Introduction to Machine Learning

Doina Precup
McGill University
Email: dprecup@cs.mcgill.ca

Deep Learning Summer School, Montreal, 2016

Outline

- Types of machine learning problems
- Linear approximators
- Error/objective functions and how to optimize them
- Bias-variance trade-off, overfitting and underfitting
- L2 and L1 regularization for linear estimators
- A Bayesian interpretation of regularization
- Logistic regression

Types of machine learning problems

Based on the information available:

- Supervised learning
- Reinforcement learning
- Unsupervised learning

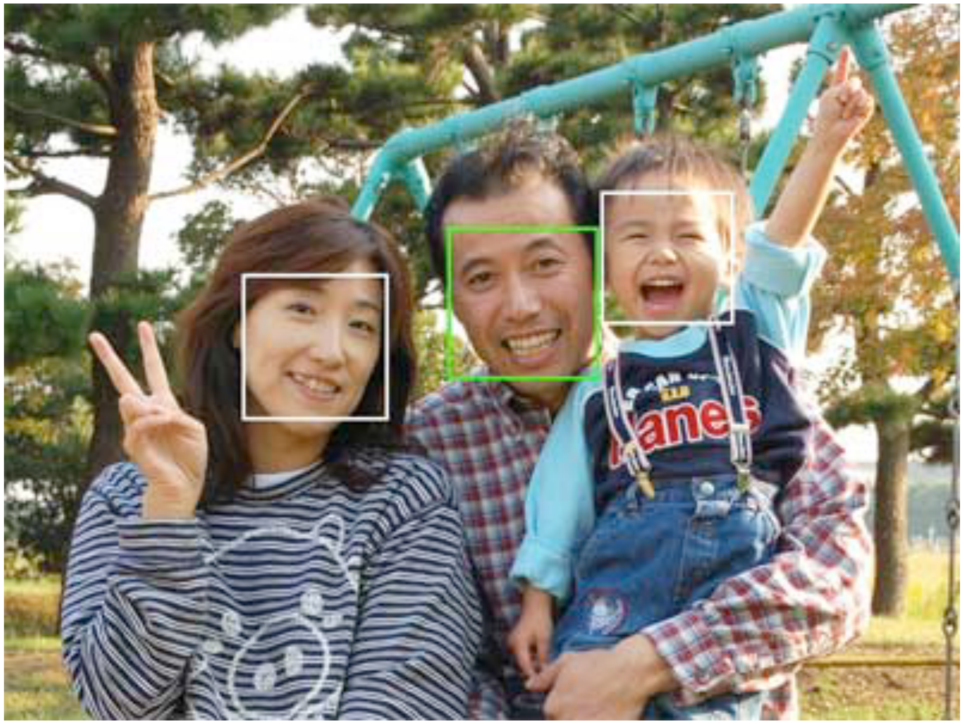
Supervised learning

- Training experience: a set of *labeled examples* of the form

  $$\langle x_1\, x_2 \ldots x_n, y \rangle,$$

  where x_j are values for *input variables* and y is the *output*
- This implies the existence of a "teacher" who knows the right answers
- What to learn: A *function* $f : X_1 \times X_2 \times \cdots \times X_n \rightarrow Y$, which maps the input variables into the output domain
- Goal: *minimize the error (loss) function*
  - Ideally, we would like to minimize error on *all possible instances*
  - But we only have access to a limited set of data...

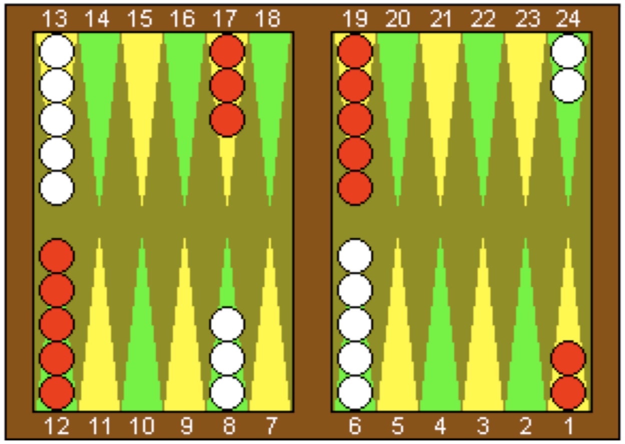
Example: Face detection and recognition

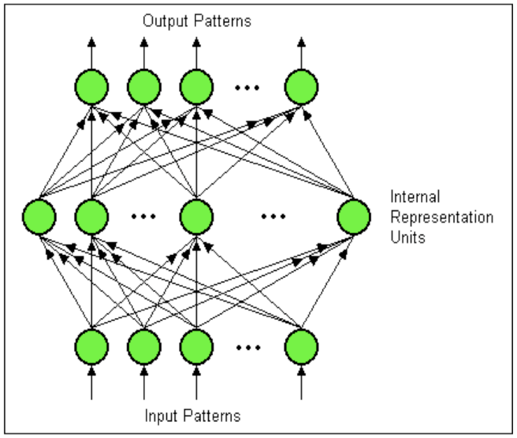


Reinforcement learning

- Training experience: interaction with an environment; the agent receives a numerical reward signal
- E.g., a trading agent in a market; the reward signal is the profit
- What to learn: a way of behaving that is very rewarding in the long run
- Goal: estimate and maximize the long-term cumulative reward

Example: TD-Gammon (Tesauro, 1990-1995)



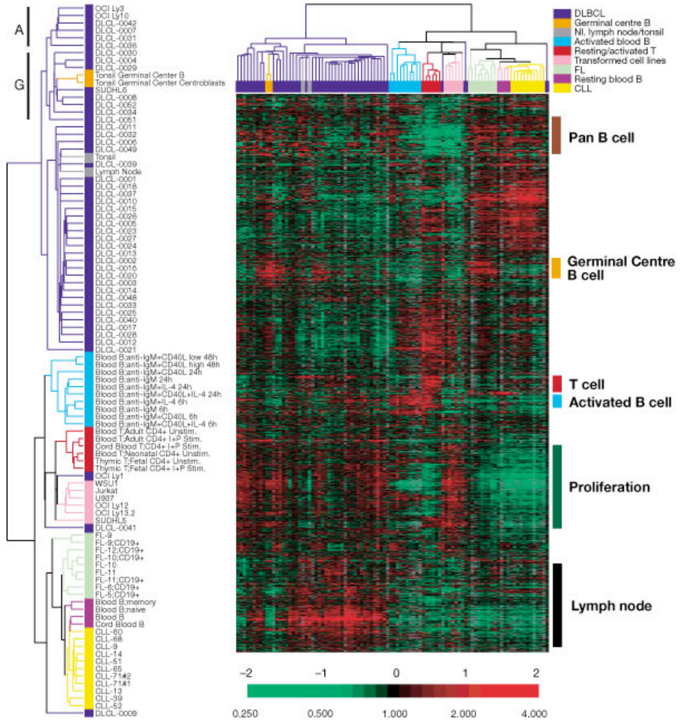


- Early predecessor of AlphaGo
- Learning from self-play, using TD-learning
- Became the best player in the world
- Discovered new ways of opening not used by people before

Unsupervised learning

- Training experience: unlabelled data
- What to learn: interesting associations in the data
- E.g., clustering, dimensionality reduction
- Often there is no single correct answer

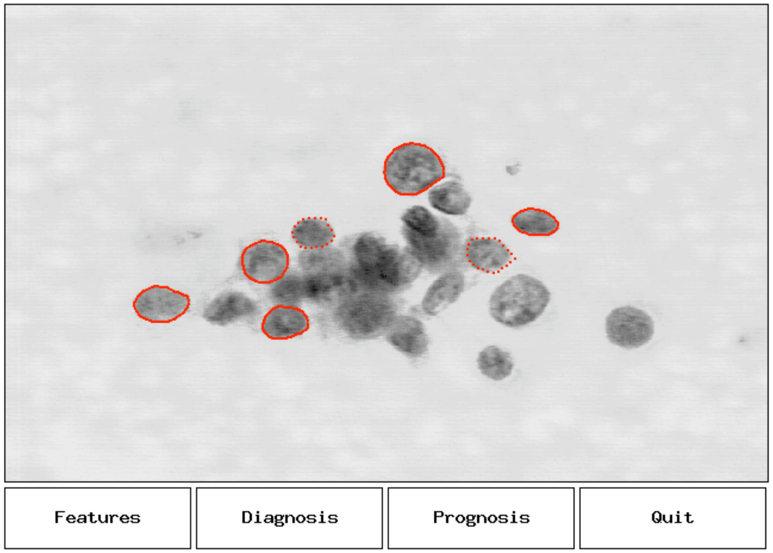
Example: Oncology (Alizadeh et al.)



