \log\sigma + \frac{(t^{(n)} - y^{(n)})^2}{2\sigma^2}$$

Loss function

can be ignored if sigma is fixed

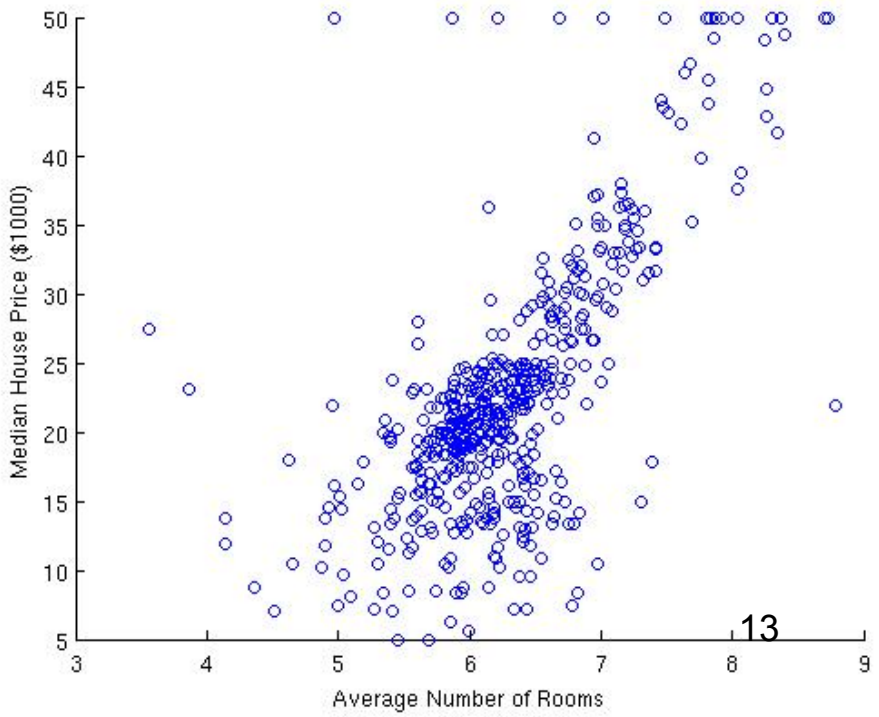
can be ignored if sigma is same for every case

Multi-dimensional inputs

One method of extending the model is to consider other input dimensions

$$y(x) = w_0 + w_1 x_1 + w_2 x_2$$

In the Boston housing example, we can look at the number of rooms input feature



We can use gradient descent to solve for each coefficient, or use linear algebra – solve system of equations

Linear Regression

- Imagine now want to predict the median house price from these multi-dimensional observations

- Each house is a data point *n*, with observations indexed by *j*:

$$\mathbf{x}^{(n)} = (x_1^{(n)}, \ldots, x_d^{(n)})$$

- Simple predictor is analogue of linear classifier, producing real-valued *y* for input **x** with parameters **w** (effectively fixing $x_0 = 1$):

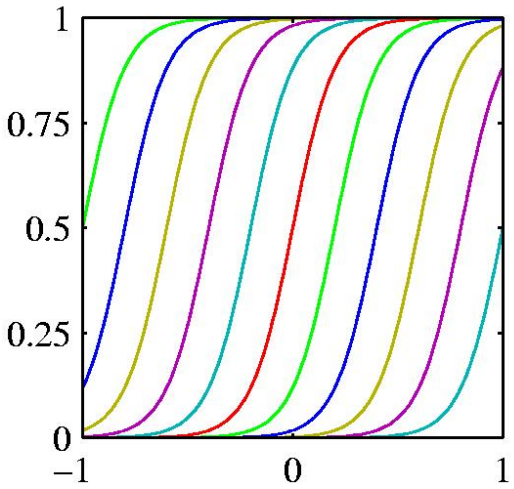
$$y = w_0 + \sum_{j=1}^{d} w_j x_j = \mathbf{w}^T \mathbf{x}$$

Why is the last equation correct?

