COMP 6321
Machine Learning

Instructor: Adam Krzyżak
Email: krzyzak@cs.concordia.ca

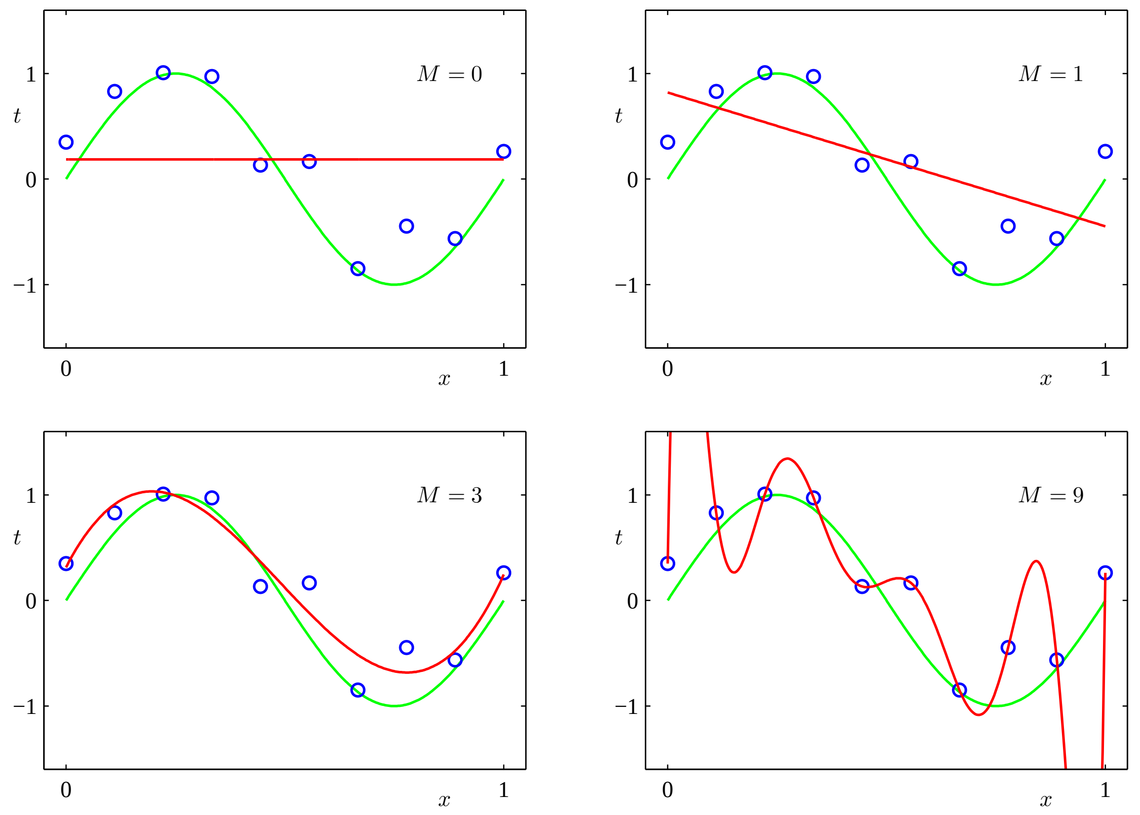
Lecture 2: More on linear methods for regression

- Overfitting and bias-variance trade-off
- Linear basis functions models
- Sequential (on-line, incremental) learning
- Why least-squares? A probabilistic analysis
- L2 and L1 regularization for linear estimators
- Bayesian learning and regularization

Recall: Linear and polynomial regression

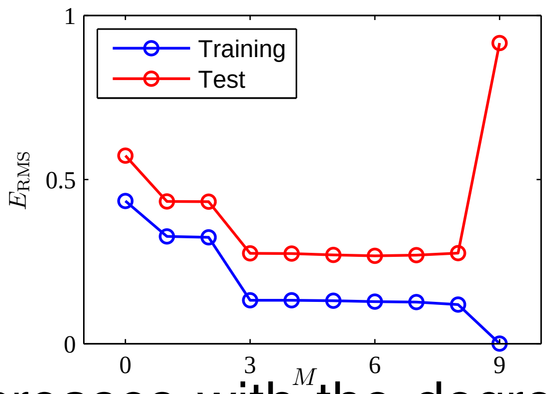
- Our first assumption was that it is good to minimize sum- (or mean-) squared error
- Algorithms that minimize this function are called *least-squares*
- Our second assumption was the linear form of the hypothesis class
- The terms were powers of the input variables (and possibly cross-terms of these powers)

Recall: Overfitting



- The higher the degree of the polynomial M, the more degrees of freedom, and the more capacity to “overfit” the training data
- Typical overfitting means that error on the training data is very low, but error on new instances is high

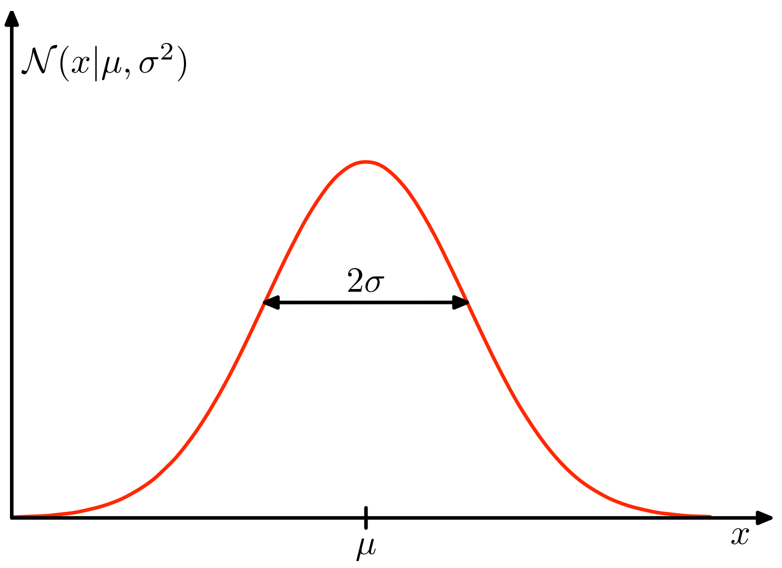
Recall: Typical overfitting plot



- The training error decreases with the degree of the polynomial M, i.e. *the complexity of the hypothesis*. For larger M the polynomials become increasingly tuned to the random noise in the target values
- The testing error, measured on independent data, decreases at first, then starts increasing
- Cross-validation helps us:
  - Find a good hypothesis class (M in our case), using a *validation set of data*
  - Report unbiased results, using a *test set*, untouched during either parameter training or validation

The anatomy of the error

- Suppose we have examples $\langle \mathbf{x}, y \rangle$ where $y = f(\mathbf{x}) + \epsilon$ and ϵ is Gaussian noise with zero mean and standard deviation σ
- Reminder: normal (Gaussian) distribution



(see Bishop Ch. 2 for review)

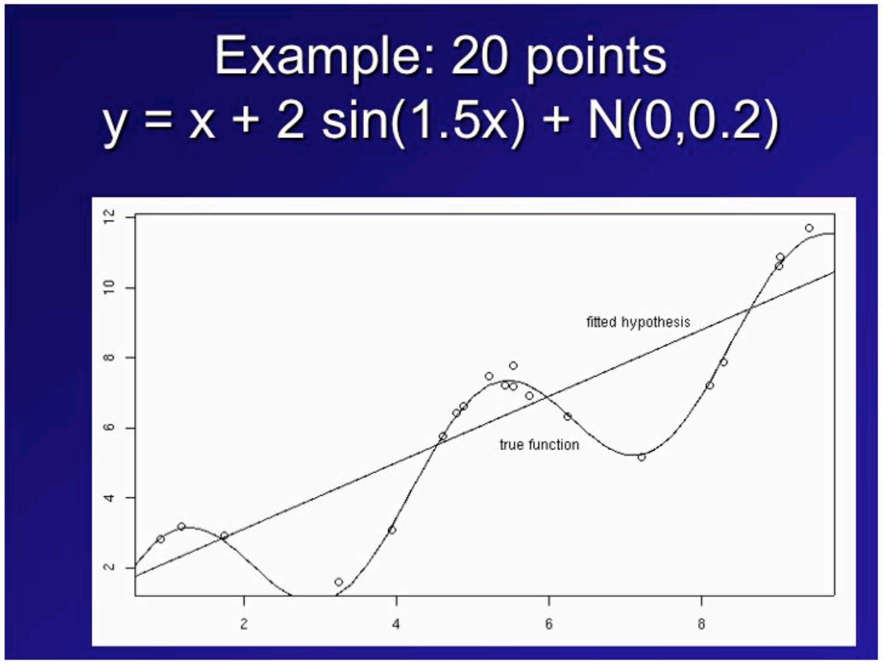
The anatomy of the error: Linear regression

- In linear regression, given a set of examples $\langle \mathbf{x}_i, y_i \rangle_{i=1\dots m}$, we fit a linear hypothesis $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$, such as to minimize sum-squared error over the training data:

$$\sum_{i=1}^{m}(y_i - h(\mathbf{x}_i))^2$$

- Because of the hypothesis class that we chose (linear hypotheses) for some target functions f we will have a *systematic prediction error*
- Even if f were truly linear, depending on the data set we have, the parameters $\mathbf{w}$ that we find may be different; this *variability* due to the specific data set on hand is a different source of error

An example (Tom Dietterich)