- Activity levels of all (≈ 25,000) genes were measured in lymphoma patients
- Cluster analysis determined three different subtypes (where only two were known before), having different clinical outcomes

Example: A data set

Cell Nuclei of Fine Needle Aspirate



- Cell samples were taken from tumors in breast cancer patients before surgery, and imaged
- Tumors were excised
- Patients were followed to determine whether or not the cancer recurred, and how long until recurrence or disease free

Data (continued)

- Thirty real-valued variables per tumor.
- Two variables that can be predicted:
  - Outcome (R=recurrence, N=non-recurrence)
  - Time (until recurrence, for R, time healthy, for N).

tumor size	texture	perimeter	. . .	outcome	time
18.02	27.6	117.5		N	31
17.99	10.38	122.8		N	61
20.29	14.34	135.1		R	27
. . .					

Terminology

tumor size	texture	perimeter	. . .	outcome	time
18.02	27.6	117.5		N	31
17.99	10.38	122.8		N	61
20.29	14.34	135.1		R	27
. . .					

- Columns are called *input variables* or *features* or *attributes*
- The outcome and time (which we are trying to predict) are called *output variables* or *targets*
- A row in the table is called *training example* or *instance*
- The whole table is called *(training) data set*.
- The problem of predicting the recurrence is called *(binary) classification*
- The problem of predicting the time is called *regression*

More formally

tumor size	texture	perimeter	. . .	outcome	time
18.02	27.6	117.5		N	31
17.99	10.38	122.8		N	61
20.29	14.34	135.1		R	27
. . .					

- A training example i has the form: $\langle x_{i,1}, \ldots x_{i,n}, y_i \rangle$ where n is the number of attributes (30 in our case).
- We will use the notation $\mathbf{x}_i$ to denote the column vector with elements $x_{i,1}, \ldots x_{i,n}$.
- The training set D consists of m training examples
- We denote the $m \times n$ matrix of attributes by $\mathbf{X}$ and the size-m column vector of outputs from the data set by $\mathbf{y}$.

Supervised learning problem

- Let $\mathcal{X}$ denote the space of input values
- Let $\mathcal{Y}$ denote the space of output values
- Given a data set $D \subset \mathcal{X} \times \mathcal{Y}$, find a function:

$$h : \mathcal{X} \to \mathcal{Y}$$

  such that $h(\mathbf{x})$ is a *"good predictor"* for the value of y.
- h is called a *hypothesis*
- Problems are categorized by the type of output domain
  - If $\mathcal{Y} = \mathbb{R}$, this problem is called *regression*
  - If $\mathcal{Y}$ is a categorical variable (i.e., part of a finite discrete set), the problem is called *classification*
  - In general, $\mathcal{Y}$ could be a lot more complex (graph, tree, etc), which is called *structured prediction*

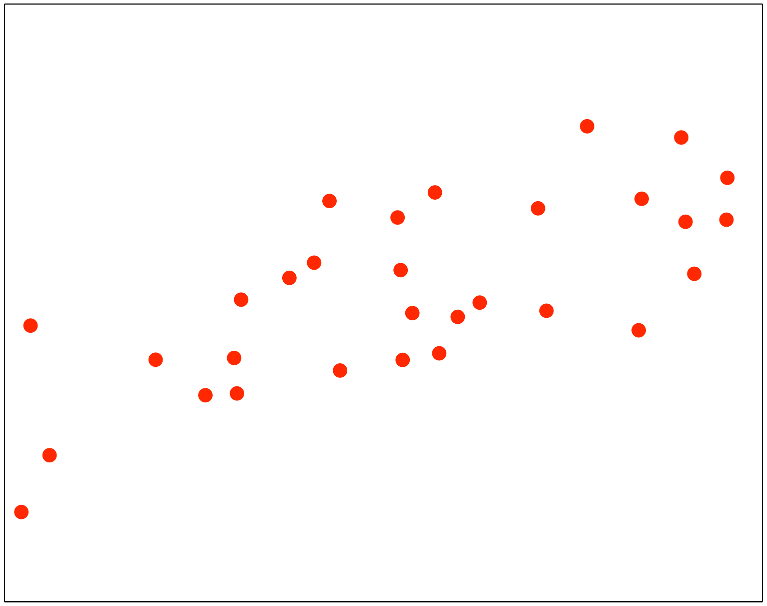
Steps to solving a supervised learning problem

1. Decide what the input-output pairs are.
2. Decide how to encode inputs and outputs.

   This defines the input space $\mathcal{X}$, and the output space $\mathcal{Y}$.

   (We will discuss this in detail later)
3. Choose a class of hypotheses/representations $\mathcal{H}$.
4. ...

Example: What hypothesis class should we pick?



x	y
0.86	2.49
0.09	0.83
-0.85	-0.25
0.87	3.10
-0.44	0.87
-0.43	0.02
-1.10	-0.12
0.40	1.81
-0.96	-0.83
0.17	0.43

Linear hypothesis

- Suppose y was a linear function of $\mathbf{x}$:

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 (+ \cdots)$$

- w_i are called *parameters* or *weights*
- To simplify notation, we can add an attribute $x_0 = 1$ to the other n attributes (also called *bias term* or *intercept term*):

$$h_{\mathbf{w}}(\mathbf{x}) = \sum_{i=0}^{n} w_i x_i = \mathbf{w}^T \mathbf{x}$$

where $\mathbf{w}$ and $\mathbf{x}$ are vectors of size $n+1$.

How should we pick $\mathbf{w}$*?*

Error minimization!

- Intuitively, $\mathbf{w}$ should make the predictions of $h_{\mathbf{w}}$ close to the true values y on the data we have
- Hence, we will define an *error function* or *cost function* to measure how much our prediction differs from the "true" answer
- We will pick $\mathbf{w}$ such that the error function is minimized

How should we choose the error function?

Least mean squares (LMS)

- Main idea: try to make $h_{\mathbf{w}}(\mathbf{x})$ close to y on the examples in the training set
- We define a *sum-of-squares* error function

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

(the $1/2$ is just for convenience)

- We will choose $\mathbf{w}$ such as to minimize $J(\mathbf{w})$

Steps to solving a supervised learning problem

1. Decide what the input-output pairs are.
2. Decide how to encode inputs and outputs.

   This defines the input space $\mathcal{X}$, and the output space $\mathcal{Y}$.
3. Choose a class of hypotheses/representations $\mathcal{H}$.
4. Choose an error function (cost function) to define the best hypothesis
5. Choose an algorithm for searching efficiently through the space of hypotheses.

Notation reminder

- Consider a function $f(u_1, u_2, \ldots, u_n) : \mathbb{R}^n \mapsto \mathbb{R}$ (for us, this will usually be an error function)
- The *partial derivative* w.r.t. u_i is denoted:

$$\frac{\partial}{\partial u_i} f(u_1, u_2, \ldots, u_n) : \mathbb{R}^n \mapsto \mathbb{R}$$

  The partial derivative is the derivative along the u_i axis, keeping all other variables fixed.

- The *gradient* $\nabla f(u_1, u_2, \ldots, u_n) : \mathbb{R}^n \mapsto \mathbb{R}^n$ is a function which outputs a vector containing the partial derivatives.
  That is:

$$\nabla f = \left\langle \frac{\partial}{\partial u_1} f, \frac{\partial}{\partial u_2} f, \ldots, \frac{\partial}{\partial u_n} f \right\rangle$$

A bit of algebra

$$
\begin{aligned}
\frac{\partial}{\partial w_j} J(\mathbf{w}) &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_{i=1}^{m} (h_{\mathbf{w}}(\mathbf{x}_i) - y_i)^2 \\
&= \frac{1}{2} \cdot 2 \sum_{i=1}^{m} (h_{\mathbf{w}}(\mathbf{x}_i) - y_i) \frac{\partial}{\partial w_j} (h_{\mathbf{w}}(\mathbf{x}_i) - y_i) \\
&= \sum_{i=1}^{m} (h_{\mathbf{w}}(\mathbf{x}_i) - y_i) \frac{\partial}{\partial w_j} \left(\sum_{l=0}^{n} w_l x_{i,l} - y_i \right) \\
&= \sum_{i=1}^{m} (h_{\mathbf{w}}(\mathbf{x}_i) - y_i) x_{i,j}
\end{aligned}
$$

Setting all these partial derivatives to 0, we get a linear system with $(n+1)$ equations and $(n+1)$ unknowns.