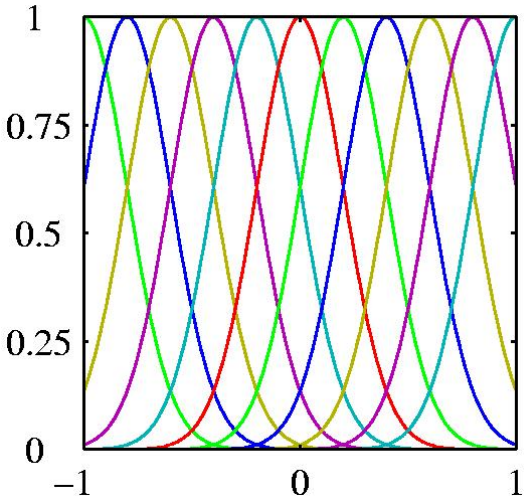
Linear models

- It is mathematically easy to fit linear models to data.
  - We can learn a lot about model-fitting in this relatively simple case.
- There are many ways to make linear models more powerful while retaining their nice mathematical properties:
  - By using non-linear, non-adaptive basis functions, we can get generalized linear models that learn non-linear mappings from input to output but are linear in their parameters – only the linear part of the model learns.
  - By using kernel methods we can handle expansions of the raw data that use a huge number of non-linear, non-adaptive basis functions.
- But linear methods will not solve most AI problems.
  - They have fundamental limitations.

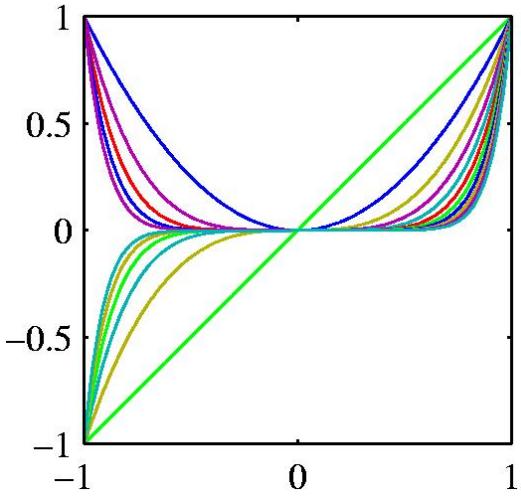
Some types of basis functions in 1-D



Sigmoids



Gaussians



Polynomials

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 much messier.

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

bias

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

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

- The first model has the same number of adaptive coefficients as the dimensionality of the data +1.
- The 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 we'll just focus on the first model

Fitting a polynomial

- Now we use one of these basis functions: an M^{th} order polynomial function

$$y(x, \mathbf{w}) = w_0 + \sum_{j=1}^{M} w_j x^j$$

- We can use the same approaches to optimize the values of the weights on each coefficient: analytic, or iterative

Minimizing squared error

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

$$error = \sum_i (t^{(n)} - \mathbf{w}^T \mathbf{x}^{(n)})^2$$

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

vector of target values

optimal weights

inverse of the covariance matrix of the input vectors

The transposed design matrix has one input vector per row

Online Least mean squares: An alternative approach for really big datasets

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \lambda \nabla J^{(n,\tau)}$$