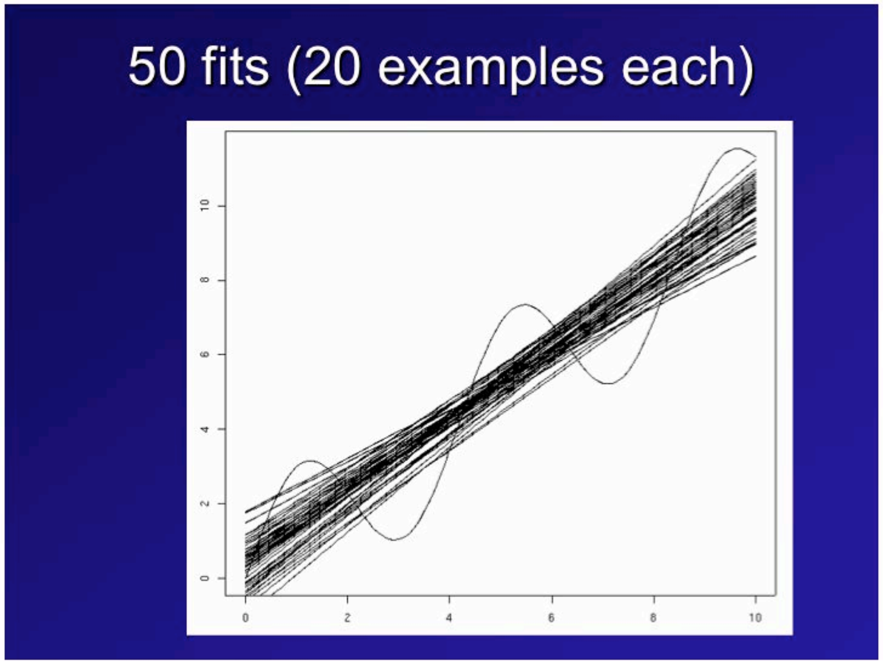


Example: 20 points

$y = x + 2\sin(1.5x) + N(0,0.2)$

- The sine is the true function
- The circles are data points (x drawn uniformly randomly, y given by the formula)
- The straight line is the linear regression fit (see lecture 1)

Example continued



With different sets of 20 points, we get different lines

Bias-variance analysis

- Given a new data point $\mathbf{x}$, what is the *expected prediction error*?
- Assume that the data points are drawn *independently and identically distributed (i.i.d.)* from a unique underlying probability distribution $P(\langle \mathbf{x}, y \rangle) = P(\mathbf{x})P(y|\mathbf{x})$
- The goal of the analysis is to compute, for an arbitrary given point $\mathbf{x}$,

$$E_P \left[(y - h(\mathbf{x}))^2 | \mathbf{x} \right]$$

  where y is the value of $\mathbf{x}$ in a data set, and the expectation is over all training sets of a given size, drawn according to P
- For a given hypothesis class, we can also compute the *true error*, which is the expected error over the input distribution:

$$\sum_{\mathbf{x}} E_P \left[(y - h(\mathbf{x}))^2 | \mathbf{x} \right] P(\mathbf{x})$$

  (if $\mathbf{x}$ continuous, sum becomes integral with appropriate conditions).
- We will decompose this expectation into three components

Recall: Statistics 101

- Let X be a random variable with possible values $x_i, i = 1 \ldots n$ and with probability distribution $P(X)$
- The *expected value* or *mean* of X is:

$$E[X] = \sum_{i=1}^{n} x_i P(x_i)$$

- If X is continuous, roughly speaking, the sum is replaced by an integral, and the distribution by a density function
- The *variance* of X is:

$$\begin{aligned} Var[X] &= E[(X - E(X))^2] \\ &= E[X^2] - (E[X])^2 \end{aligned}$$

The variance lemma

$$\begin{aligned}
Var[X] &= E[(X - E[X])^2] \\
&= \sum_{i=1}^{n} (x_i - E[X])^2 P(x_i) \\
&= \sum_{i=1}^{n} (x_i^2 - 2x_i E[X] + (E[X])^2) P(x_i) \\
&= \sum_{i=1}^{n} x_i^2 P(x_i) - 2E[X] \sum_{i=1}^{n} x_i P(x_i) + (E[X])^2 \sum_{i=1}^{n} P(x_i) \\
&= E[X^2] - 2E[X]E[X] + (E[X])^2 \cdot 1 \\
&= E[X^2] - (E[X])^2
\end{aligned}$$

We will use the form:

$$E[X^2] = (E[X])^2 + Var[X]$$

Bias-variance decomposition

- Simple algebra:

$$\begin{aligned} E_P\left[(y-h(\mathbf{x}))^2|\mathbf{x}\right] &= E_P\left[(h(\mathbf{x}))^2 - 2yh(\mathbf{x}) + y^2|\mathbf{x}\right] \\ &= E_P\left[(h(\mathbf{x}))^2|\mathbf{x}\right] + E_P\left[y^2|\mathbf{x}\right] - 2E_P[y|\mathbf{x}]E_P\left[h(\mathbf{x})|\mathbf{x}\right] \end{aligned}$$

- Let $\bar{h}(\mathbf{x}) = E_P[h(\mathbf{x})|\mathbf{x}]$ denote the *mean prediction* of the hypothesis at $\mathbf{x}$, when h is trained with data drawn from P
- For the first term, using the variance lemma, we have:

$$E_P[(h(\mathbf{x}))^2|\mathbf{x}] = E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2|\mathbf{x}] + (\bar{h}(\mathbf{x}))^2$$

- Note that $E_P[y|\mathbf{x}] = E_P[f(\mathbf{x}) + \epsilon|\mathbf{x}] = f(\mathbf{x})$ (because of linearity of expectation and the assumption on $\epsilon \sim \mathcal{N}(0,\sigma)$)
- For the second term, using the variance lemma, we have:

$$E_P[y^2|\mathbf{x}] = E_P[(y - f(\mathbf{x}))^2|\mathbf{x}] + (f(\mathbf{x}))^2$$

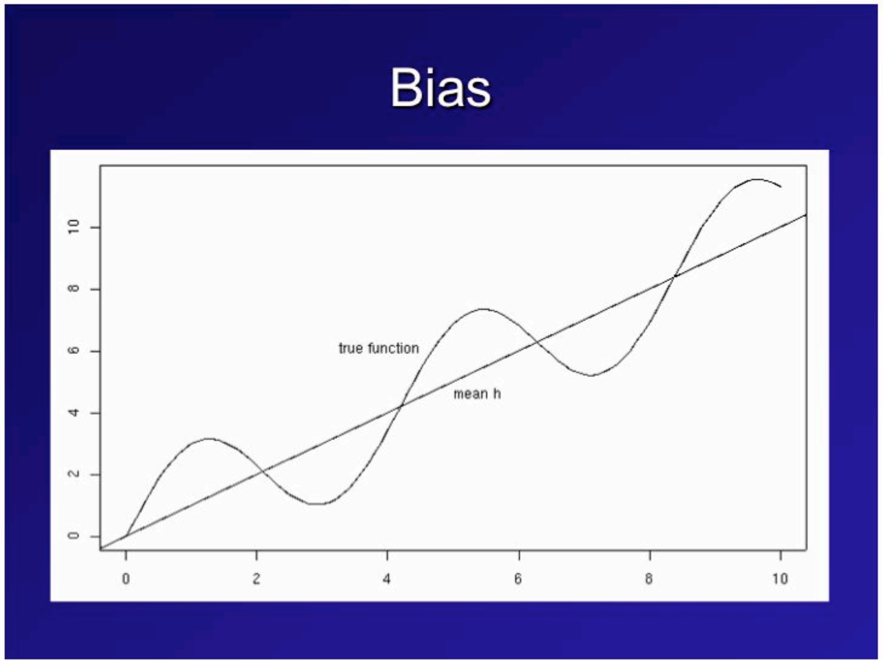
Bias-variance decomposition (2)

- Putting everything together, we have:

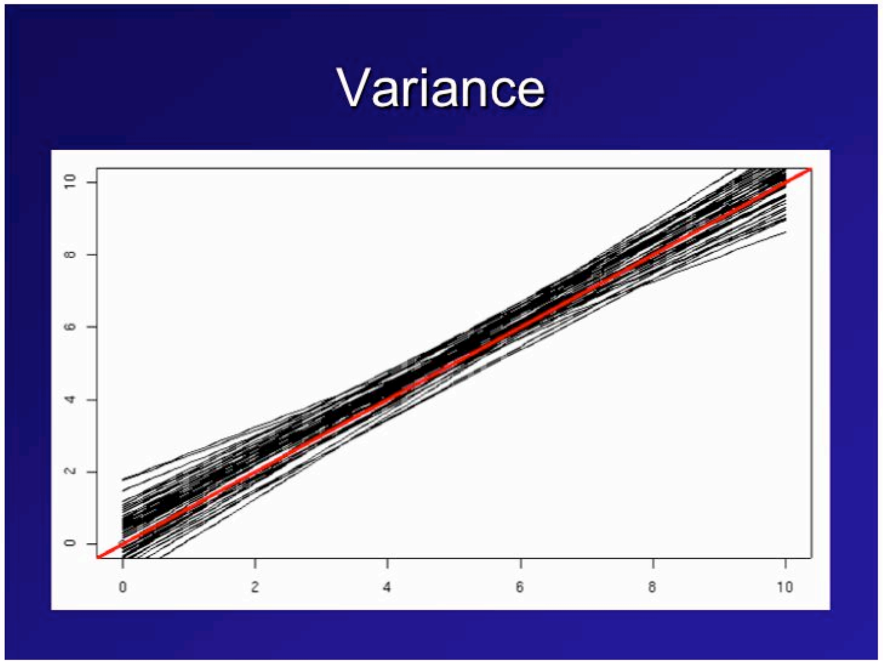
$$\begin{aligned}
E_P\left[(y - h(\mathbf{x}))^2|\mathbf{x}\right] &= E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2|\mathbf{x}] + (\bar{h}(\mathbf{x}))^2 \\
&+ E_P[(y - f(\mathbf{x}))^2|\mathbf{x}] + (f(\mathbf{x}))^2 - 2f(\mathbf{x})\bar{h}(\mathbf{x}) \\
&= E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2|\mathbf{x}] + (f(\mathbf{x}) - \bar{h}(\mathbf{x}))^2 \\
&+ E[(y - f(\mathbf{x}))^2|\mathbf{x}]
\end{aligned}$$