The solution

- Recalling some multivariate calculus:

$$\begin{aligned}\nabla_{\mathbf{w}} J &= \nabla_{\mathbf{w}} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= \nabla_{\mathbf{w}} (\mathbf{w}^T \mathbf{X}^T \mathbf{X}\mathbf{w} - \mathbf{y}^T \mathbf{X}\mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{y} + \mathbf{y}^T \mathbf{y}) \\ &= 2\mathbf{X}^T \mathbf{X}\mathbf{w} - 2\mathbf{X}^T \mathbf{y}\end{aligned}$$

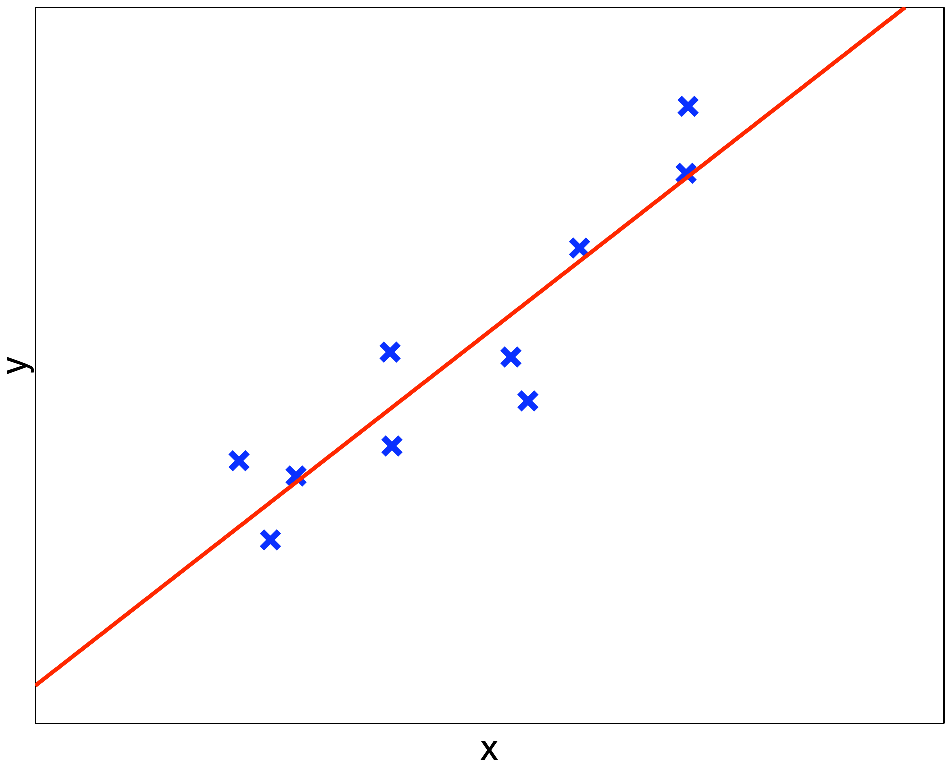
- Setting gradient equal to zero:

$$\begin{aligned}2\mathbf{X}^T \mathbf{X}\mathbf{w} - 2\mathbf{X}^T \mathbf{y} &= 0 \\ \Rightarrow \mathbf{X}^T \mathbf{X}\mathbf{w} &= \mathbf{X}^T \mathbf{y} \\ \Rightarrow \mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}\end{aligned}$$

- The inverse exists if the columns of $\mathbf{X}$ are linearly independent.

Example: Data and best linear hypothesis

$$y = 1.60x + 1.05$$



Linear regression summary

- The optimal solution (minimizing sum-squared-error) can be computed in polynomial time in the size of the data set.
- The solution is $\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$, where $\mathbf{X}$ is the data matrix augmented with a column of ones, and $\mathbf{y}$ is the column vector of target outputs.
- A very rare case in which an analytical, exact solution is possible

Linear function approximation in general

- 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 other methods (e.g. gradient descent - as will be seen later)

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*
- Usually, we will assume that $\phi_0(\mathbf{x}) = 1, \forall \mathbf{x}$, to create a bias term
- The hypothesis can alternatively be written as:

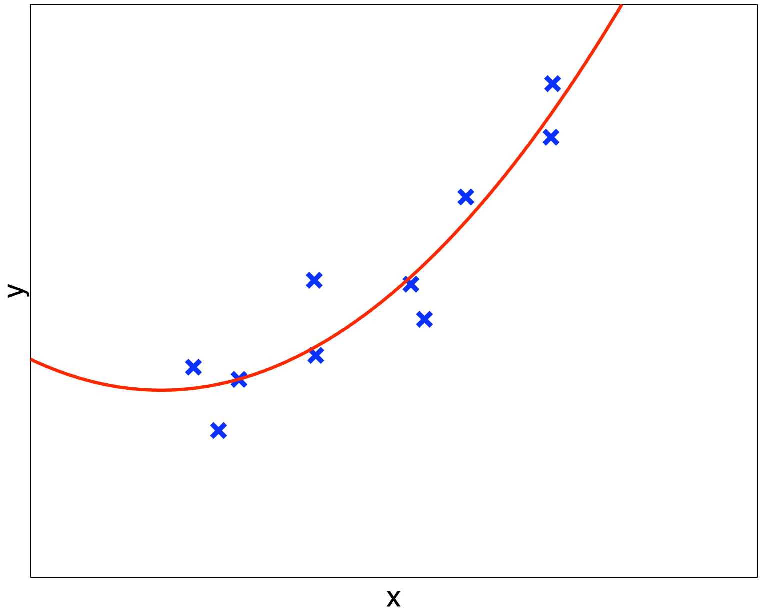
  $$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{\Phi} \mathbf{w}$$

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

Remarks

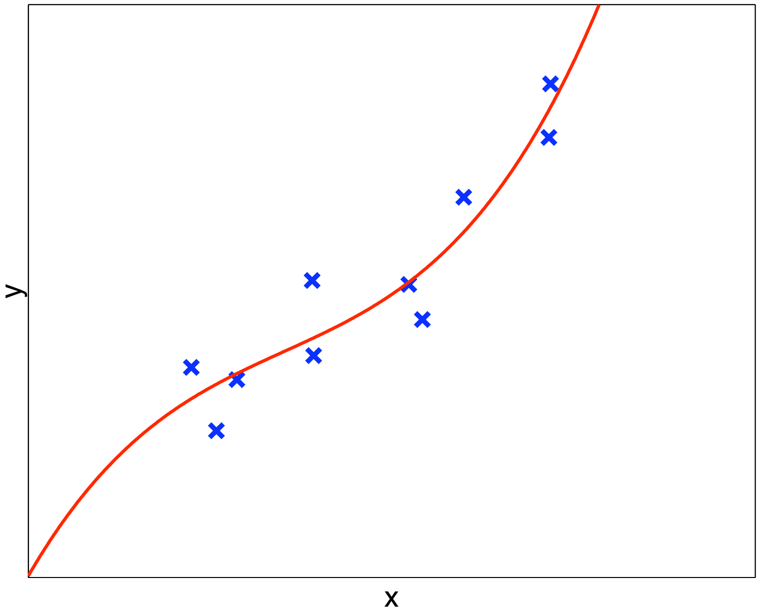
- Linear models are an example of *parametric* models, because we choose a priori a number of parameters that does not depend on the size of the data
- *Non-parametric models* grow with the size of the data
- Eg. Nearest neighbour, locally weighted linear regression
- Deep nets are very large parametric models.

Order-2 fit



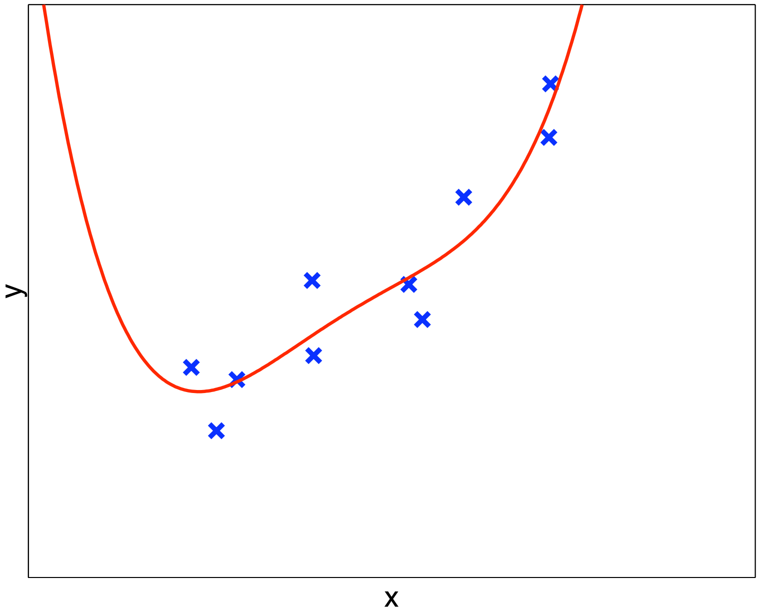
Is this a better fit to the data?

Order-3 fit



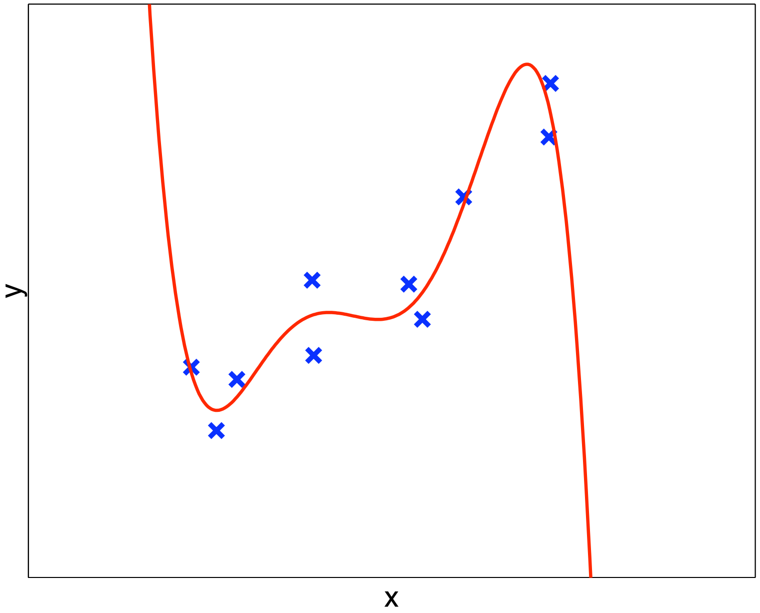
Is this a better fit to the data?