weights after seeing training case tau+1

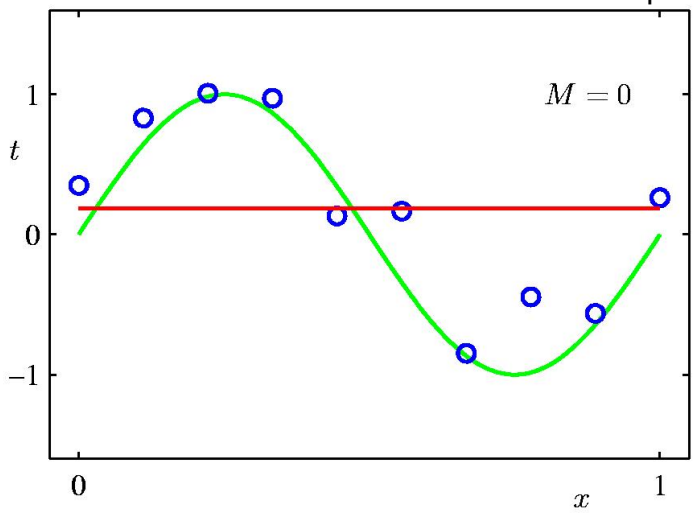
learning rate

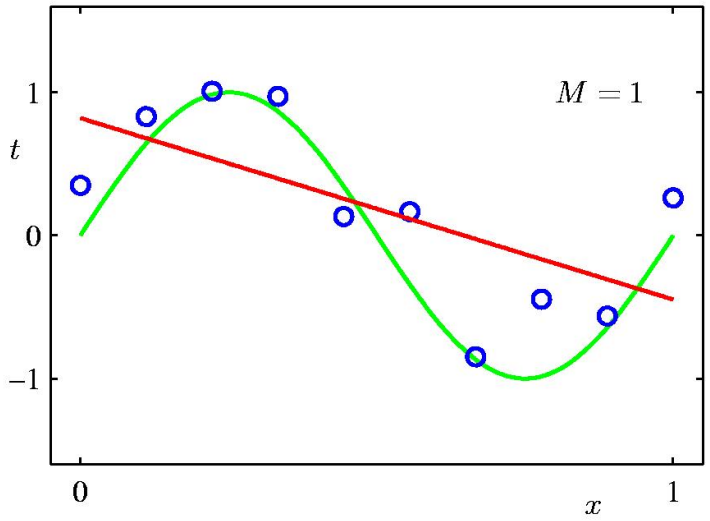
vector of derivatives of the squared error w.r.t. the weights on the training case presented at time tau.

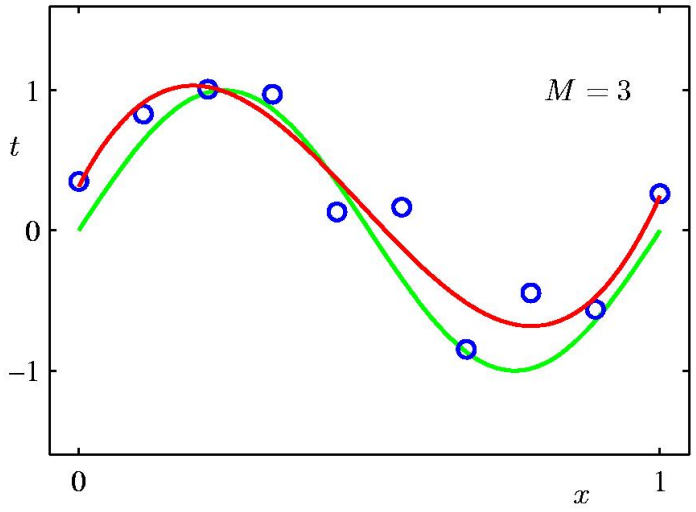
- This is called "online" learning. It can be more efficient if the dataset is very redundant and it is simple to implement in hardware.
  - It is also called stochastic gradient descent if the training cases are picked at random.
  - Care must be taken with the learning rate to prevent divergent oscillations,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