- The first term, $E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2|\mathbf{x}]$, is the *variance* of the hypothesis h at $\mathbf{x}$, when trained with finite data sets sampled randomly from P
- The second term, $(f(\mathbf{x}) - \bar{h}(\mathbf{x}))^2$, is the *squared bias* (or systematic error) which is associated with the class of hypotheses we are considering
- The last term, $E[(y - f(\mathbf{x}))^2|\mathbf{x}]$ is the *noise*, which is due to the problem at hand, and cannot be avoided

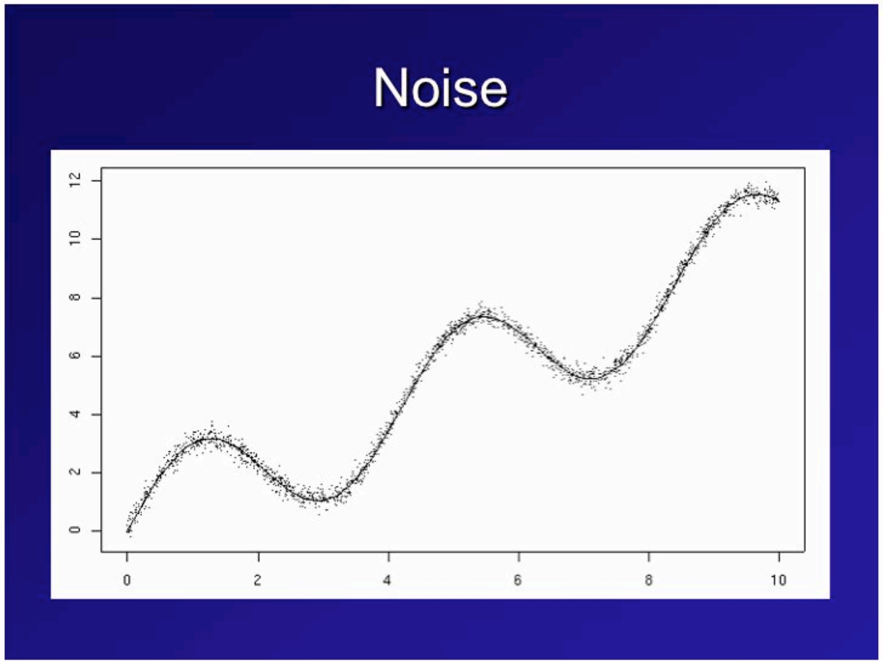
Example revisited: Bias



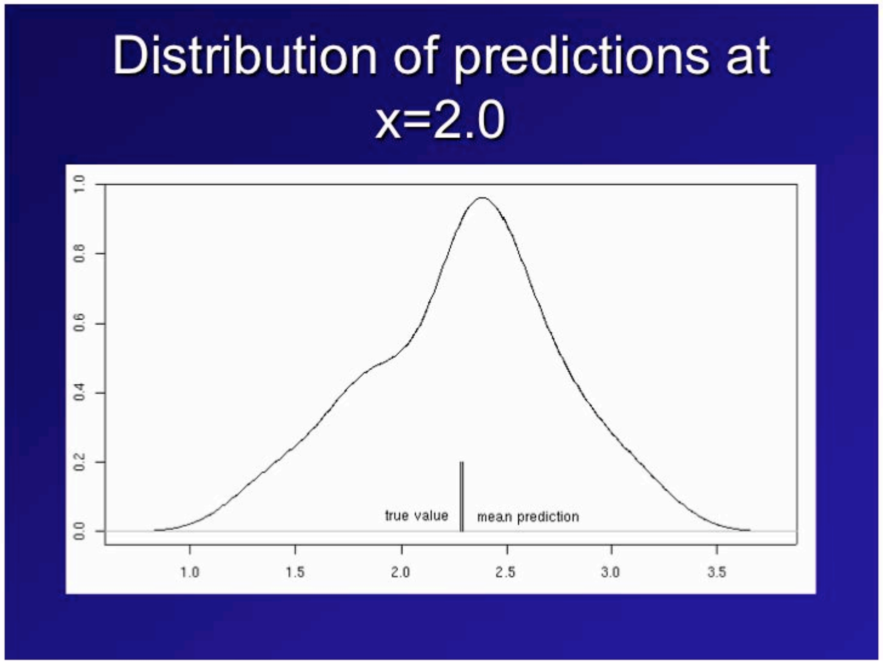
Example revisited: Variance



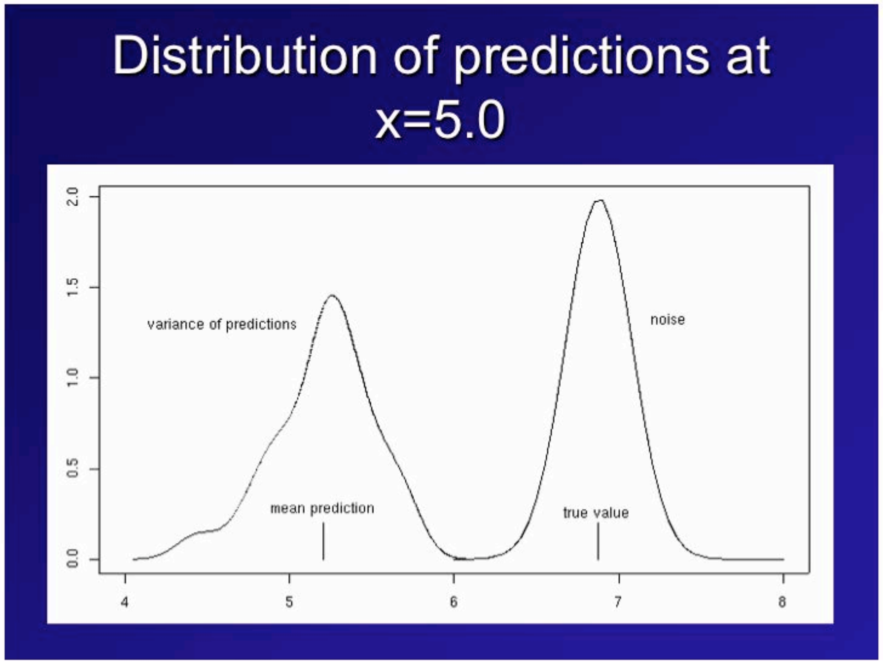
Example revisited: Noise



A point with low bias



A point with high bias



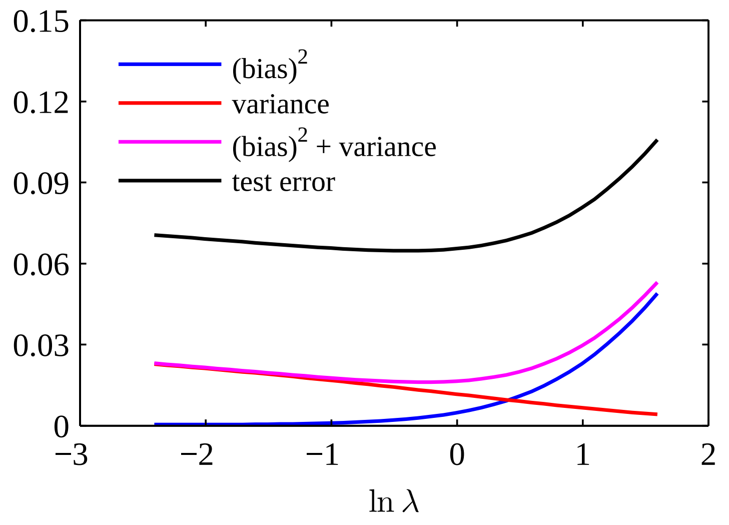
Average error of a hypothesis class

- In the example before, sampling points uniformly and doing linear regression leads to high bias on the average
- If we sampled points around the period of the sine function, the bias would be very low, even with a linear hypothesis
- In order to estimate the true error, and the true bias/variance of a hypothesis class for a given problem in closed form, we need to know $P(\mathbf{x})$
- Instead, in practice we estimate it empirically using cross-validation: instead of $P(\mathbf{x})$, we use samples, which we assume are drawn i.i.d. from $P(\mathbf{x})$

Bias-variance trade-off

- Consider fitting a small degree vs. a high degree polynomial
- Which one do you expect to have higher bias? Higher variance?

Error decomposition



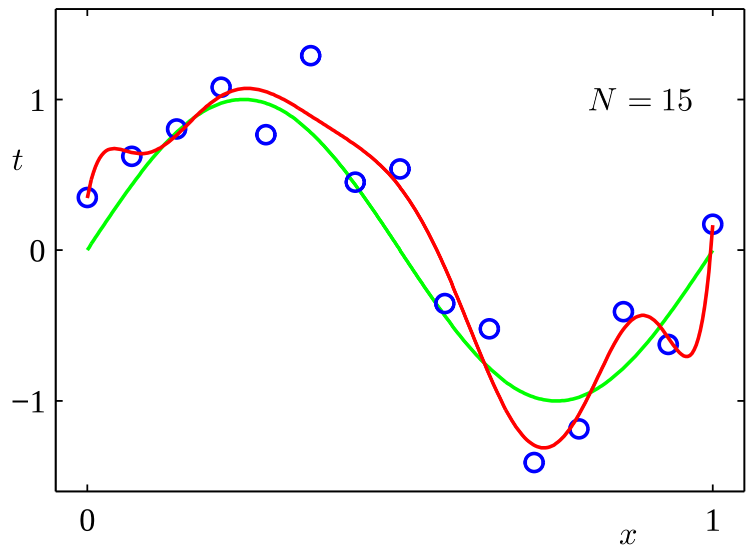
- The bias-variance sum approximates well the test error over a set of 1000 points
- x-axis measures the hypothesis complexity (decreasing left-to-right)
- Simple hypotheses usually have high bias (bias will be high at many points, so it will likely be high for many possible input distributions)
- Complex hypotheses have high variance: the hypothesis is very dependent on the data set on which it was trained.