Order-4 fit



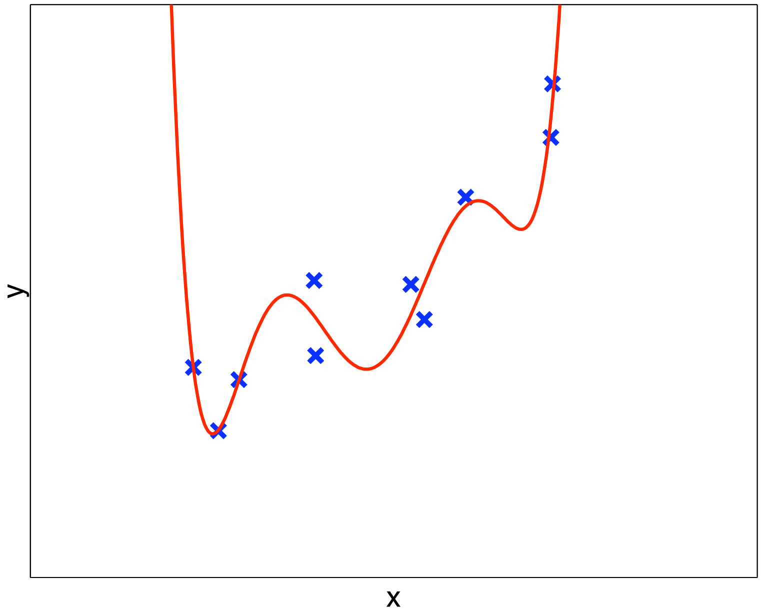
Is this a better fit to the data?

Order-5 fit



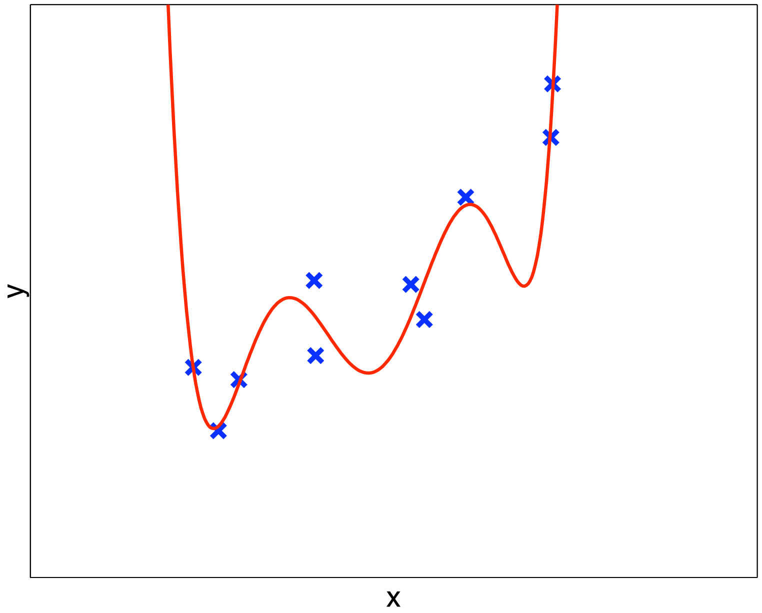
Is this a better fit to the data?

Order-6 fit



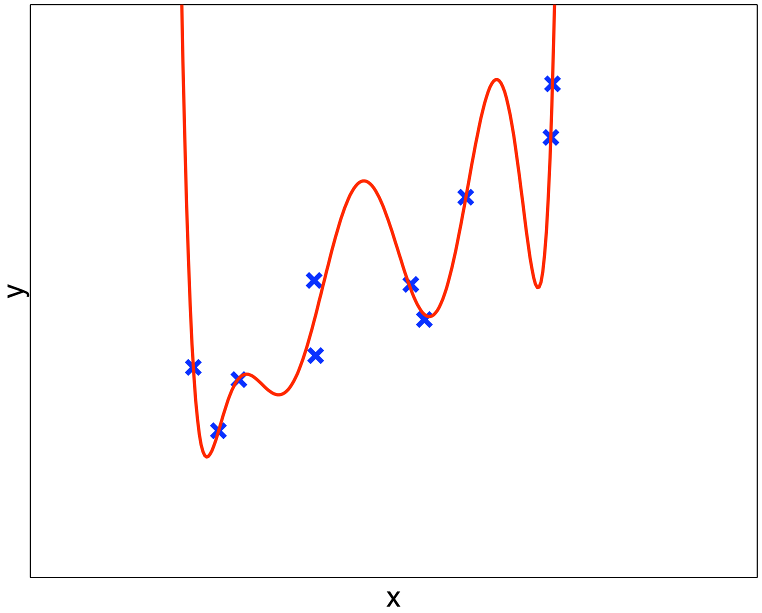
Is this a better fit to the data?

Order-7 fit



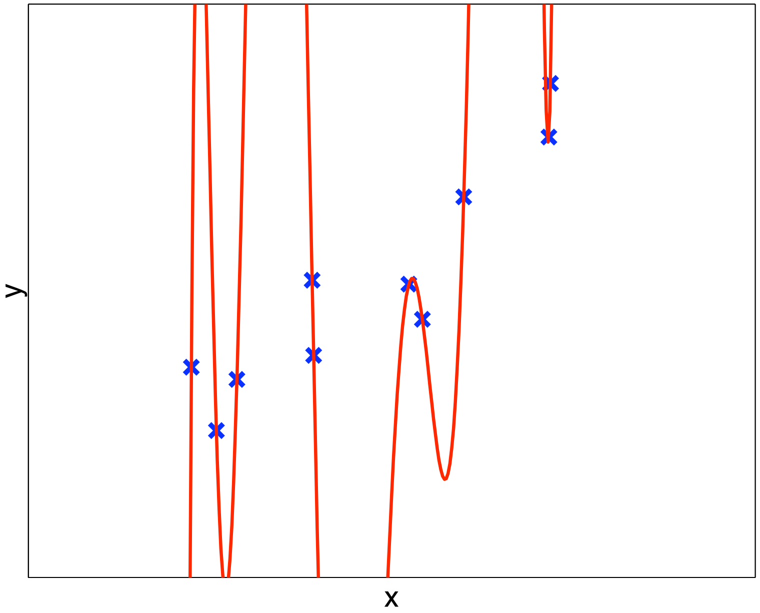
Is this a better fit to the data?

Order-8 fit



Is this a better fit to the data?

Order-9 fit

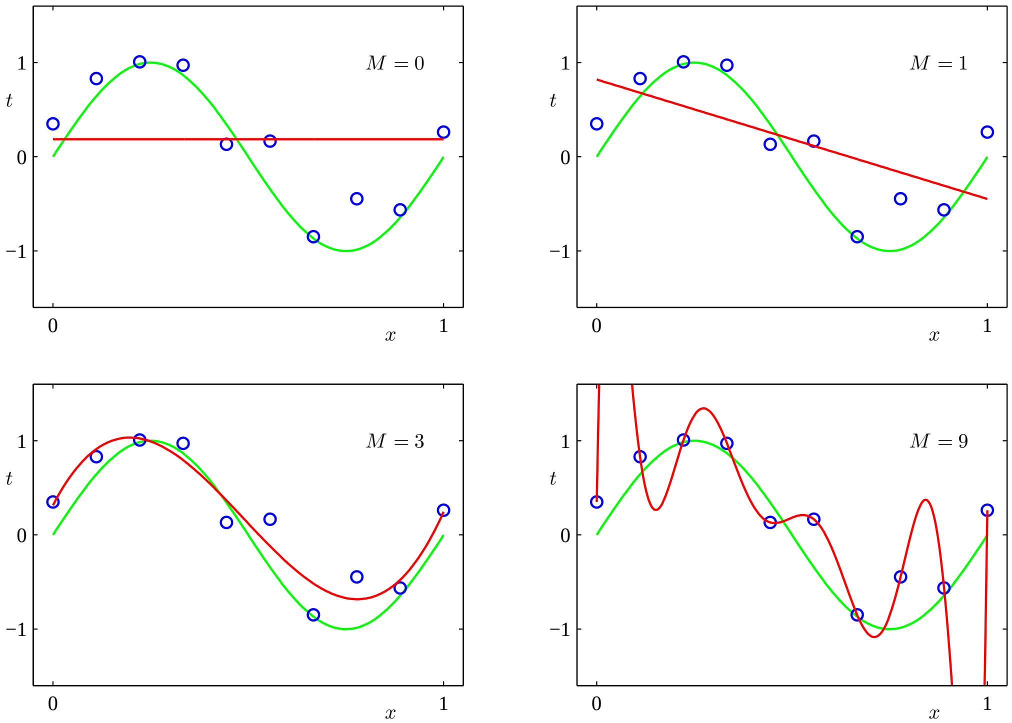


Is this a better fit to the data?