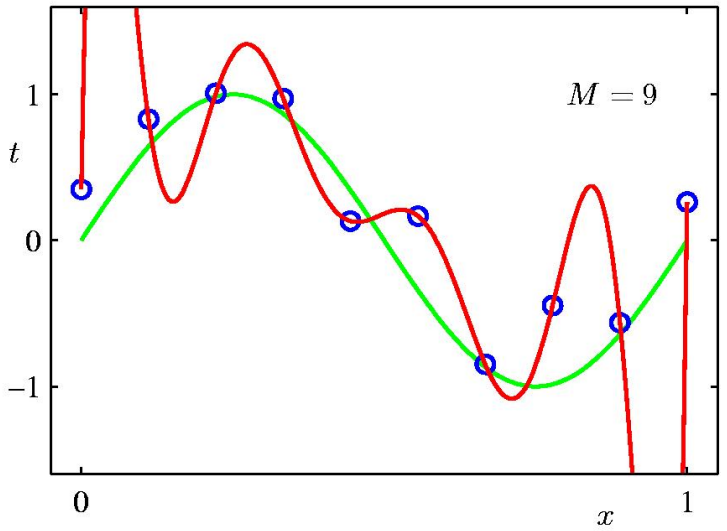
Some fits to the data: which is best?

from Bishop









1-D regression illustrates key concepts

Data fits – is linear model best (model selection)?

- Simplest models do not capture all the important variations (signal) in the data: underfit
- More complex model may overfit the training data (fit not only the signal but also the noise in the data), especially if not enough data to constrain model

One method of assessing fit: test generalization = model's ability to predict the held out data

Optimization is essential: stochastic and batch iterative approaches; analytic when available

Regularized least squares

$$\tilde{J}(\mathbf{w}) = \sum_{n=1}^{N} \{y(\mathbf{x}^{(n)}, \mathbf{w}) - t^{(n)}\}^2 + \alpha \mathbf{w}^T \mathbf{w}$$

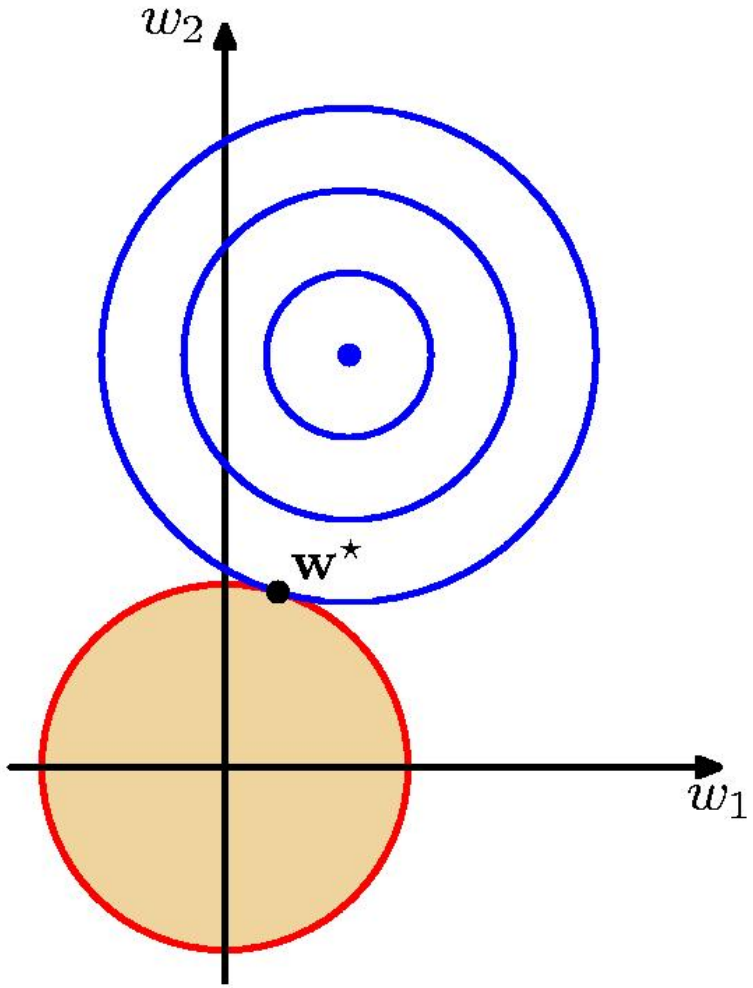
The penalty on the squared weights (aka ridge regression, weight decay) is mathematically compatible with the squared error function, so we get a nice closed form for the optimal weights with this regularizer:

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

identity matrix

Probabilistic Interpretation of the Regularizer

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 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 → squared penalty has very different effects

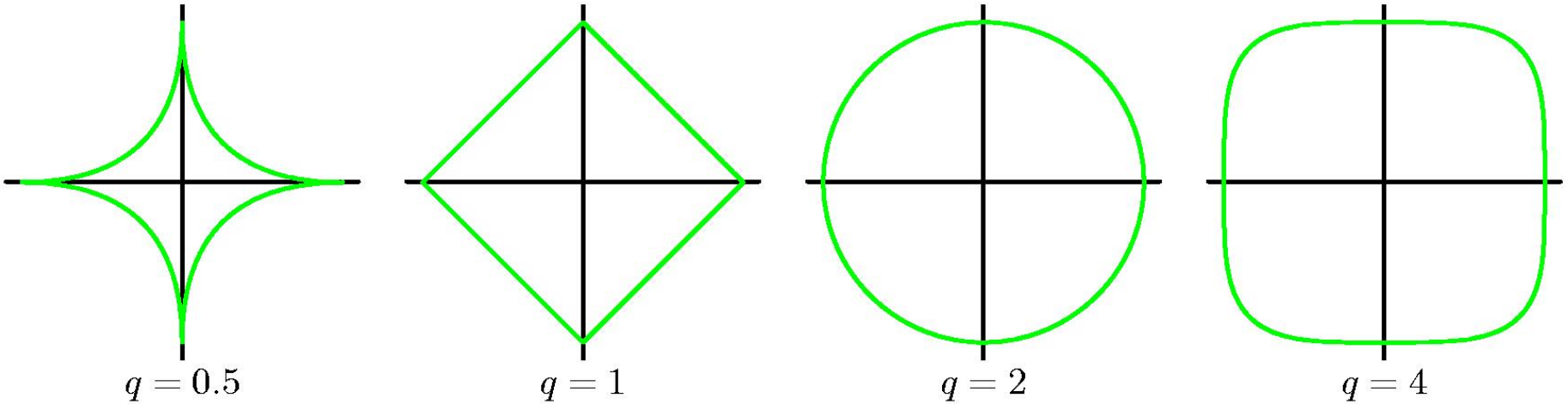
- One way to avoid the units problem: Whiten the data so that the input components all have unit variance and no covariance. This stops the regularizer from being applied to the whitening matrix.

$$\mathbf{X}^T_{whitened} = (\mathbf{X}^T \mathbf{X})^{-1/2} \mathbf{X}^T$$

  - 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.



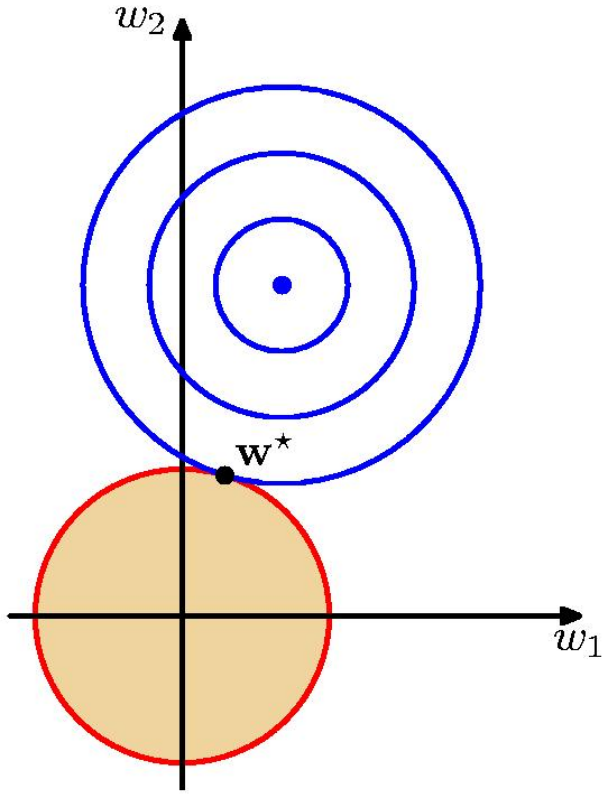
$$\tilde{J}(\mathbf{w}) = \sum_{n=1}^{N} \{ y(\mathbf{x}^{(n)}, \mathbf{w}) - t^{(n)} \}^2 + \alpha \| \mathbf{w} \|_q^q$$

