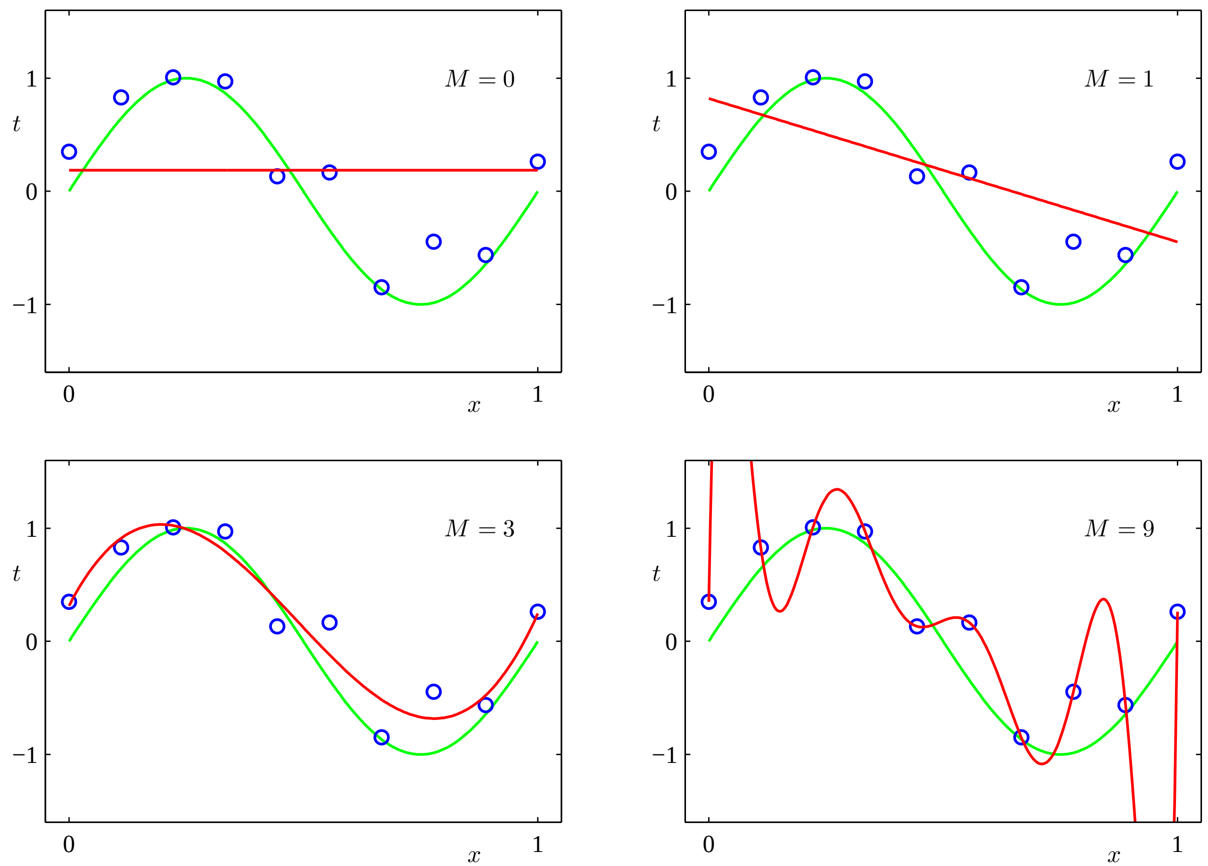
Lecture 2: More on linear methods for regression

- Overfitting and bias-variance trade-off
- Linear basis functions models
- Sequential (on-line, incremental) learning
- Why least-squares? A probabilistic analysis
- If we have time: Regularization

Recall: Linear and polynomial regression

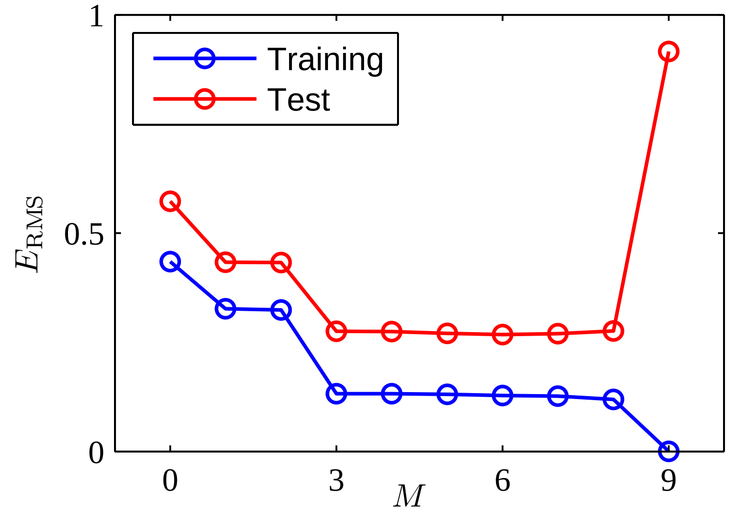
- Our first assumption was that it is good to minimize sum- (or mean-) squared error
- Algorithms that minimize this function are called *least-squares*
- Our second assumption was the linear form of the hypothesis class
- The terms were powers of the input variables (and possibly cross-terms of these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