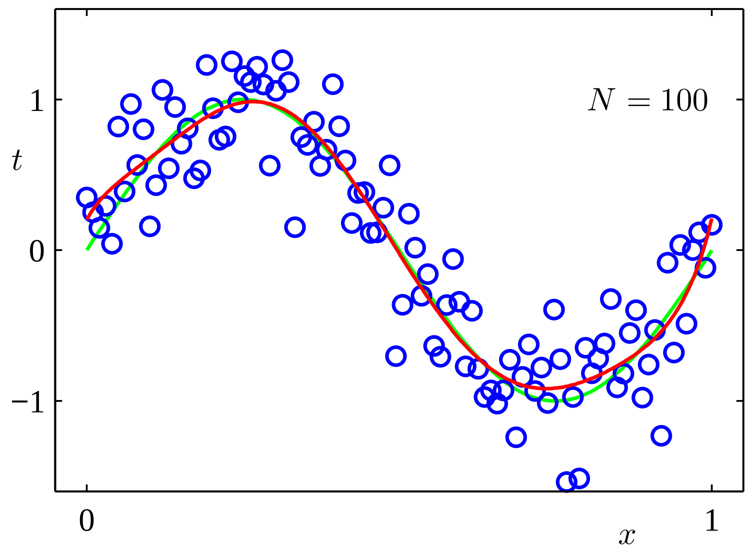
Bias-variance trade-off

- Typically, bias comes from not having good hypotheses in the considered class
- Variance results from the hypothesis class containing “too many” hypotheses
- Hence, we are faced with a *trade-off*: choose a more expressive class of hypotheses, which will generate higher variance, or a less expressive class, which will generate higher bias
- The trade-off depends also on how much data is available

More on overfitting

- Overfitting depends on the amount of data, relative to the complexity of the hypothesis
- With more data, we can explore more complex hypotheses spaces, and still find a good solution





Linear models in general

- By linear models, we mean that the hypothesis function $h_{\mathbf{w}}(\mathbf{x})$ is a *linear function of the parameters* $\mathbf{w}$
- This *does not mean the* $h_{\mathbf{w}}(\mathbf{x})$ *is a linear function of the input vector* $\mathbf{x}$ (e.g., polynomial regression)
- In general

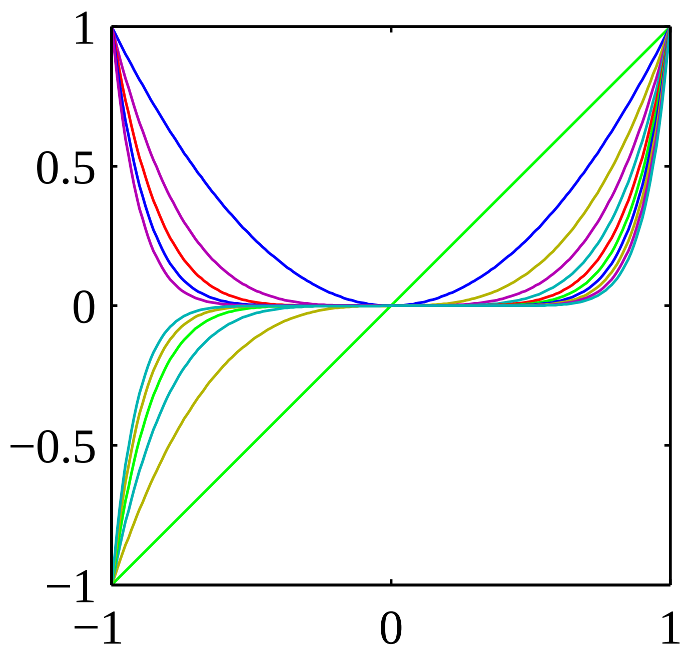
$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{k=0}^{K-1} w_k \phi_k(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

  where ϕ_k are called basis functions
- As usual, we will assume that $\phi_0(\mathbf{x}) = 1, \forall \mathbf{x}$, to create a bias term
- The hypothesis can alternatively be written as:

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{\Phi}\mathbf{w}$$

  where $\mathbf{\Phi}$ is a matrix with one row per instance; row j contains $\phi(\mathbf{x}_j)$.
- Basis functions are *fixed*

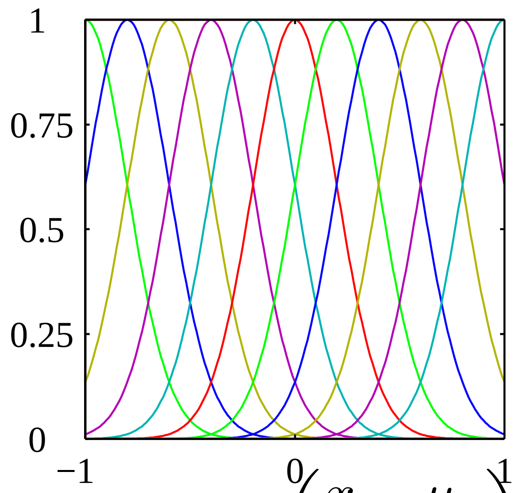
Example basis functions: Polynomials



$$\phi_k(x) = x^k$$

"Global" functions: a small change in x may cause large change in the output of many basis functions

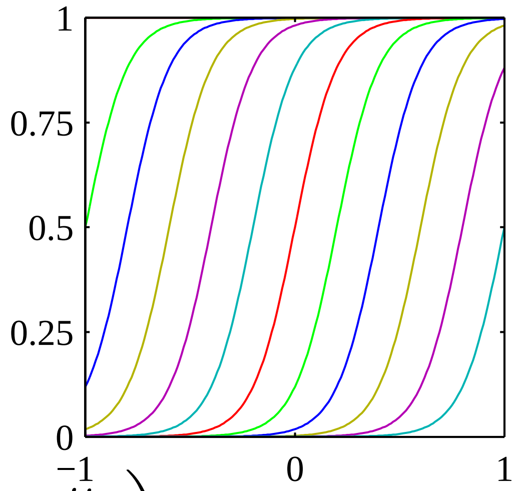
Example basis functions: Gaussians



$$\phi_k(x) = \exp\left(\frac{x-\mu_k}{2\sigma^2}\right)$$

- μ_k controls the position along the x-axis
- σ controls the width (activation radius)
- μ_k, σ fixed for now (later we discuss adjusting them)
- Usually thought as "local" functions: if σ is relatively small, a small change in x only causes a change in the output of a few basis functions (the ones with means close to x)

Example basis functions: Sigmoidal



$$\phi_k(x) = \sigma\left(\frac{x-\mu_k}{s}\right) \text{ where } \sigma(a) = \frac{1}{1+\exp(-a)}$$

- μ_k controls the position along the x-axis
- s controls the slope
- μ_k, s fixed for now (later we discuss adjusting them)
- "Local" functions: a small change in x only causes a change in the output of a few basis (most others will stay close to 0 or 1)

Minimizing the mean-squared error

- Recall from last time: we want to find the weight vector $\mathbf{w}^* = \arg\min_{\mathbf{w}} J_D(\mathbf{w})$, where:

$$J_D(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{m}(h_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 = \frac{1}{2}(\mathbf{\Phi w} - \mathbf{y})^T(\mathbf{\Phi w} - \mathbf{y})$$

- Compute the gradient and set it to 0:

$$\nabla_{\mathbf{w}} J_D(\mathbf{w}) = \frac{1}{2}\nabla_{\mathbf{w}}(\mathbf{w}^T\mathbf{\Phi}^T\mathbf{\Phi w} - \mathbf{w}^T\mathbf{\Phi}^T\mathbf{y} - \mathbf{y}^T\mathbf{\Phi w} + \mathbf{y}^T\mathbf{y}) = \mathbf{\Phi}^T\mathbf{\Phi w} - \mathbf{\Phi}^T\mathbf{y} = 0$$

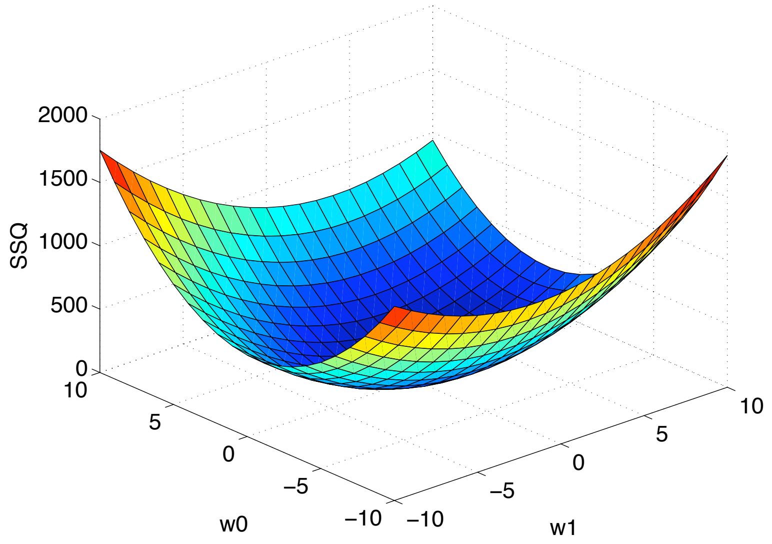
- Solve for $\mathbf{w}$:

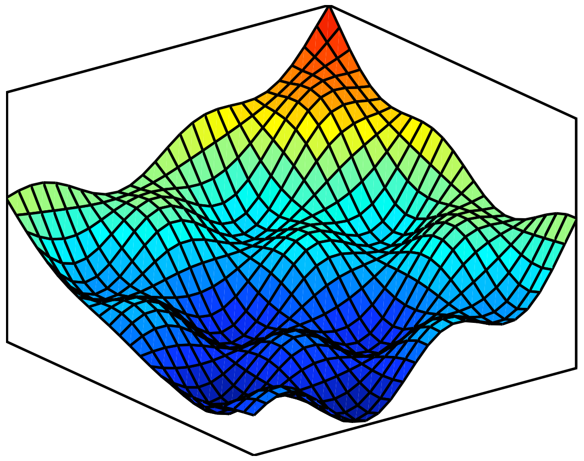
$$\mathbf{w} = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{y}$$

- What if $\mathbf{\Phi}$ is too big to compute this explicitly?