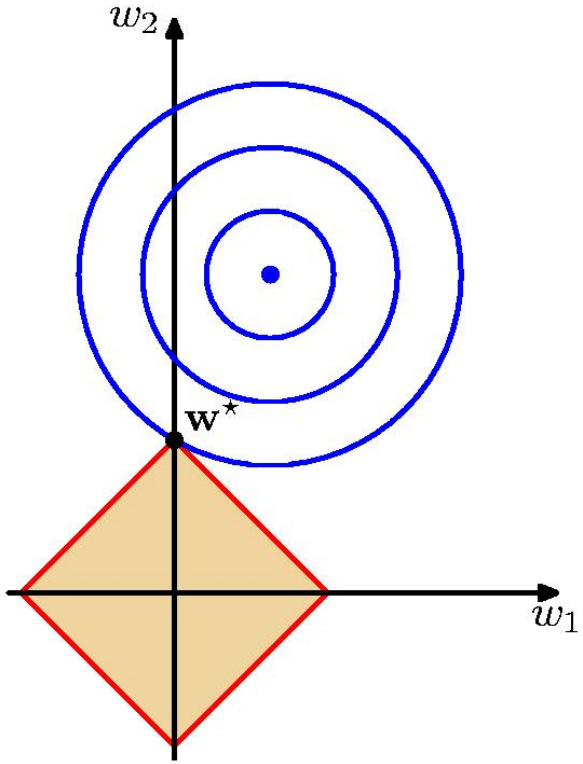
The lasso: penalizing the absolute values of the weights

$$\tilde{\mathcal{J}}(\mathbf{w}) = \sum_{n=1}^{N} \{\mathcal{y}(\mathbf{x}^{(n)}, \mathbf{w}) - t^{(n)}\}^2 + \alpha \sum_{j=1}^{d} |w_j|$$

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

A geometrical view of the lasso compared with a penalty on the squared weights





Notice that w_1=0 at the optimum

Minimizing the absolute error

$$\min_{over\ \mathbf{w}} \quad \sum_n | t^{(n)} - \mathbf{w}^T \mathbf{x}^{(n)} |$$

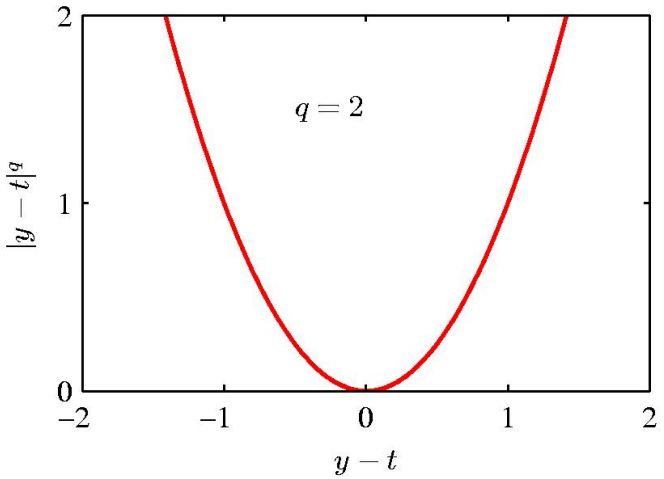
- This minimization involves solving a linear programming problem.
- It corresponds to maximum likelihood estimation if the output noise is modeled by a Laplacian instead of a Gaussian.