Recall: Overfitting



The higher the degree of the polynomial, the more degrees of freedom, and the more capacity to "overfit" (think: memorize) the training data

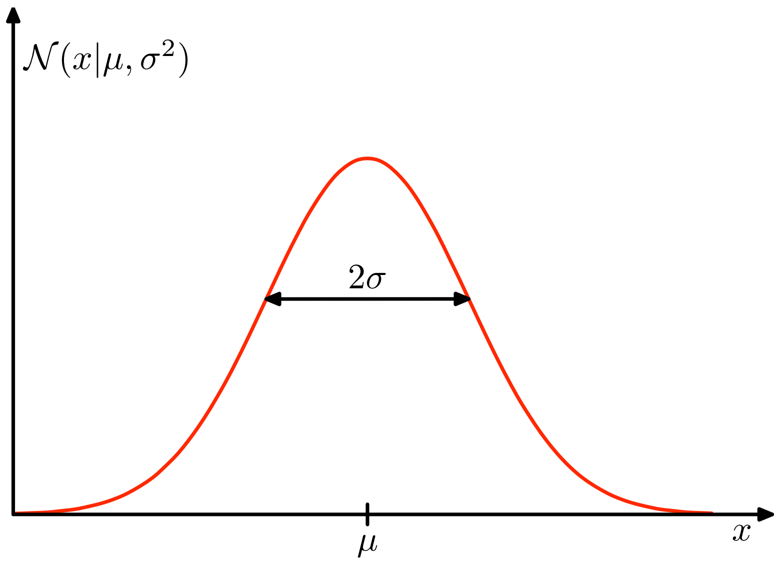
Recall: Typical overfitting plot



- The training error decreases with the degree of the polynomial, i.e. *the complexity of the hypothesis*
- The testing error, measured on independent data, decreases at first, then starts increasing
- Cross-validation helps us
  - Find a good hypothesis class
  - Report unbiased results

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



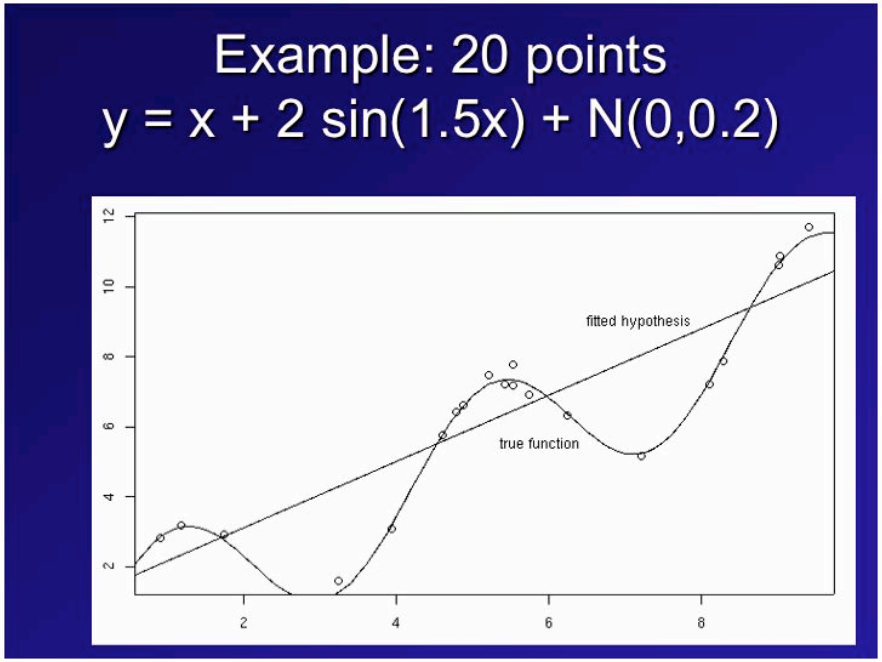
The anatomy of the error: Linear regression

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

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

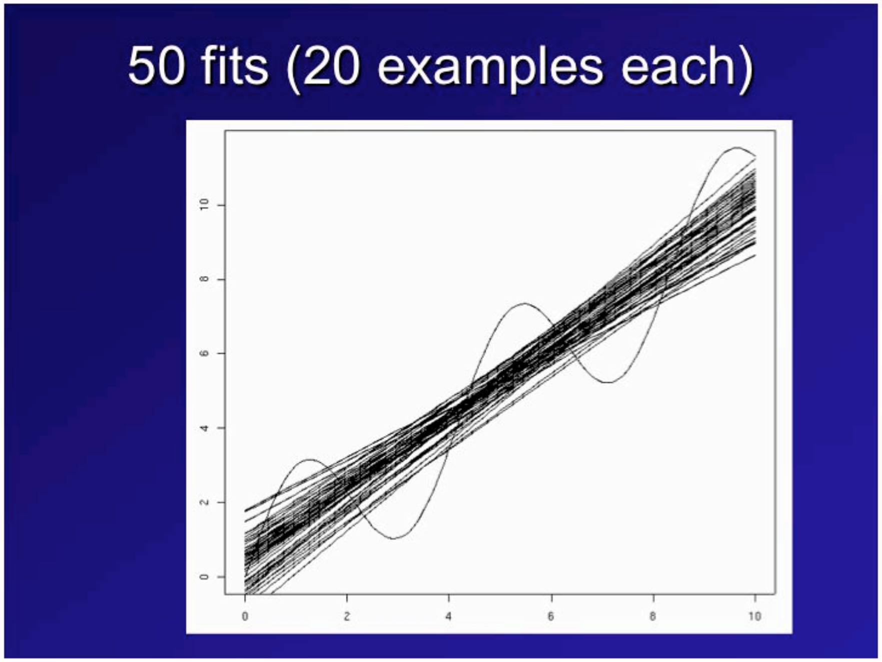
- Because of the hypothesis class that we chose (linear hypotheses) for some functions f we will have a *systematic prediction error*
- Depending on the data set we have, the parameters $\mathbf{w}$ that we find will be different