Gradient descent

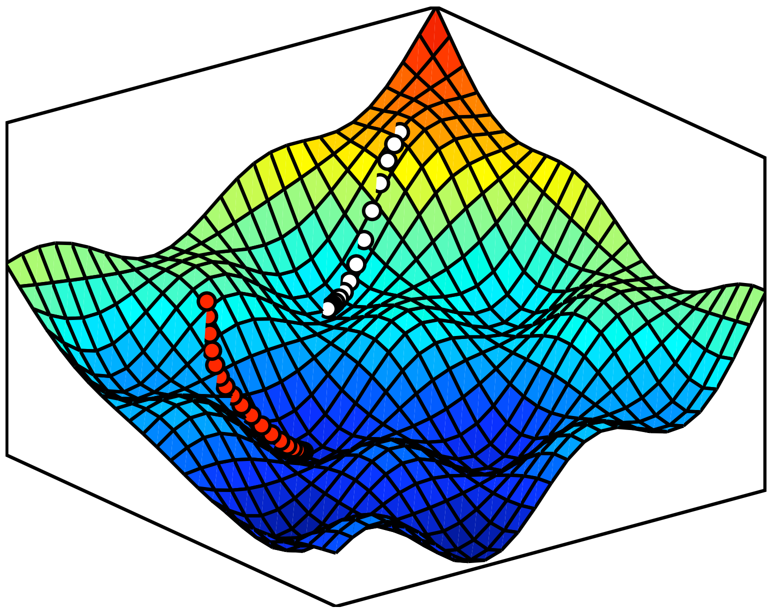
- The gradient of J at a point $\mathbf{w}$ can be thought of as a vector indicating which way is "uphill".





- If this is an error function, we want to move "downhill" on it, i.e., in the direction opposite to the gradient

Example gradient descent traces



- In general, there may be may local optima
- Final solution depends on the initial parameters

Gradient descent algorithm

- The basic algorithm assumes that ∇J is easily computed
- We want to produce a sequence of vectors $\mathbf{w}^1, \mathbf{w}^2, \mathbf{w}^3, \ldots$ with the goal that:
  - $J(\mathbf{w}^1) > J(\mathbf{w}^2) > J(\mathbf{w}^3) > \ldots$
  - $\lim_{i\to\infty} \mathbf{w}^i = \mathbf{w}$ and $\mathbf{w}$ is locally optimal.
- The algorithm: Given $\mathbf{w}^0$, do for $i = 0, 1, 2, \ldots$

$$\mathbf{w}^{i+1} = \mathbf{w}^i - \alpha_i \nabla J(\mathbf{w}^i) \ ,$$

where $\alpha_i > 0$ is the *step size* or *learning rate* for iteration i.

Step size and convergence

- Convergence to a local minimum depends in part on the α_i.
- If they are too large (such as constant) oscillation or "bubbling" may occur.
  (This suggests the α_i should tend to zero as $i \to \infty$.)
- If they are too small, the $\mathbf{w}^i$ may not move far enough to reach a local minimum, or may do so very slowly.

Robbins-Monroe conditions

- The α_i are a Robbins-Monroe sequence if:

$$\sum_{i=0}^{\infty} \alpha_i = +\infty \text{ and } \sum_{i=0}^{\infty} \alpha_i^2 < \infty$$

- E.g., $\alpha_i = \frac{1}{i+1}$ (averaging)
- E.g., $\alpha_i = \frac{1}{2}$ for $i = 1 \dots T$, $\alpha_i = \frac{1}{2^2}$ for $i = T+1, \dots (T+1) + 2T$ etc
- These conditions, along with appropriate conditions on J are sufficient to ensure convergence of the $\mathbf{w}^i$ to a point $\mathbf{w}^\infty$ such that $\nabla J(\mathbf{w}^\infty) = 0$.
- Many variants are possible: e.g., we may use at each step *a random vector* with mean $\nabla J(\mathbf{w}^i)$; this is *stochastic gradient descent*.

“Batch” versus “On-line” optimization

- The error function, J_D, is a sum of errors attributed to each instance: $(J_D = J_1 + J_2 + \ldots + J_m.)$
- In *batch gradient descent*, the true gradient is computed at each step:

$$\nabla J_D = \nabla J_1 + \nabla J_2 + \ldots \nabla J_m.$$

- In *on-line (incremental) gradient descent*, at each iteration one instance, $i \in \{1, \ldots, m\}$, is chosen at random and only ∇J_i is used in the update.
- Linear case (LMS or Widrow-Hoff rule): pick instance i and update:

$$\mathbf{w}^{i+1} = \mathbf{w}^i + \alpha_i(y_i - \mathbf{w}^T\phi(\mathbf{x}_i))\phi(\mathbf{x}_i),$$

- Why prefer one or the other?

“Batch” versus “On-line” optimization

- Batch is simple, repeatable.
- On-line:
  - Requires less computation per step.
  - Randomization may help escape poor local minima.
  - Allows working with a stream of data, rather than a static set (hence “on-line”).

Termination

There are many heuristics for deciding when to stop gradient descent.

1. Run until $\|\nabla J\|$ is smaller than some threshold.
2. Run it for as long as you can stand.
3. Run it for a short time from 100 different starting points, see which one is doing best, goto 2.
4. …

Gradient descent in linear models and beyond

- In linear models, gradient descent can be used with larger data sets than the exact solution method
- Very useful if the data is *non-stationary* (i.e., the data distribution changes over time)
- In this case, use *constant learning rates* (not obeying Robbins-Munro conditions)
- Crucial method for non-linear function approximation (where closed-form solutions for the parameters are impossible)

Annoyances:

- Speed of convergence depends on the learning rate schedule
- If the error function has local minima wrt the parameter vector, randomizing the initial parameters is crucial

Recall: Linear function approximation

- Given a set of examples $\langle \mathbf{x}_i, y_i \rangle_{i=1\ldots m}$, we fit a hypothesis

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{k=0}^{K-1} w_k \phi_k(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

  where ϕ_k are called basis functions
- The best $\mathbf{w}$ is considered the one which minimizes the sum-squared error over the training data:

$$\sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2$$

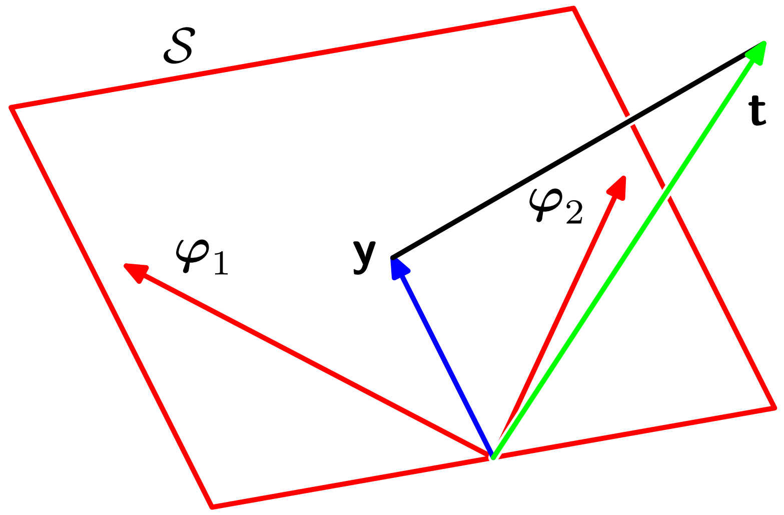
- We can find the best $\mathbf{w}$ in closed form:

$$\mathbf{w} = (\mathbf{\Phi}^T \mathbf{\Phi})^{-1} \mathbf{\Phi}^T \mathbf{y}$$

  or by gradient descent (if we want to avoid the matrix inversion)

Coming back to mean-squared error function...

- Good intuitive feel (small errors are ignored, large errors are penalized)
- Nice math (closed-form solution, unique global optimum)
- Geometric interpretation (in our notation, $\mathbf{t}$ is $\mathbf{y}$ and $\mathbf{y}$ is $h_{\mathbf{w}}(\mathbf{x})$)



- Any other interpretation?

A probabilistic assumption

- Assume y_i is a noisy target value, generated from a hypothesis $h_{\mathbf{w}}(\mathbf{x})$
- More specifically, assume that there exists $\mathbf{w}$ such that:

$$y_i = h_{\mathbf{w}}(\mathbf{x}_i) + \epsilon_i$$

  where ϵ_i is random variable (noise) drawn independently for each $\mathbf{x}_i$ according to some Gaussian (normal) distribution with mean zero and variance σ.
- How should we choose the parameter vector $\mathbf{w}$?

Bayes theorem in learning

Let h be a hypothesis and D be the set of training data.
Using Bayes theorem, we have:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)},$$

where:

- $P(h)$ is the *prior probability of hypothesis h*
- $P(D) = \int_h P(D|h)P(h)$ is the probability of training data D (normalization, independent of h)
- $P(h|D)$ is the probability of h given D
- $P(D|h)$ is the probability of D given h (*likelihood of the data*)

Choosing hypotheses

- What is the most probable hypothesis given the training data?
- *Maximum a posteriori (MAP)* hypothesis h_{MAP}:

$$\begin{aligned} h_{MAP} &= \arg\max_{h \in H} P(h|D) \\ &= \arg\max_{h \in H} \frac{P(D|h)P(h)}{P(D)} \text{(using Bayes theorem)} \\ &= \arg\max_{h \in H} P(D|h)P(h) \end{aligned}$$

  Last step is because $P(D)$ is independent of h (so constant for the maximization)

- This is the Bayesian answer (more in a minute)

Maximum likelihood estimation - review

Let

$$X_1 \ , \ X_2 \ , \ \cdots \ , \ X_n \ ,$$

be *independent, identically distributed* random variables each having density function $f(X;\sigma)$ with *unknown parameter* σ .