Overfitting

- A general, *HUGELY IMPORTANT* problem for all machine learning algorithms
- We can find a hypothesis that predicts perfectly the training data but *does not generalize* well to new data
- E.g., a lookup table!
- We are seeing an instance here: if we have a lot of parameters, the hypothesis ”memorizes” the data points, but is wild everywhere else.

Overfitting and underfitting

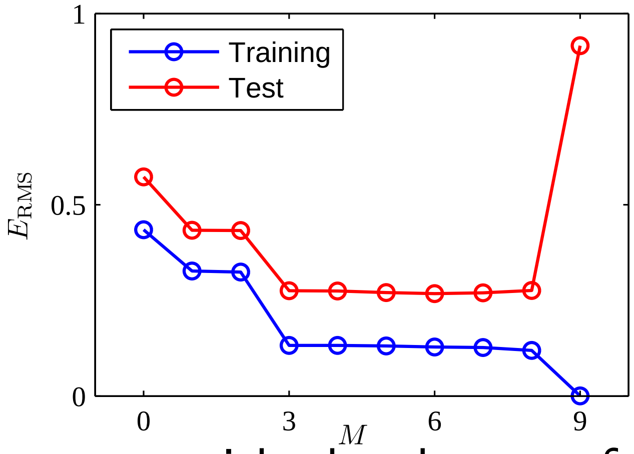


- The higher the degree of the polynomial M, the more degrees of freedom
- Typical overfitting means that error on the training data is very low, but error on new instances is high
- Typical underfitting means that error on the training data is very high (few dof)

Overfitting more formally

- Assume that the data is drawn from some fixed, unknown probability distribution
- Every hypothesis has a "true" error $J^*(h)$, which is the expected error when data is drawn from the distribution.
- Because we do not have all the data, we measure the error on the training set $J_D(h)$
- Suppose we compare hypotheses h_1 and h_2 on the training set, and $J_D(h_1) < J_D(h_2)$
- If h_2 is "truly" better, i.e. $J^*(h_2) < J^*(h_1)$, our algorithm is overfitting.
- We need theoretical and empirical methods to guard against it!

Typical overfitting plot



- The training error decreases with the degree of the polynomial M, 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 (M in our case), using a *validation set of data*
  - Report unbiased results, using a *test set*, untouched during either parameter training or validation

Cross-validation

- A general procedure for estimating the true error of a predictor
- The data is split into two subsets:
  - A *training and validation set* used only to find the right predictor
  - A *test set* used to report the prediction error of the algorithm
- These sets *must be disjoint*!
- The process is repeated several times, and the results are averaged to provide error estimates.

The anatomy of the error of an estimator

- 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
- 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 (hypotheses linear in the parameters) for some target functions f we will have a *systematic prediction error*
- Even if f were truly from the hypothesis class we picked, 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

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:

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

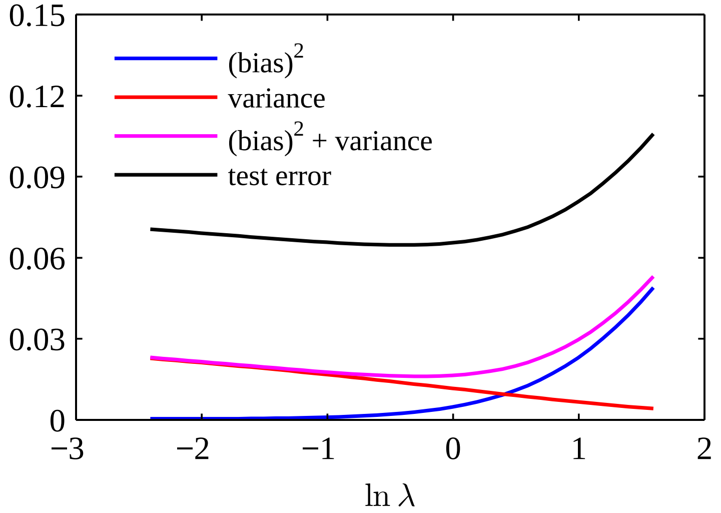
Bias-variance decomposition (2)

- Putting everything together, we have:

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

- 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

Error decomposition



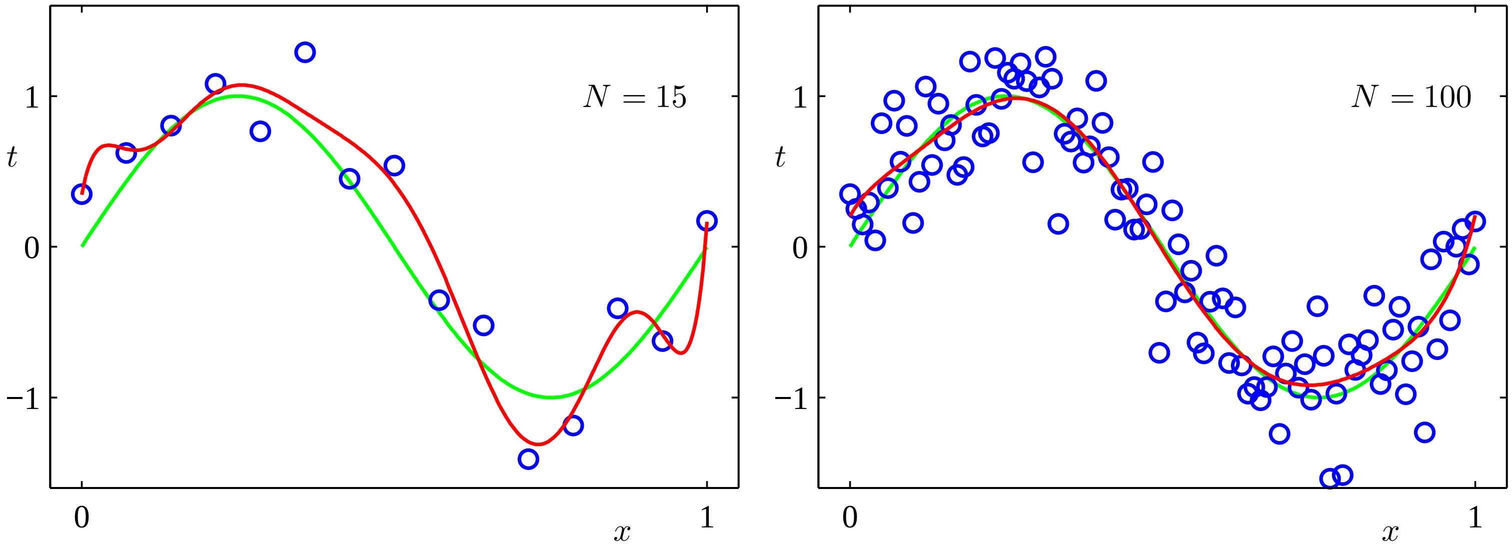
- The bias-variance sum approximates well the test error over a set of 1000 points
- x-axis measures the hypothesis complexity (decreasing left-to-right)
- Simple hypotheses usually have high bias (bias will be high at many points, so it will likely be high for many possible input distributions)
- Complex 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
- MLE estimation is typically unbiased, but has high variance
- Bayesian estimation is biased, but typically has lower variance
- 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
- Making the trade-off has to depend on the amount of data available to fit the parameters (data usually mitigates the variance problem)

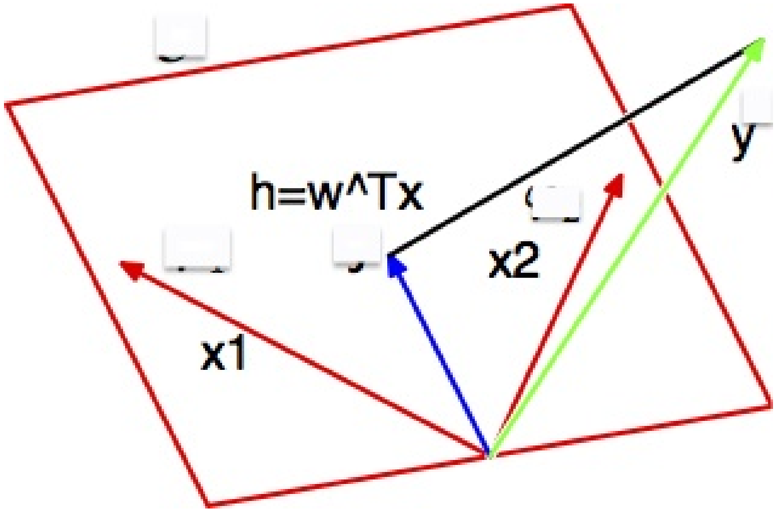
More on overfitting

- 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



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



- 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}:

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

  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

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

- 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*:

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

- Standard assumption: the training examples are *independently identically distributed (i.i.d.)*
- This alows 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

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:

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

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

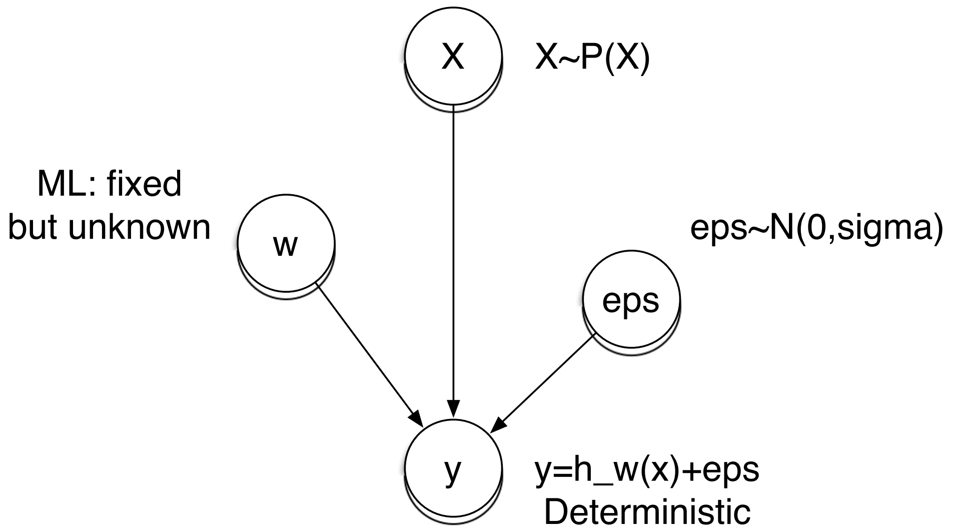
Maximum likelihood hypothesis for least-squares estimators

- 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:

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

- 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
- In practice, different loss functions are used depending on the noise assumption

A graphical representation for the data generation process



- Circles represent (random) variables)
- Arrows represent dependencies between variables
- Some variables are observed, others need to be inferred because they are hidden (latent)
- New assumptions can be incorporated by making the model more complicated

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

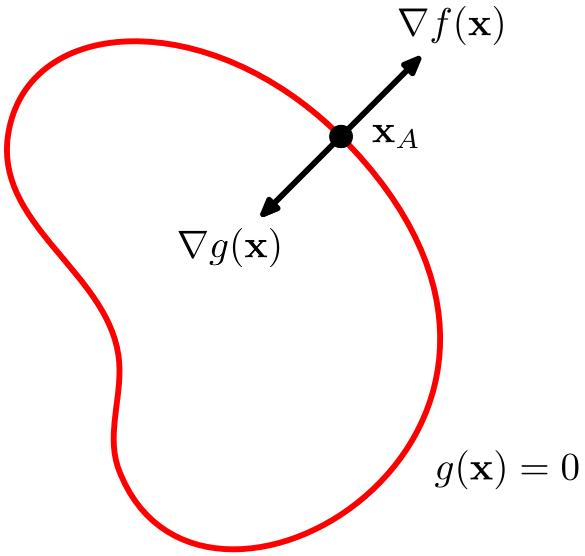
$$\arg\min_{\mathbf{w}} \frac{1}{2}(\mathbf{\Phi w} - \mathbf{y})^T(\mathbf{\Phi w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^T\mathbf{y}$$

- 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
- This is also called *ridge regression*, and it is a special case of Tikhonov regularization (more on that later)
- 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.

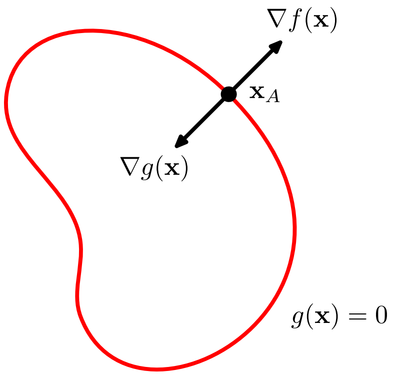
Detour: Constrained optimization

Suppose we want to find

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



Detour: Lagrange multipliers

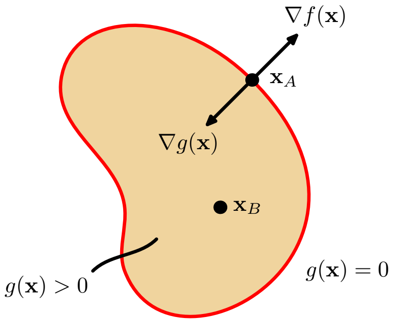


- ∇g has to be orthogonal to the constraint surface (red curve)
- At the optimum, ∇f and ∇g have to be parallel (in same or opposite direction)
- Hence, there must exist some $\lambda \in \mathbb{R}$ such that $\nabla f + \lambda \nabla g = 0$
- *Lagrangian function*: $L(\mathbf{x}, \lambda) = f(\mathbf{x}) + \lambda g(\mathbf{x})$
  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

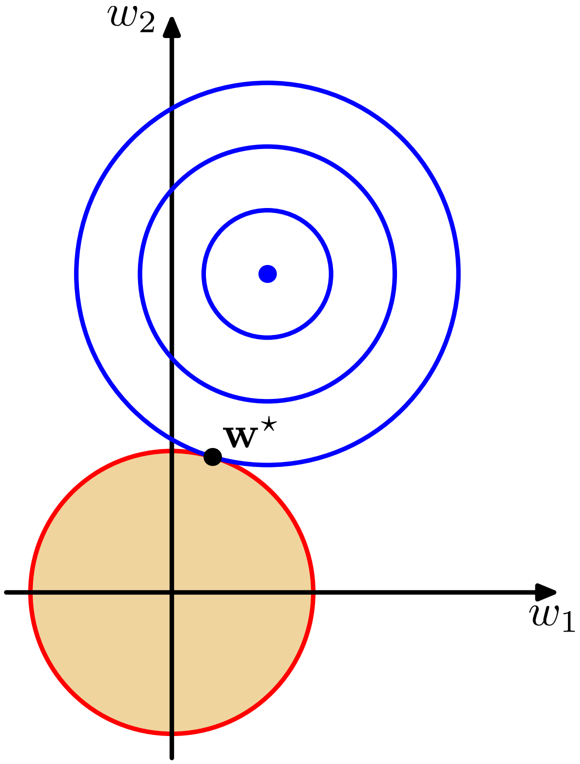
$$\begin{aligned}\min_{\mathbf{w}} J_D(\mathbf{w}) &= \min_{\mathbf{w}} (\mathbf{\Phi w} - \mathbf{y})^T(\mathbf{\Phi w} - \mathbf{y}) \\ \text{such that } \mathbf{w}^T\mathbf{w} &\leq \eta\end{aligned}$$

- 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_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)

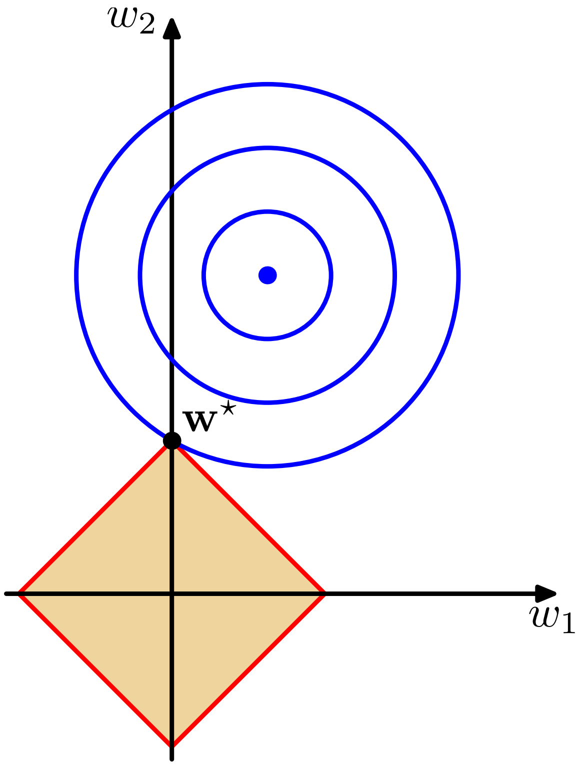
Solving L_1 regularization

- 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:

$$\begin{aligned}
\min_{w_1, w_2} & \quad \sum_{j=1}^{m} (y_j - w_1 x_1 - w_2 x_2)^2 \\
\text{such that } w_1 + w_2 & \leq \eta \\
w_1 - w_2 & \leq \eta \\
-w_1 + w_2 & \leq \eta \\
-w_1 - w_2 & \leq \eta
\end{aligned}$$