$$p(t^{(n)} | y^{(n)}) = a\, e^{-a|t^{(n)} - y^{(n)}|}$$

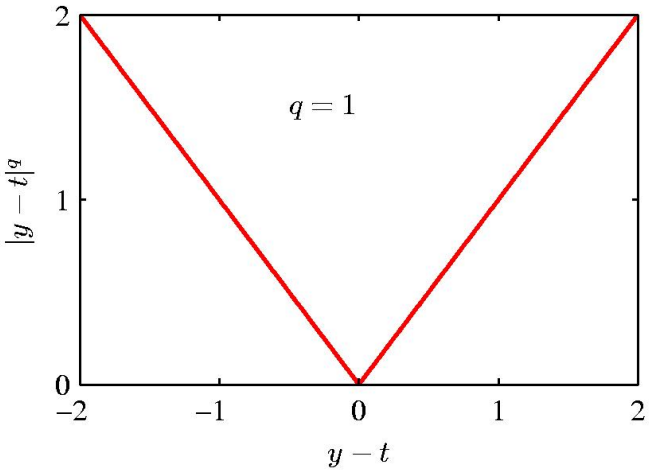
$$-\log p(t^{(n)} | y^{(n)}) = -a | t^{(n)} - y^{(n)} | + const$$

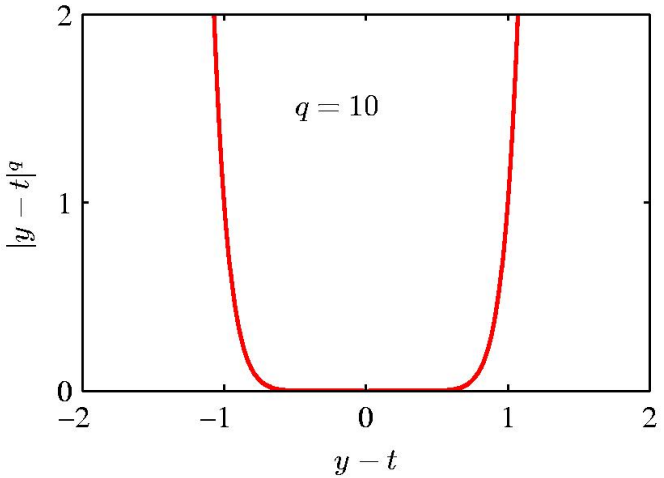
One dimensional cross-sections of loss functions with different powers

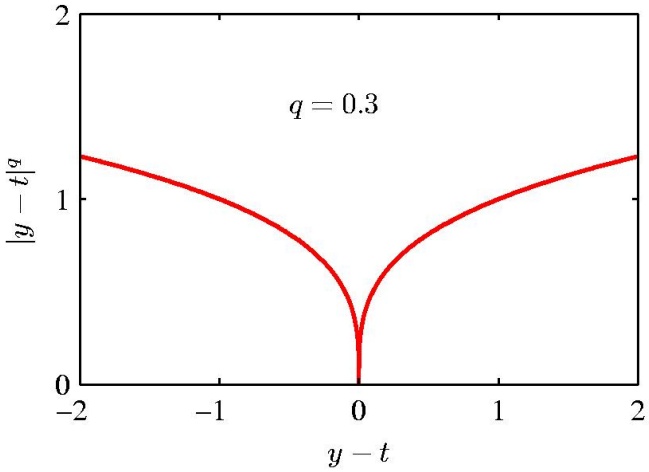
Negative log of Gaussian



Negative log of Laplacian







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.