An example (Tom Dietterich)



- The sine is the true function
- The circles are the data points
- The straight line is the linear regression fit

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 $P(\langle \mathbf{x}, y \rangle)$
- The goal of the analysis is to compute, for an arbitrary new point $\mathbf{x}$,

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

  where y is the value of $\mathbf{x}$ that could be present in a data set, and the expectation is over all all training sets drawn according to P
- 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

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

Let $\bar{h}(\mathbf{x}) = E_P[h(\mathbf{x})]$ 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:

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

Note that $E_P[y] = E_P[f(\mathbf{x}) + \epsilon] = f(\mathbf{x})$

For the second term, using the variance lemma, we have:

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

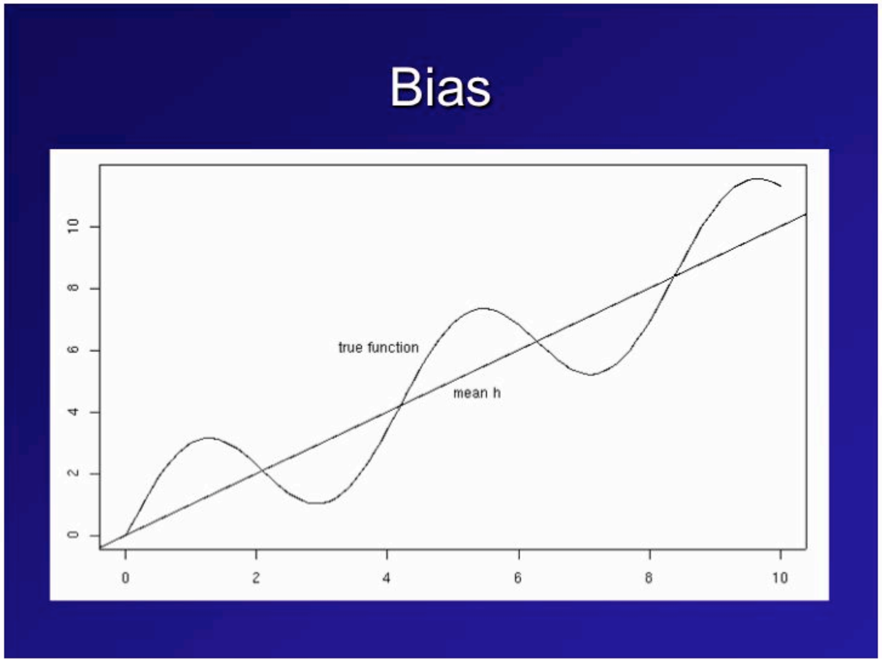
Bias-variance decomposition (2)

- Putting everything together, we have:

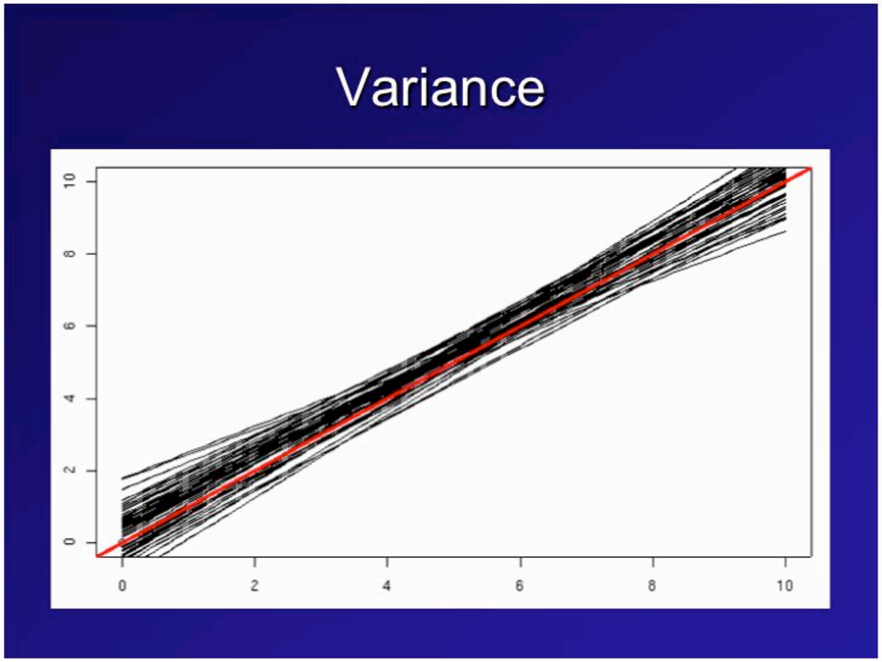
$$\begin{aligned} E_P\left[(y - h(\mathbf{x}))^2\right] &= E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2] + (\bar{h}(\mathbf{x}))^2 - 2f(\mathbf{x})\bar{h}(\mathbf{x}) \\ &+ E_P[(y - f(\mathbf{x}))^2] + (f(\mathbf{x}))^2 \\ &= E_P[(h(\mathbf{x}) - \bar{h}(\mathbf{x}))^2] + (f(\mathbf{x}) - \bar{h}(\mathbf{x}))^2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