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

Visualizing L_1 regularization

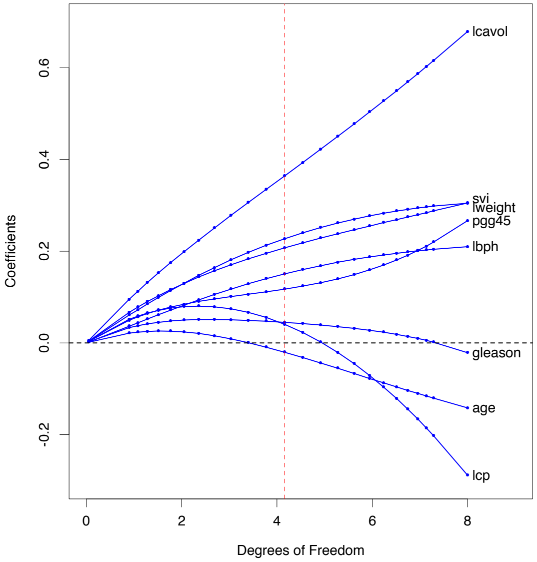


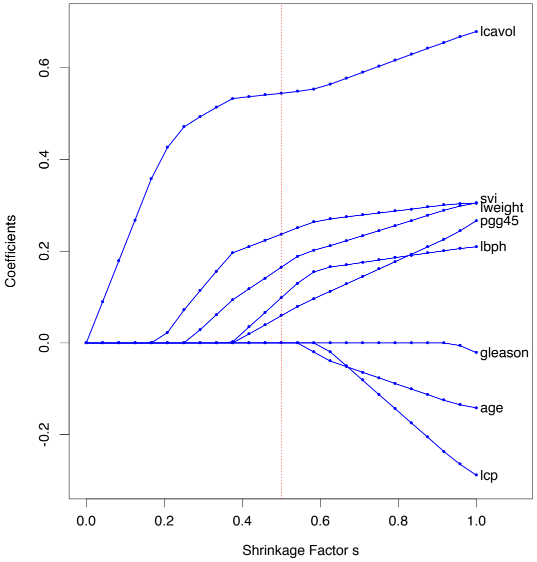
- If λ is big enough, the circle is very likely to intersect the diamond at one of the corners
- This makes L_1 regularization much more likely to make some weights *exactly* 0

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

Example of L1 vs L2 effect



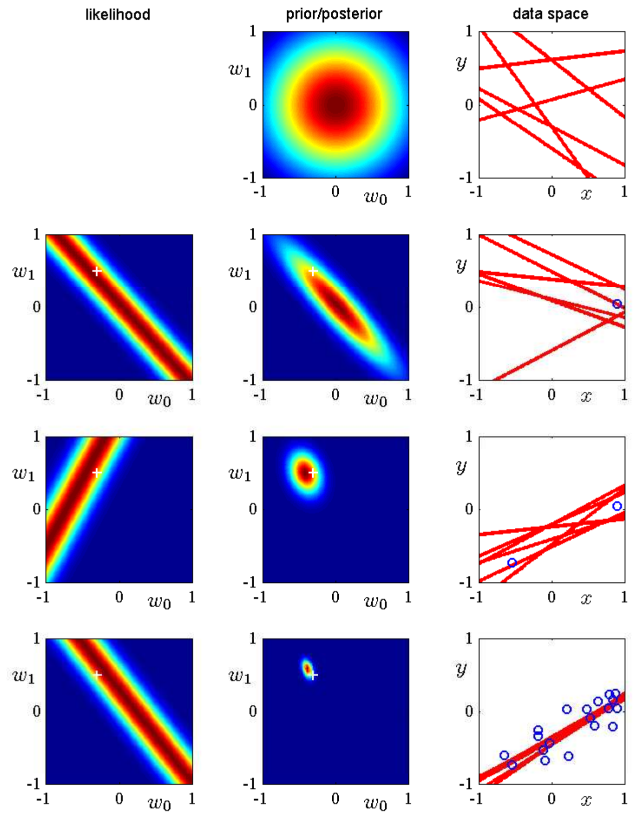


- Note the sparsity in the coefficients induces by L_1
- Lasso is an efficient way of performing the L_1 optimization

Bayesian view of regularization

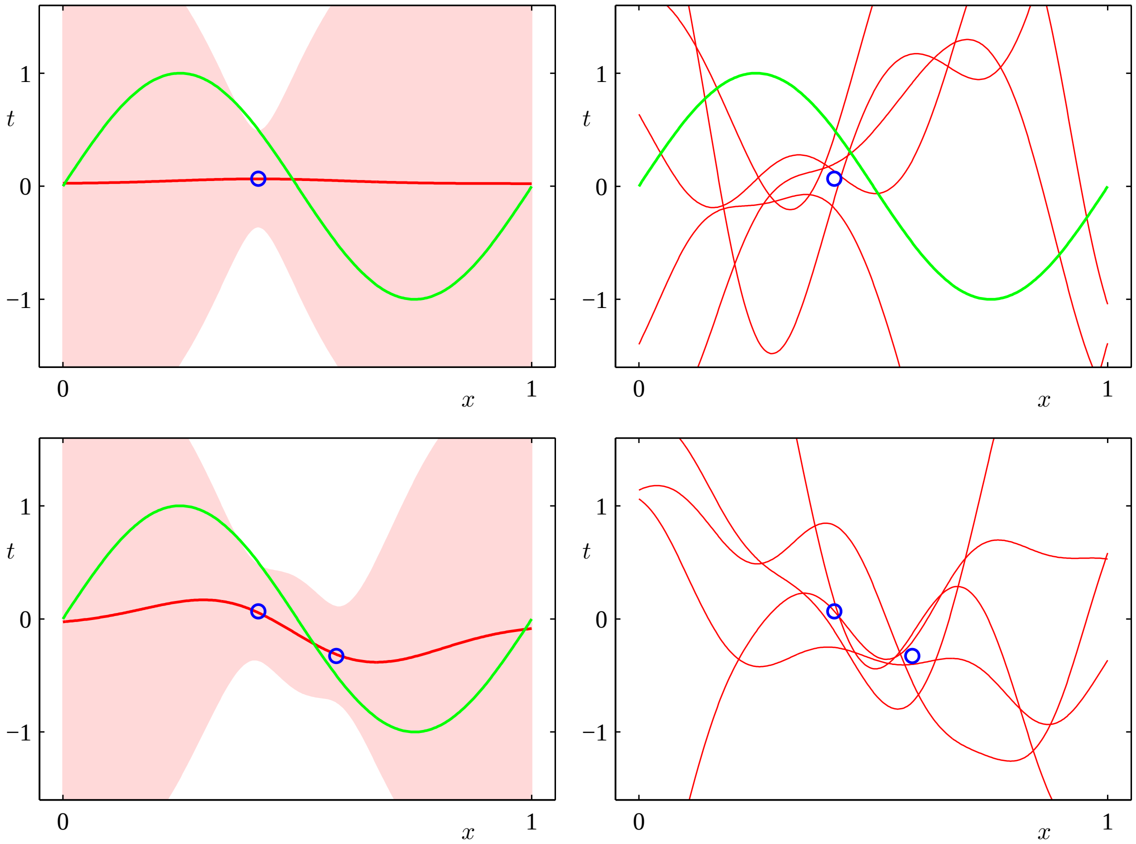
- Start with a *prior distribution* over hypotheses
- As data comes in, compute a *posterior distribution*
- We often work with *conjugate priors*, which means that when combining the prior with the likelihood of the data, one obtains the posterior in the same form as the prior
- Regularization can be 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



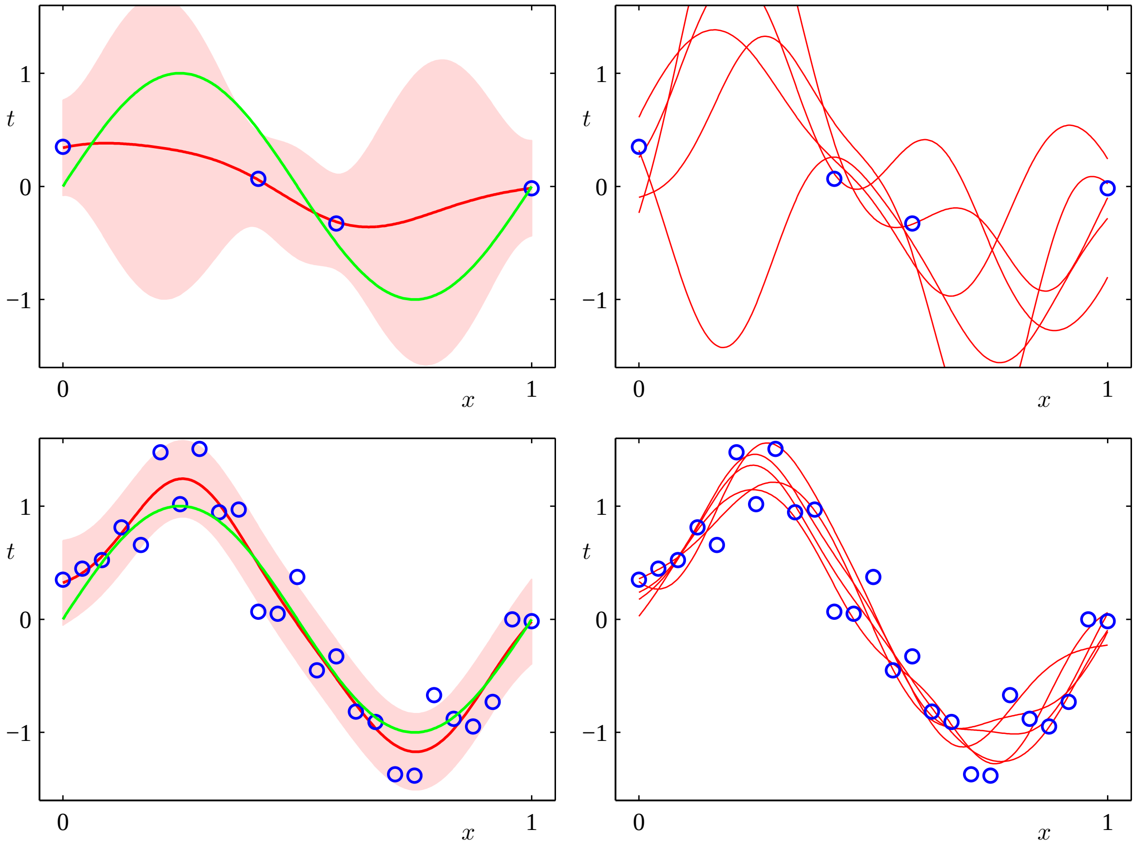
- Prior is round Gaussian
- Posterior will be skewed by the data