By *independence* , the *joint density function* is

$$f(X_1, X_2, \ \cdots \ , X_n \ ; \sigma) \ = \ f(X_1;\sigma) \ f(X_2;\sigma) \ \cdots \ f(X_n;\sigma) \ .$$

DEFINITION: The *maximum likelihood estimate* $\hat{\sigma}$ is the value of σ that *maximizes* $f(X_1, X_2, \ \cdots \ , X_n \ ; \sigma)$.

Maximum likelihood estimation - review

EXAMPLE: Consider $X \sim N(\mu, \sigma)$. For our zero mean normally distributed RV we have

$$f(x_1, x_2, \cdots, x_n ; \sigma) = \frac{e^{-\frac{1}{2\sigma^2}\sum_{k=1}^{n}(x_k-\mu)^2}}{(\sqrt{2\pi}\ \sigma)^n} .$$

density of Normal distrib

To find the maximum (with respect to μ and σ) we set

$$\frac{d}{d\mu}\left(\frac{e^{-\frac{1}{2\sigma^2}\sum_{k=1}^{n}(x_k-\mu)^2}}{\sigma^n}\right) = 0 ,$$

$$\frac{d}{d\sigma}\left(\frac{e^{-\frac{1}{2\sigma^2}\sum_{k=1}^{n}(x_k-\mu)^2}}{\sigma^n}\right) = 0 ,$$

(if derivatives exist) or, *equivalently* , we set

$$\frac{d}{d\mu}\ \log\Big(\frac{e^{-\frac{1}{2\sigma^2}\sum_{k=1}^{n}(x_k-\mu)^2}}{\sigma^n}\Big)\ =\ \frac{d}{d\mu}\ \Big(-\frac{1}{2\sigma^2}\sum_{k=1}^{n}(x_k-\mu)^2-n\ \log\ \sigma\ \Big)\ =\ 0\ ,$$

$$\frac{d}{d\sigma}\ \log\Big(\frac{e^{-\frac{1}{2\sigma^2}\sum_{k=1}^{n}(x_k-\mu)^2}}{\sigma^n}\Big)\ =\ \frac{d}{d\sigma}\ \Big(-\frac{1}{2\sigma^2}\sum_{k=1}^{n}(x_k-\mu)^2-n\ \log\ \sigma\ \Big)\ =\ 0\ ,$$

(why *equivalent* ?) from which we get the *maximum likelihood estimate* of μ and σ

$$\frac{1}{\sigma^2}\sum_{k=1}^{n}(x_k-\mu)\ =\ 0\qquad\Rightarrow\quad\hat{\mu}\ =\ \frac{1}{n}\sum_{k=1}^{n}x_k\quad(\ =\ \bar{X}\quad!)\ ,$$

$$\frac{\sum_{k=1}^{n}(x_k-\mu)^2}{\sigma^3}\ -\ \frac{n}{\sigma}\ =\ 0\quad\Rightarrow\quad\hat{\sigma}\ =\ \frac{1}{\sqrt{n}}\Big(\sum_{k=1}^{n}(x_k-\hat{\mu})^2\Big)^{\frac{1}{2}}\quad(\ =\ S\ \ !)\ .$$

Maximum likelihood estimation

$$h_{MAP} = \arg\max_{h \in H} P(D|h)P(h)$$

- If we assume $P(h_i) = P(h_j)$ (all hypotheses are equally likely a priori) then we can further simplify, and choose the *maximum likelihood (ML) hypothesis*:

$$h_{ML} = \arg\max_{h \in H} P(D|h) = \arg\max_{h \in H} L(h)$$

- Standard assumption: the training examples are *independently identically distributed (i.i.d.)*
- This allows us to simplify $P(D|h)$:

$$P(D|h) = \prod_{i=1}^{m} P(\langle \mathbf{x_i}, y_i \rangle | h) = \prod_{i=1}^{m} P(y_i | \mathbf{x}_i; h) P(\mathbf{x}_i)$$

The $\log$ trick

- We want to maximize:

$$L(h) = \prod_{i=1}^{m} P(y_i|\mathbf{x}_i; h)P(\mathbf{x}_i)$$

  This is a product, and products are hard to maximize!

- Instead, we will maximize $\log L(h)$! (the log-likelihood function)

$$\log L(h) = \sum_{i=1}^{m} \log P(y_i|\mathbf{x}_i; h) + \sum_{i=1}^{m} \log P(\mathbf{x}_i)$$

- The second sum depends on D, but not on h, so it can be ignored in the search for a good hypothesis

Maximum likelihood for regression

- Adopt the assumption that:

$$y_i = h_{\mathbf{w}}(\mathbf{x}_i) + \epsilon_i,$$

  where $\epsilon_i \sim \mathcal{N}(0, \sigma)$.
- The best hypothesis maximizes the likelihood of $y_i - h_{\mathbf{w}}(\mathbf{x}_i) = \epsilon_i$
- Hence,

$$L(\mathbf{w}) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{y_i - h_{\mathbf{w}}(\mathbf{x}_i)}{\sigma}\right)^2}$$

  because the noise variables ϵ_i are from a Gaussian distribution

$$y_i \in N(h_w(x_i), \delta)$$

Applying the $\log$ trick

$$\begin{aligned}
\log L(\mathbf{w}) &= \sum_{i=1}^{m} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2}{\sigma^2}}\right) \\
&= \sum_{i=1}^{m} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \sum_{i=1}^{m} \frac{1}{2}\frac{(y_i - h_{\mathbf{w}}(\mathbf{x}_i))^2}{\sigma^2}
\end{aligned}$$

Maximizing the right hand side is the same as minimizing:

$$\sum_{i=1}^{m} \frac{1}{2}\frac{(y_i - h_w(\mathbf{x}_i))^2}{\sigma^2}$$

This is our old friend, the sum-squared-error function! (the constants that are independent of h can again be ignored)

Maximum likelihood hypothesis for least-squares estimators

- Under the assumption that the training examples are i.i.d. and that we have *Gaussian target noise*, the maximum likelihood parameters $\mathbf{w}$ are those minimizing the sum squared error:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(\mathbf{x_i}))^2$$

- This makes explicit the hypothesis behind minimizing the sum-squared error
- If the noise is not normally distributed, maximizing the likelihood will not be the same as minimizing the sum-squared error (see assignment 1)
- In practice, different loss functions may be needed

Regularization

- Remember the intuition: complicated hypotheses lead to overfitting
- Idea: change the error function to *penalize hypothesis complexity*:

$$J(\mathbf{w}) = J_D(\mathbf{w}) + \lambda J_{pen}(\mathbf{w})$$

  This is called *regularization* in machine learning and *shrinkage* in statistics
- λ is called *regularization coefficient* and controls how much we value fitting the data well, vs. a simple hypothesis

Regularization for linear models

- A squared penalty on the weights would make the math work nicely in our case:

$$\frac{1}{2}(\mathbf{\Phi w} - \mathbf{y})^T(\mathbf{\Phi w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

- This is also known as L_2 *regularization*, or *weight decay* in neural networks
- By re-grouping terms, we get:

$$J_D(\mathbf{w}) = \frac{1}{2}(\mathbf{w}^T(\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I})\mathbf{w} - \mathbf{w}^T\mathbf{\Phi}^T\mathbf{y} - \mathbf{y}^T\mathbf{\Phi w} + \mathbf{y}^T\mathbf{y})$$

- Optimal solution (obtained by solving $\nabla_{\mathbf{w}} J_D(\mathbf{w}) = 0$)

$$\mathbf{w} = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^T\mathbf{y}$$

What L_2 regularization does

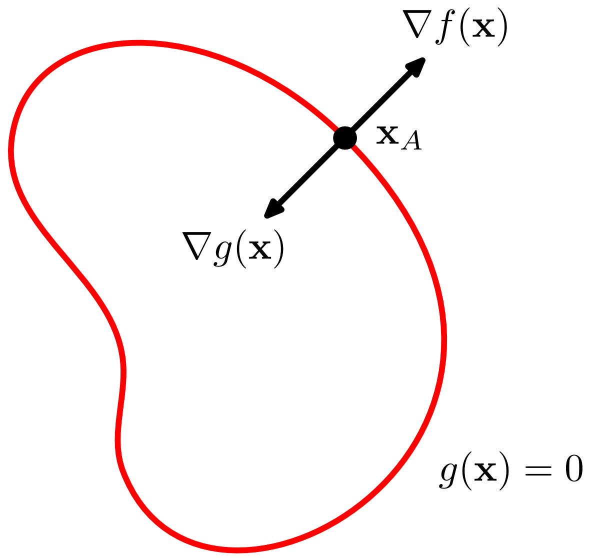
$$\arg\min_{\mathbf{w}} \frac{1}{2}(\mathbf{\Phi w} - \mathbf{y})^T(\mathbf{\Phi w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^T\mathbf{y}$$

- If $\lambda = 0$, the solution is the same as in regular least-squares linear regression
- If $\lambda \to \infty$, the solution $\mathbf{w} \to 0$
- Positive λ will cause the magnitude of the weights to be smaller than in the usual linear solution
- A different view of regularization: we want to optimize the error while keeping the L_2 norm of the weights, $\mathbf{w}^T\mathbf{w}$, bounded.