- The first term is the*variance* of the hypothesis h when trained with finite data sets sampled randomly from P
- The second term is the *squared bias* (or systematic error) which is associated with the class of hypotheses we are considering
- The last term is the *noise*, which is due to the problem at hand, and cannot be avoided

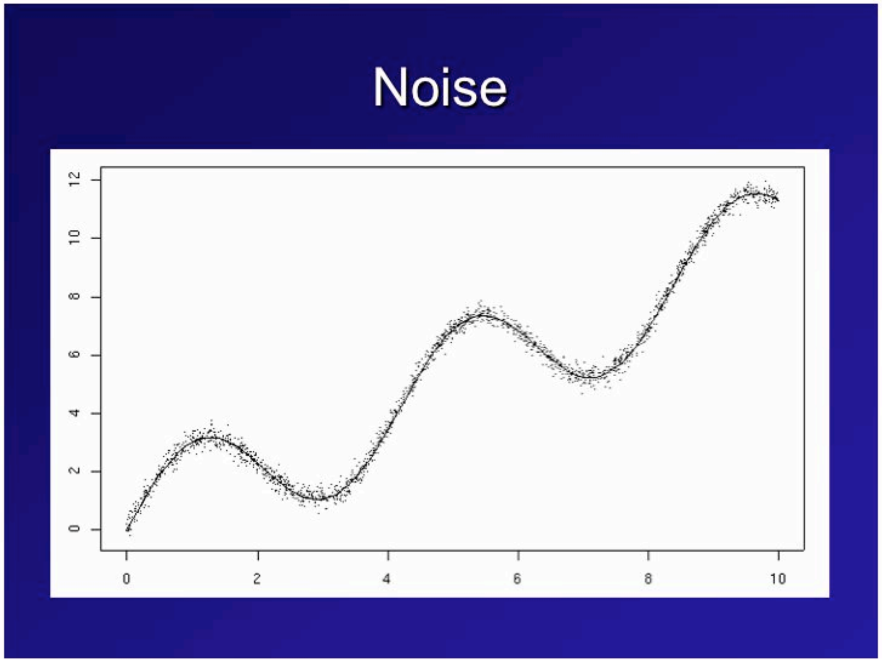
Example revisited: Bias



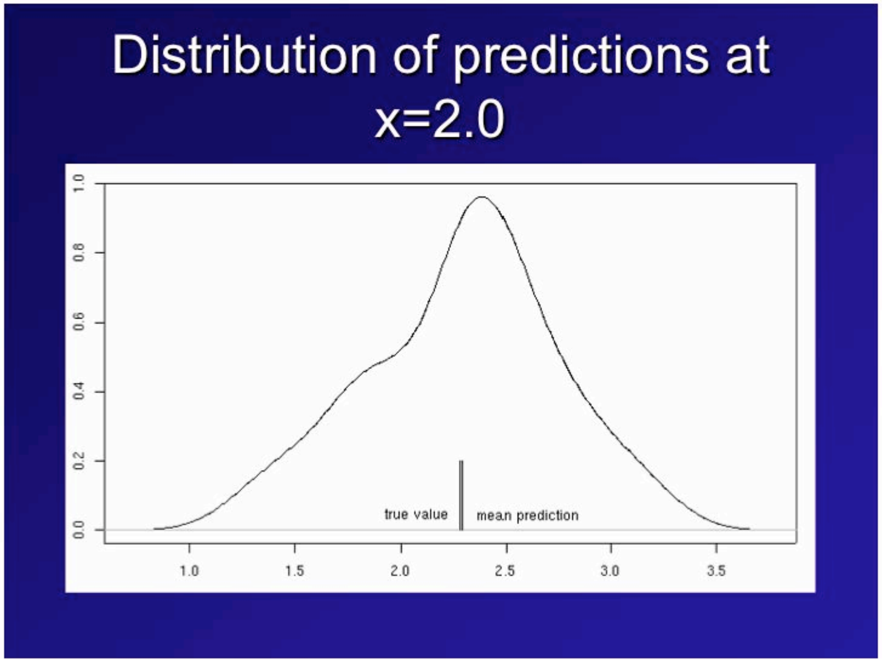