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

angle brackets are notation for expectation over D

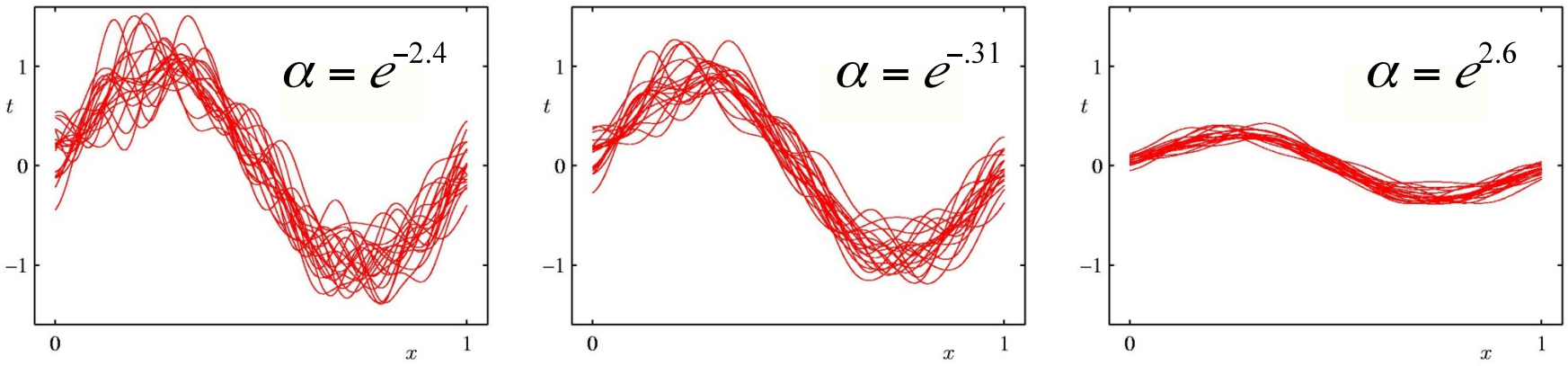
The "variance" term is the variance, over all training datasets, of the model's estimate.

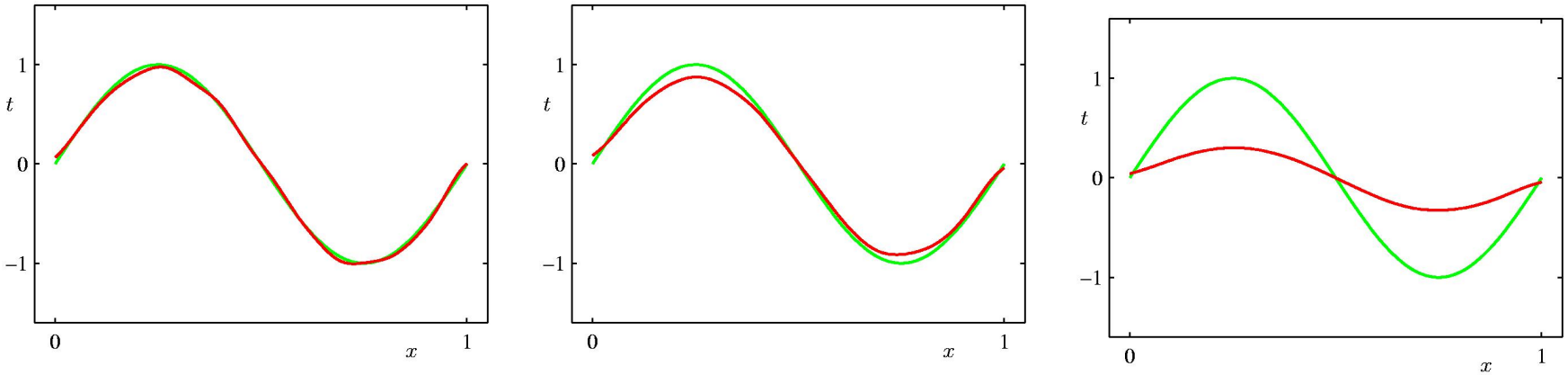
a derivation using a different notation (Bishop, 149)

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