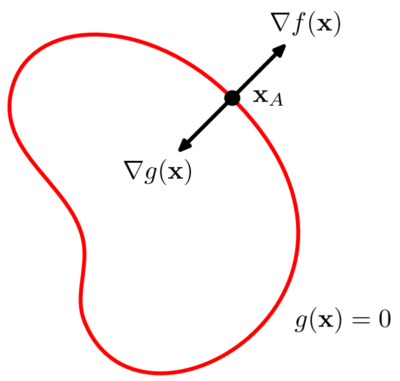
Detour: Constrained optimization

Suppose we want to find

$$\min_{\mathbf{w}} f(\mathbf{w})$$
$$\text{such that } g(\mathbf{w}) = 0$$



Detour: Lagrange multipliers

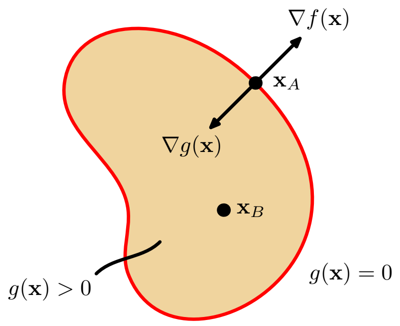


- ∇g has to be orthogonal to the constraint surface (red curve)
- At the optimum, ∇f and ∇g have to be parallel (in same or opposite direction)
- Hence, there must exist some $\lambda \in \mathbb{R}$ such that $\nabla f + \lambda \nabla g = 0$
- *Lagrangian function*: $L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$
  λ is called *Lagrange multiplier*
- We obtain the solution to our optimization problem by setting both $\nabla_{\mathbf{x}} L = 0$ and $\frac{\partial L}{\partial \lambda} = 0$

Detour: Inequality constraints

- Suppose we want to find

$$\min_{\mathbf{w}} f(\mathbf{w})$$
$$\text{such that } g(\mathbf{w}) \quad \geq \quad 0$$



- In the interior ($g(\mathbf{x}) > 0$) - simply find $\nabla f(\mathbf{x}) = 0$
- On the boundary ($g(\mathbf{x}) = 0$) - same situation as before, but the sign matters this time
  For minimization, we want ∇f pointing in the same direction as ∇g

Detour: KKT conditions

- Based on the previous observations, let the Lagrangian be $L(\mathbf{x}, \lambda) = f(\mathbf{x}) - \lambda g(\mathbf{x})$
- We minimize L wrt $\mathbf{x}$ subject to the following constraints:

$$\begin{aligned} \lambda &\geq 0 \\ g(\mathbf{x}) &\geq 0 \\ \lambda g(\mathbf{x}) &= 0 \end{aligned}$$

- These are called *Karush-Kuhn-Tucker (KKT) conditions*

L_2 Regularization for linear models revisited

- Optimization problem: minimize error while keeping norm of the weights bounded

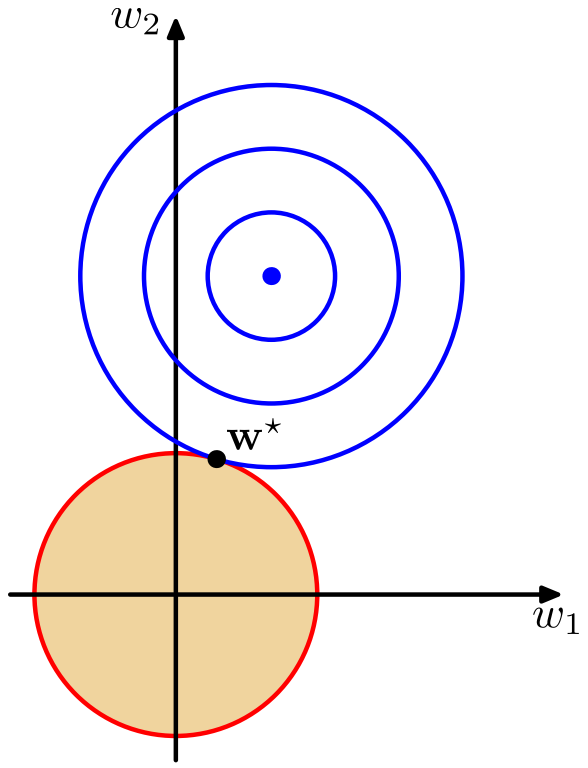
$$\begin{aligned} \min_{\mathbf{w}} J_D(\mathbf{w}) &= \min_{\mathbf{w}} (\mathbf{\Phi w} - \mathbf{y})^T (\mathbf{\Phi w} - \mathbf{y}) \\ \text{such that } \mathbf{w}^T\mathbf{w} &\leq \eta \end{aligned}$$

- The Lagrangian is:

$$L(\mathbf{w}, \lambda) = J_D(\mathbf{w}) - \lambda(\eta - \mathbf{w}^T\mathbf{w}) = (\mathbf{\Phi w} - \mathbf{y})^T(\mathbf{\Phi w} - \mathbf{y}) + \lambda \mathbf{w}^T\mathbf{w} - \lambda\eta$$

- For a fixed λ, and $\eta = \lambda^{-1}$, the best $\mathbf{w}$ is the same as obtained by weight decay

Visualizing regularization (2 parameters)



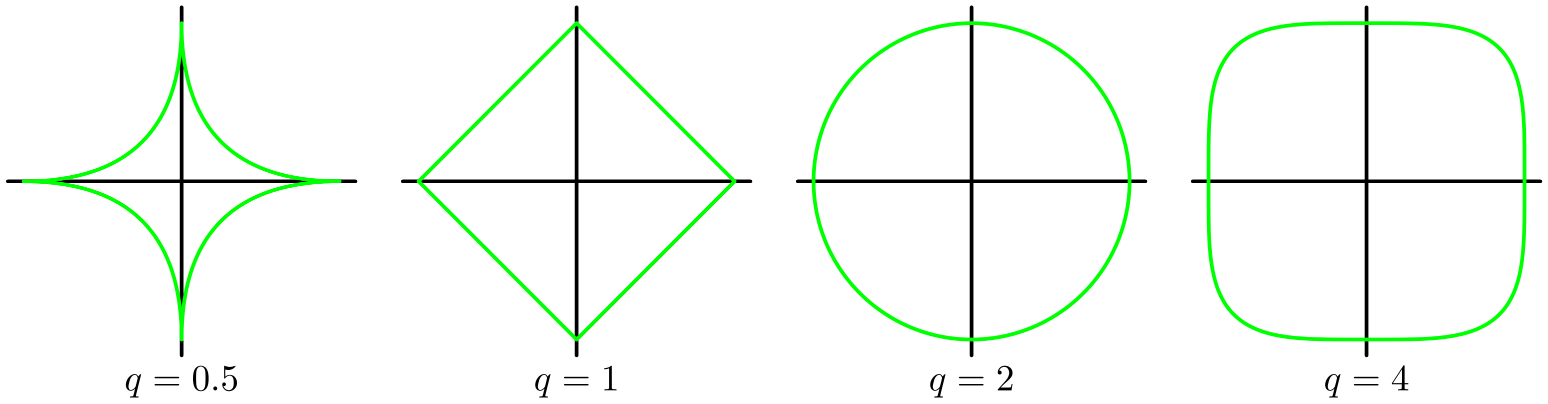
$$\mathbf{w}^* = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}\mathbf{y}$$

Pros and cons of L_2 regularization

- If λ is at a "good" value, regularization helps to avoid overfitting
- Choosing λ may be hard: cross-validation is often used
- If there are irrelevant features in the input (i.e. features that do not affect the output), L_2 will give them small, but non-zero weights.
- Ideally, irrelevant input should have weights exactly equal to 0.

$L_q, q > 0$ regularization

- $$\arg\min_{\mathbf{w}} \frac{1}{2}(\mathbf{\Phi w} - \mathbf{y})^T(\mathbf{\Phi w} - \mathbf{y}) + \frac{\lambda}{2}\sum_{k=0}^{K-1} |w_k|^q$$



L_1 Regularization for linear models

- Instead of requiring the L_2 norm of the weight vector to be bounded, make the requirement on the L_1 norm:

$$\begin{aligned} \min_{\mathbf{w}} J_D(\mathbf{w}) &= \min_{\mathbf{w}} (\mathbf{\Phi}\mathbf{w} - \mathbf{y})^T (\mathbf{\Phi}\mathbf{w} - \mathbf{y}) \\ \text{such that } \sum_{i=1}^{n} |w_i| &\leq \eta \end{aligned}$$

- This yields an algorithm called Lasso (Tibshirani, 1996)

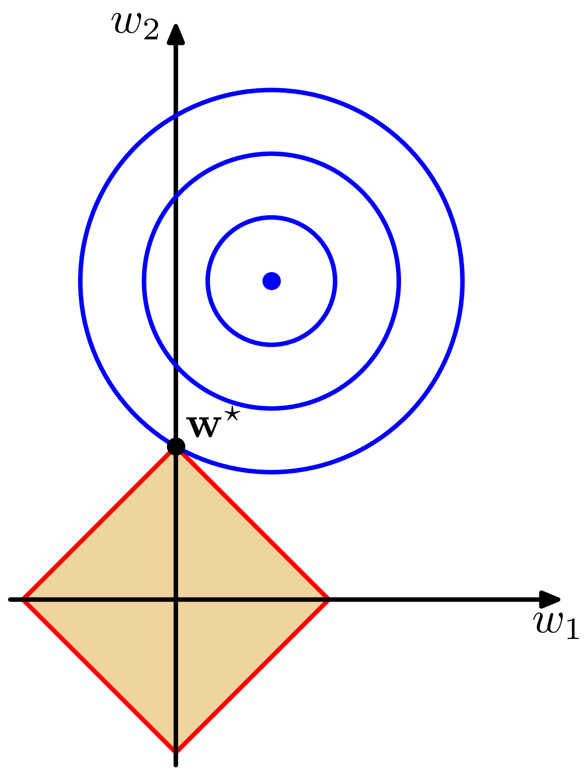
Solving L_1 regularization

- The optimization problem is a quadratic program
- There is one constraint for each possible sign of the weights (2^n constraints for n weights)
- For example, with two weights:

$$\begin{aligned} \min_{w_1,w_2} \quad & \sum_{j=1}^{m}(y_j - w_1x_1 - w_2x_2)^2 \\ \text{such that } w_1 + w_2 &\leq \eta \\ w_1 - w_2 &\leq \eta \\ -w_1 + w_2 &\leq \eta \\ -w_1 - w_2 &\leq \eta \end{aligned}$$

- Solving this program directly can be done for problems with a small number of inputs