Example revisited: Variance



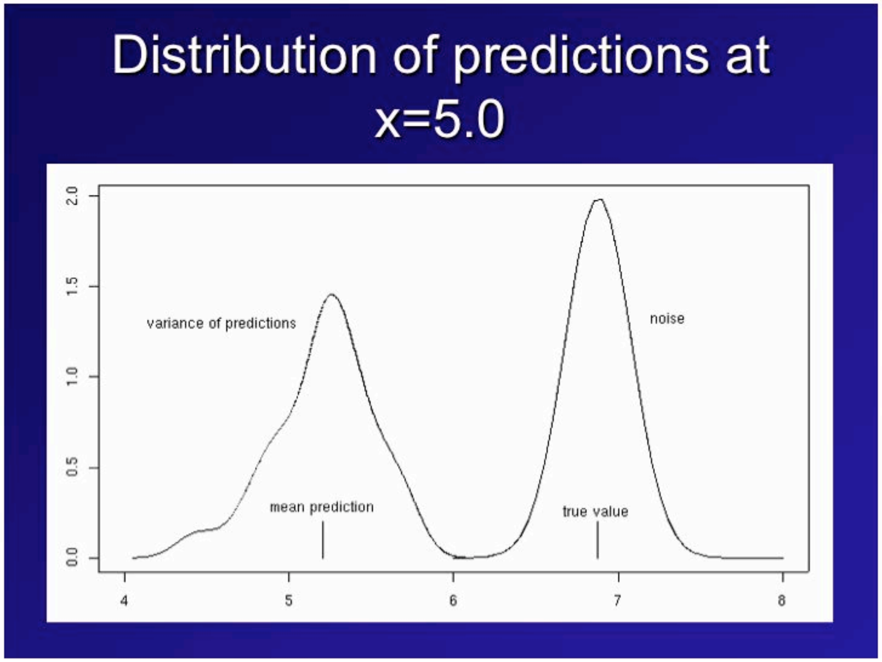
Example revisited: Noise



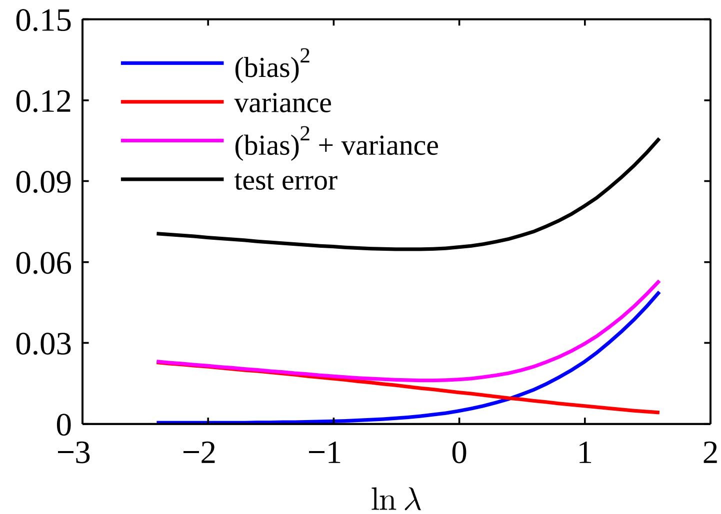
A point with low bias



A point with high bias



Error decomposition



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

Bias-variance trade-off

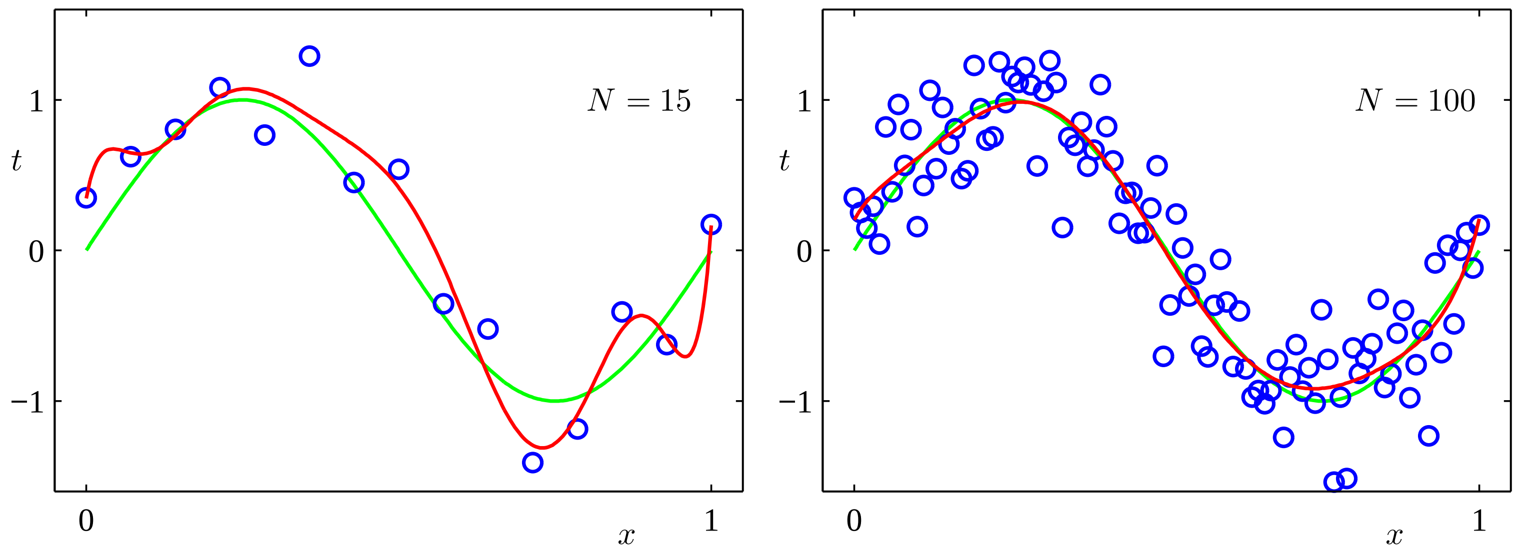
- Consider fitting a small degree vs. a high degree polynomial
- Which one do you expect to have higher bias? Higher variance?

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 you have

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* **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

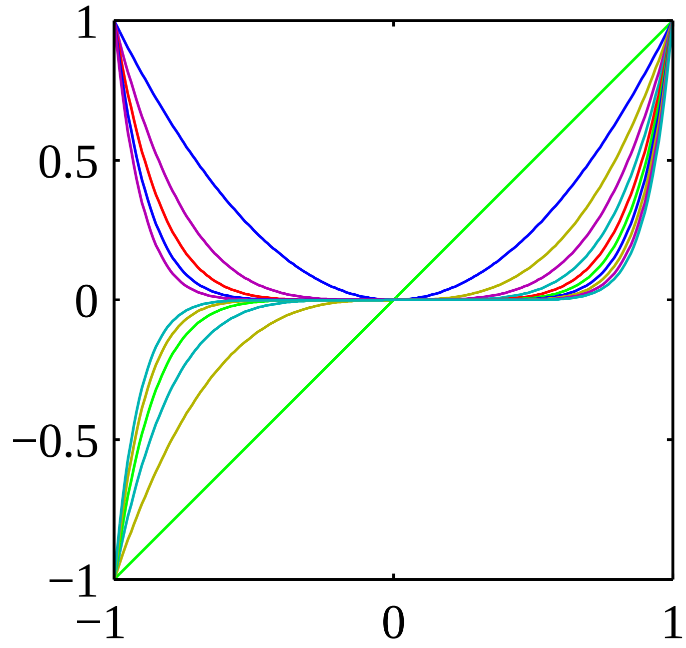
$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{k=0}^{K-1} w_k \phi_k(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$$

  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*

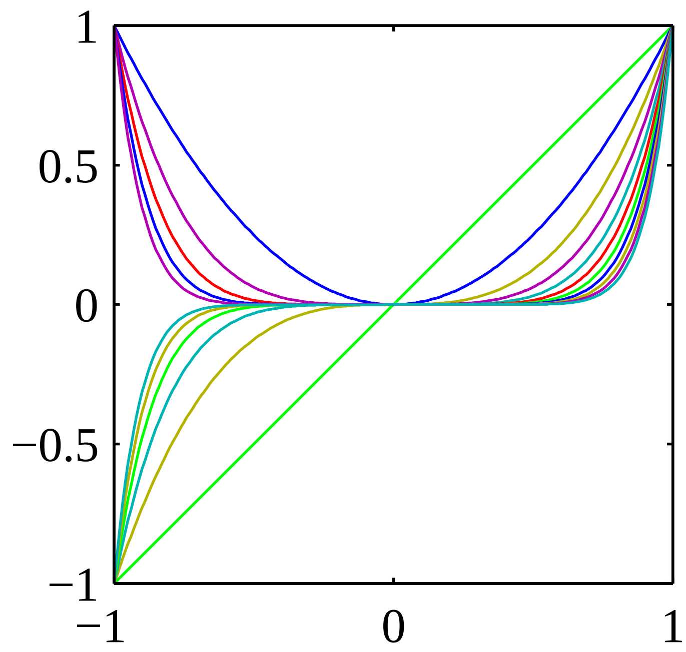
Example basis functions: Polynomials



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

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

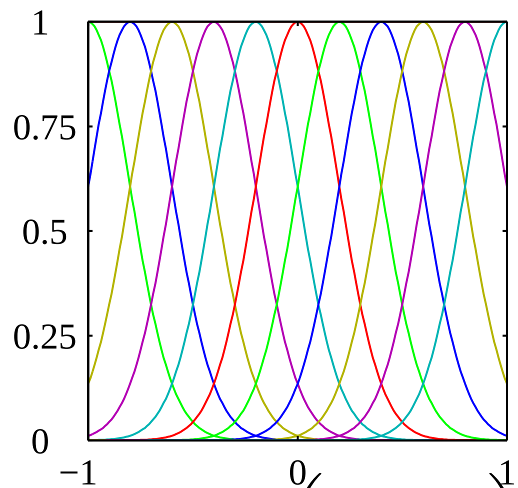
Example basis functions:



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

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

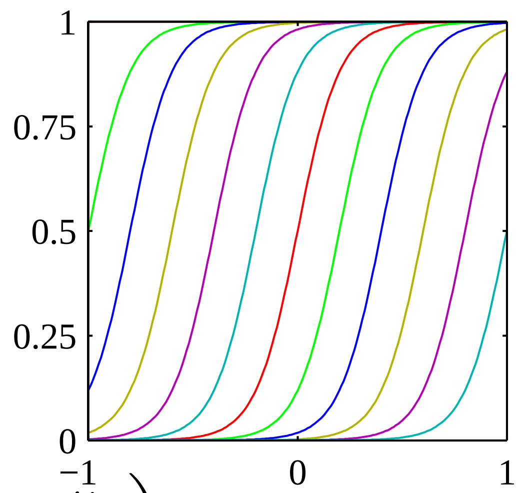
Example basis functions: Gaussians



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

- μ_k controls the position along the x-axis
- s controls the width (activation radius)
- μ_k, s fixed for now (later we discuss adjusting them)
- Usually thought as “local” functions: a small change in x only causes a change in the output of the basis 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 (others will be close to 0 or 1)

Minimizing the mean-squared error

- Recall from last time: we want $\min_{\mathbf{w}} J_D(\mathbf{w})$, where:

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

- Compute the gradient and set it to 0:

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

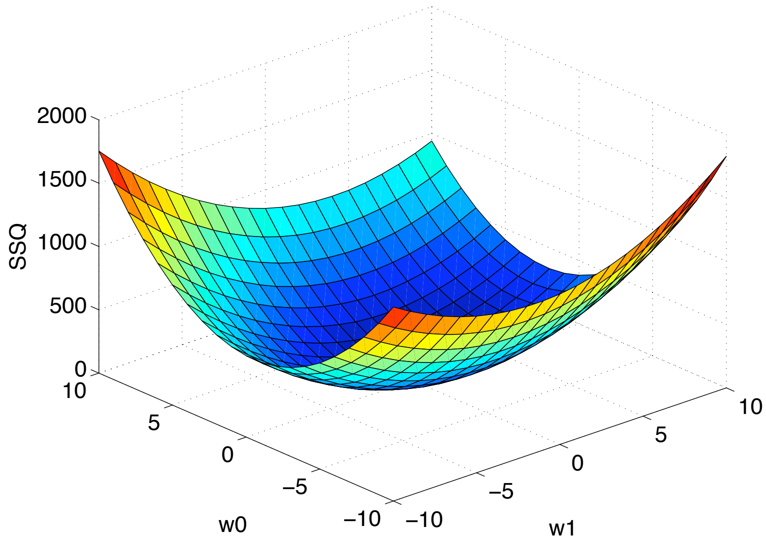
- Solve for $\mathbf{w}$:

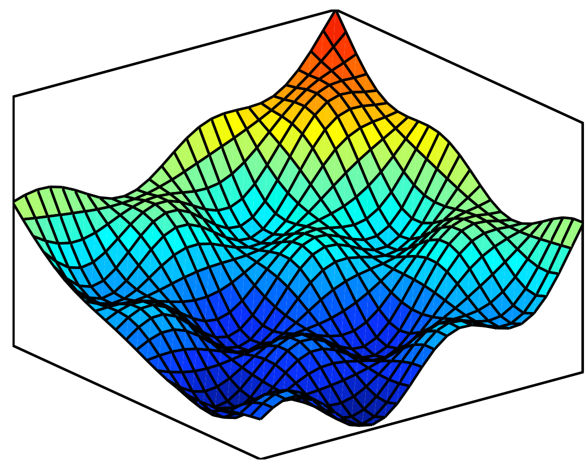
$$\mathbf{w} = (\mathbf{\Phi}^T\mathbf{\Phi})^{-1}\mathbf{\Phi}^T\mathbf{y}$$

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

Gradient descent

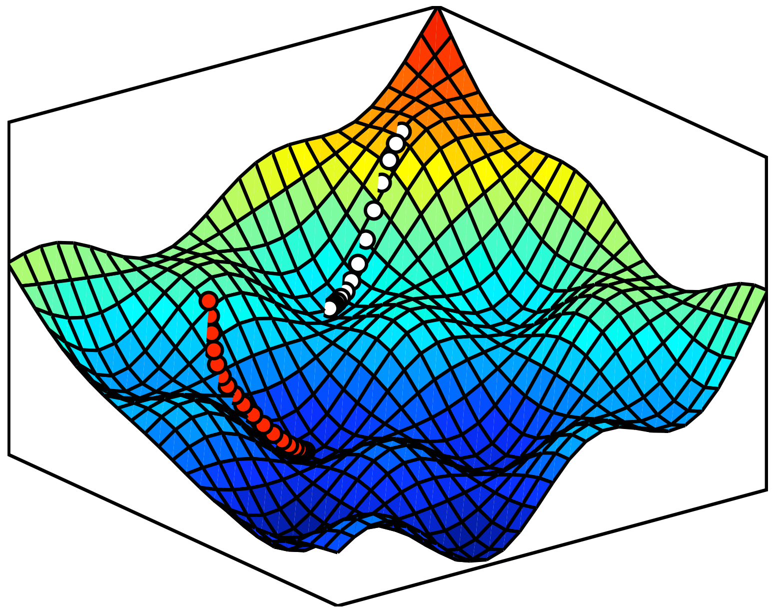
- The gradient of J at a point $\langle w_0, w_1, \ldots, w_k \rangle$ can be thought of as a vector indicating which way is "uphill".