Visualizing L_1 regularization



- If λ is big enough, the circle is very likely to intersect the diamond at one of the corners
- This makes L_1 regularization much more likely to make some weights *exactly* 0

Pros and cons of L_1 regularization

- If there are irrelevant input features, Lasso is likely to make their weights 0, while L_2 is likely to just make all weights small
- Lasso is biased towards providing *sparse solutions* in general
- Lasso optimization is computationally more expensive than L_2
- More efficient solution methods have to be used for large numbers of inputs (e.g. least-angle regression, 2003).
- L_1 methods of various types are very popular at the moment

Bayesian view of regularization

- Start with a *prior distribution* over hypotheses
- As data comes in, compute a *posterior distribution*
- We often work with *conjugate priors*, which means that when combining the prior with the likelihood of the data, one obtains the posterior in the same form as the prior
- Regularization can be obtained from particular types of prior (usually, priors that put more probability on simple hypotheses)
- E.g. L_2 regularization can be obtained using a circular Gaussian prior for the weights, and the posterior will also be Gaussian
- E.g. L_1 regularization uses double-exponential prior (see (Tibshirani, 1996))

Bayesian view of regularization

- Consider linear model

$$h_w(x) = w_0 + w_1 x$$

- Data points are generated from function

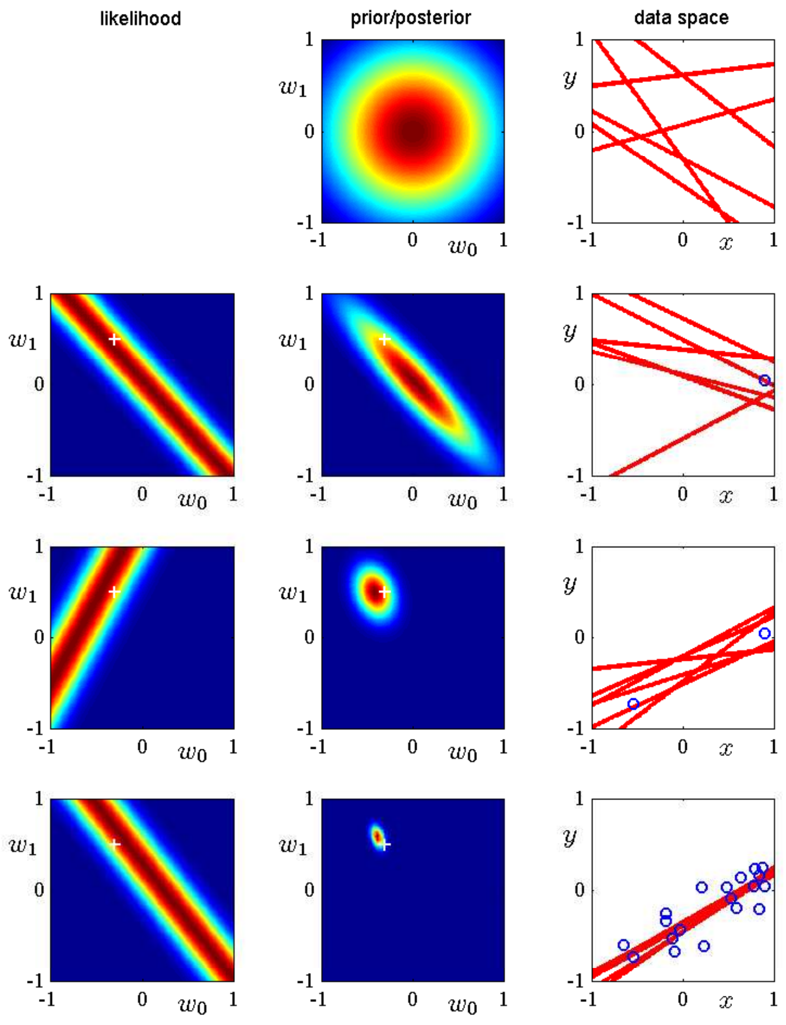
$$f(x, a) = a_0 + a_1 x = -0.3 + 0.5x$$

by generating x_n from uniform $U[0, 1]$ distribution and

$$y_n = f(x_n, a) + \epsilon_n$$

where $\epsilon_n \sim N(0, 0.2^2)$

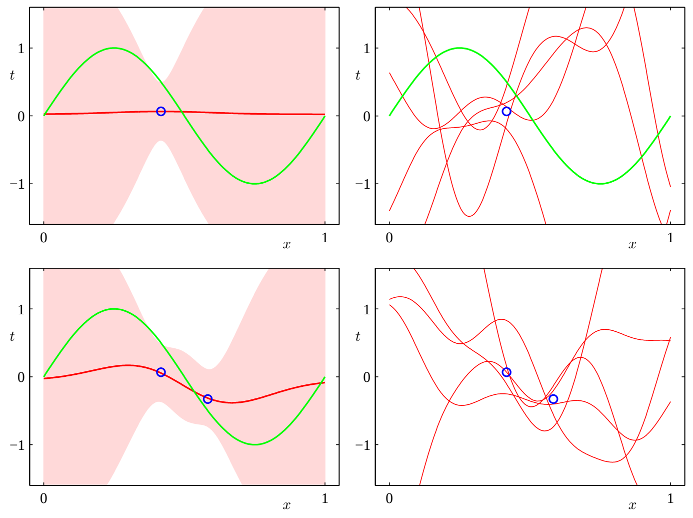
Bayesian view of regularization



Bayesian view of regularization

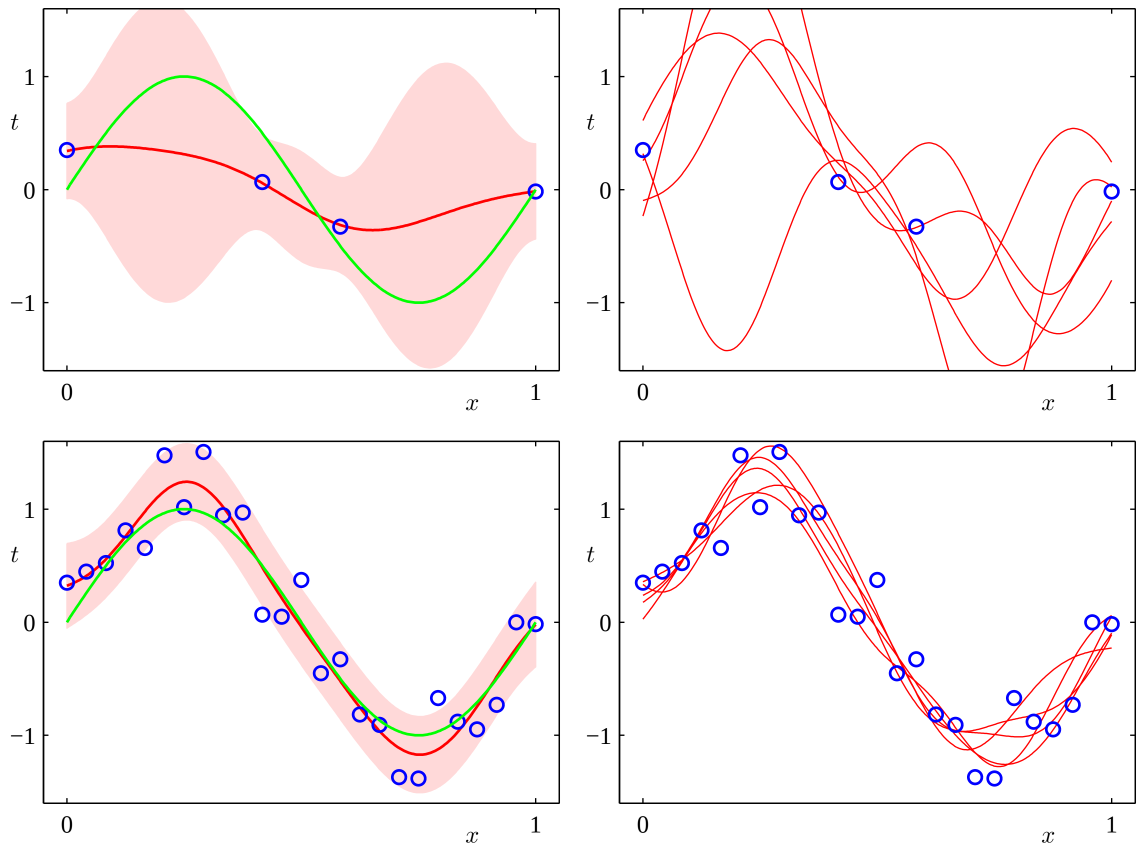
- Prior distribution of w is round Gaussian (first row, second column)
- First column shows the likelihood function for the recently added data point alone (1st, 2nd and 20th in the second, third and fourth row respectively)
- Posterior (second column) will be skewed by the data
- Samples of regression function $h_w(x)$ with w drawn from the posterior are shown in the third column
- White cross represents true parameter values $a_0 = -0.3, a_1 = 0.5$ and posterior concentrates around it more and more as more data are added

What does the Bayesian view give us?



- Circles are data points. Data size $N = 1, 2, 4, 25$
- Green is the true function $\sin(2\pi x)$
- Red lines on the left are drawn from the Gaussian posterior predictive distribution. Red shaded regions are one standard deviation away from the mean. Note that predictive uncertainty is least near data points
- Red lines on the right are plots of function $h_w(x)$ where w are generated from posterior distribution over w

What does the Bayesian view give us?



- Functions drawn from the posterior can be very different
- Uncertainty decreases where there are data points

What does the Bayesian view give us?

- Uncertainty estimates, i.e. how sure we are of the value of the function
- These can be used to guide active learning: ask about inputs for which the uncertainty in the value of the function is very high
- In the limit, Bayesian and maximum likelihood learning converge to the same answer
- In the short term, one needs a good prior to get good estimates of the parameters
- Sometimes the prior is overwhelmed by the data likelihood too early
- Using the Bayesian approach does NOT eliminate the need to do cross-validation in general
- More on this later...