- 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 $\mathbf{w}^{\mathbf{i}}$ 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 $\mathbf{w}^{\mathbf{i}}$ 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 $\nabla J(\mathbf{w}^{\mathbf{i}})$; 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 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 (least-mean-square or 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. $\ldots$

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 are impossible)

Annoyances:

- Speed of convergence depends on the learning rate schedule
- In non-linear case, randomizing the initial parameter vector is crucial

Another algorithm for optimization

- Recall Newton's method for finding the zero of a function $g : \mathbb{R} \rightarrow \mathbb{R}$
- At point w^i, approximate the function by a straight line (its tangent)
- Solve the linear equation for where the tangent equals 0, and move the parameter to this point:

$$w^{i+1} = w^i - \frac{g(w^i)}{g'(w^i)}$$

Application to machine learning

- Suppose for simplicity that the error function J has only one parameter
- We want to optimize J, so we can apply Newton's method to find the zeros of $J' = \frac{d}{dw}J$
- We obtain the iteration:

$$w^{i+1} = w^i - \frac{J'(w^i)}{J''(w^i)}$$

- Note that there is *no step size parameter*!
- This is a *second-order method*, because it requires computing the second derivative
- But, if our error function is quadratic, this will find the global optimum in one step!

Second-order methods: Multivariate setting

- If we have an error function J that depends on many variables, we can compute the *Hessian matrix*, which contains the second-order derivatives of J:

$$H_{ij} = \frac{\partial^2 J}{\partial w_i \partial w_j}$$

- The inverse of the Hessian gives the "optimal" learning rates
- The weights are updated as:

$$\mathbf{w} \leftarrow \mathbf{w} - H^{-1} \nabla_{\mathbf{w}} J$$

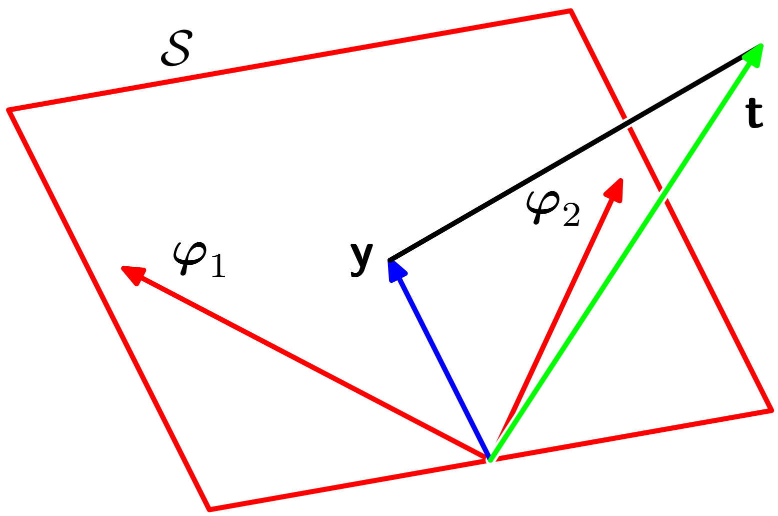
- This is also called Newton-Raphson method

Which method is better?

- Newton's method usually requires significantly fewer iterations than gradient descent
- Computing the Hessian requires a batch of data, so there is no natural on-line algorithm
- Inverting the Hessian explicitly is expensive, but there is very cute trick for computing the product we need in linear time (Schraudolph, 1996)

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:

$$y_i = h_{\mathbf{w}}(\mathbf{x_i}) + e_i$$

  where e_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:

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

Choosing hypotheses

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

What is the most probable hypothesis given the training data?

Maximum a posteriori (MAP) hypothesis h_{MAP}:

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

This is the Bayesian answer (more detail next time)

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 $P(D|h)$:

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

The $\log$ trick

- We want to maximize:

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

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

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

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

Maximum likelihood for regression

- Adopt the assumption that:

$$y_i = h_{\mathbf{w}}(\mathbf{x_i}) + e_i,$$

where e_i are normally distributed with mean 0 and variance σ

- The best hypothesis maximizes the likelihood of $y_i - h_{\mathbf{w}}(\mathbf{x}_i) = e_i$
- 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 e_i are from a Gaussian distribution

Applying the $\log$ trick

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

$$= \sum_{i=1}^{m} \log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) - \sum_{i=1}^{m} \frac{1}{2} \frac{(y_i - h_{\mathbf{w}}(\mathbf{x_i}))^2}{\sigma^2}$$

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!

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:

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

- 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 homework)
- 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
- One can view this as making complex hypotheses a priori less likely (though there are some subtleties)

Regularization for linear models

- A squared penalty on the weights would make the math work nicely in our case:
$$\frac{1}{2}(\mathbf{\Phi}\mathbf{w}-\mathbf{y})^T(\mathbf{\Phi}\mathbf{w}-\mathbf{y})+\frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$
- This regularization term is also known as *weight decay* in neural networks
- Optimal solution:
$$\mathbf{w}=(\mathbf{\Phi}^T\mathbf{\Phi}+\lambda I)^{-1}\mathbf{\Phi}\mathbf{y}$$