What does the Bayesian view give us?



- Circles are data points
- Green is the true function
- Red lines on right are drawn from the posterior distribution

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...

Logistic regression

- Suppose we represent the hypothesis itself as a logistic function of a linear combination of inputs:

  $$h(\mathbf{x}) = \frac{1}{1 + \exp(\mathbf{w}^T \mathbf{x})}$$

  This is also known as a *sigmoid neuron*
- Suppose we interpret $h(\mathbf{x})$ as $P(y = 1|\mathbf{x})$
- Then the log-odds ratio,

  $$\ln \left(\frac{P(y = 1|\mathbf{x})}{P(y = 0|\mathbf{x})} \right) = \mathbf{w}^T \mathbf{x}$$

  which is linear (nice!)
- The optimum weights will maximize the *conditional likelihood* of the outputs, given the inputs.

The cross-entropy error function

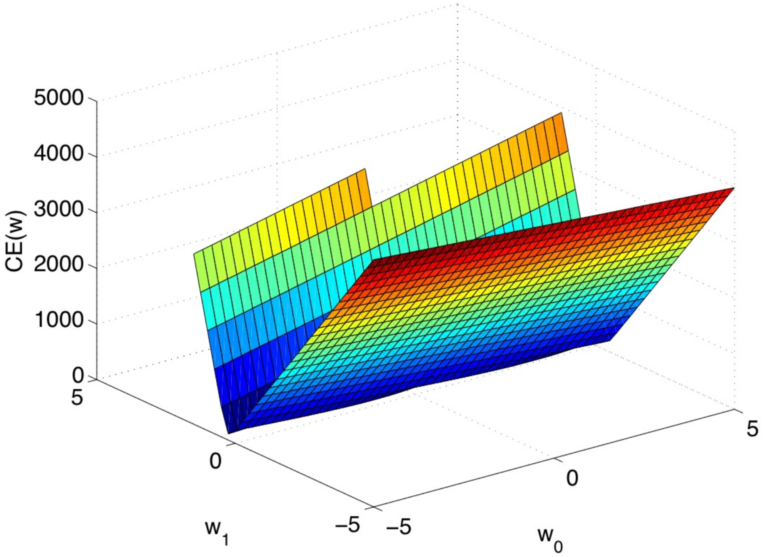
- Suppose we interpret the output of the hypothesis, $h(\mathbf{x}_i)$, as the probability that $y_i = 1$
- Then the log-likelihood of a hypothesis h is:

$$\begin{aligned}\log L(h) &= \sum_{i=1}^{m} \log P(y_i|\mathbf{x}_i, h) = \sum_{i=1}^{m} \begin{cases} \log h(\mathbf{x}_i) & \text{if } y_i = 1 \\ \log(1 - h(\mathbf{x}_i)) & \text{if } y_i = 0 \end{cases} \\ &= \sum_{i=1}^{m} y_i \log h(\mathbf{x}_i) + (1 - y_i) \log(1 - h(\mathbf{x}_i))\end{aligned}$$

- The *cross-entropy error function* is the opposite quantity:

$$J_D(\mathbf{w}) = -\left(\sum_{i=1}^{m} y_i \log h(\mathbf{x}_i) + (1 - y_i) \log(1 - h(\mathbf{x}_i))\right)$$

Cross-entropy error surface for logistic function

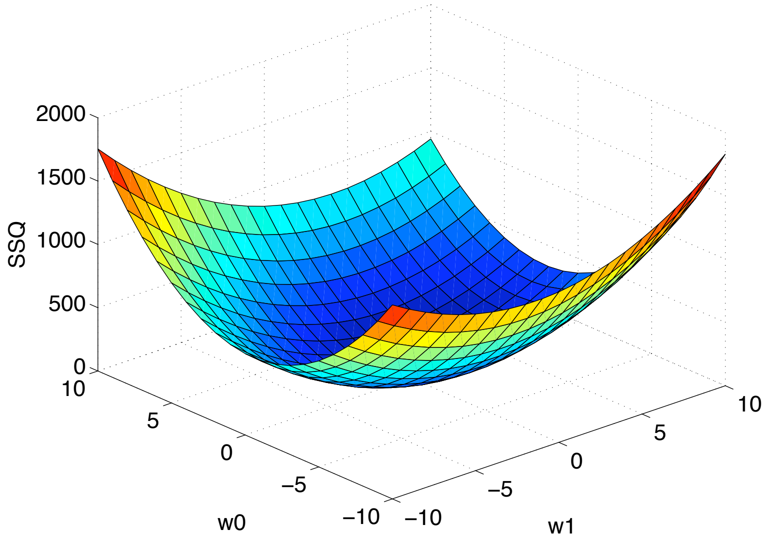


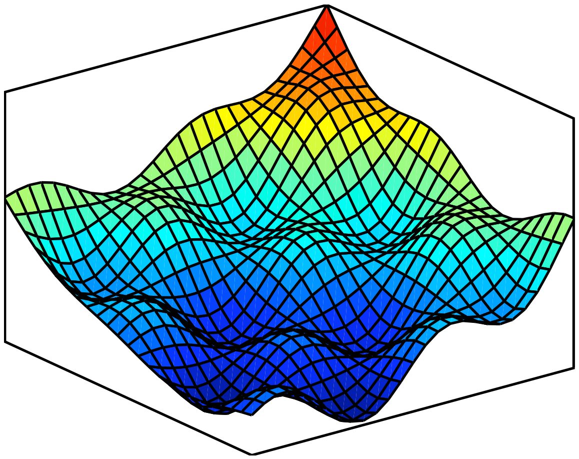
$$J_D(\mathbf{w}) = -\left(\sum_{i=1}^{m} y_i \log \sigma(\mathbf{w}^T \mathbf{x}_i) + (1 - y_i) \log(1 - \sigma(\mathbf{w}^T \mathbf{x}_i)) \right)$$

Nice error surface, unique minimum, but *cannot solve in closed form*

Gradient descent

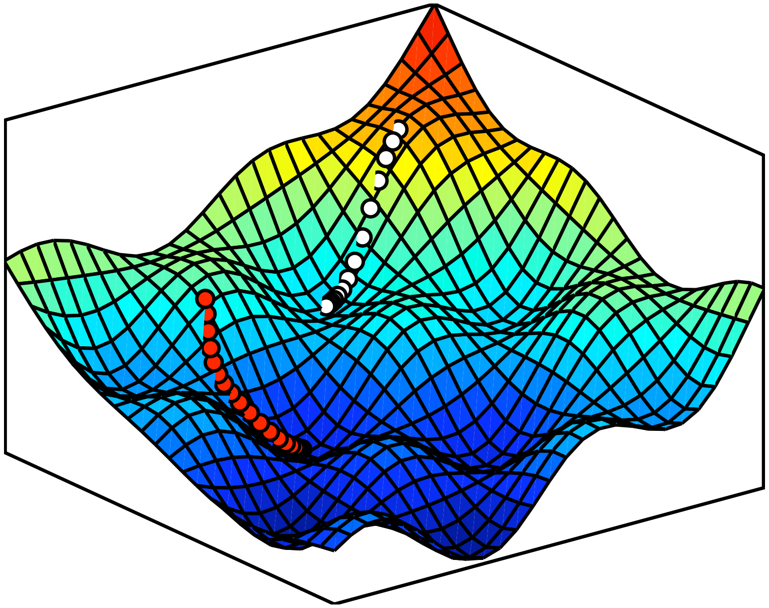
- The gradient of J at a point $\mathbf{w}$ can be thought of as a vector indicating which way is “uphill”.





- 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



- For more general hypothesis classes, there may be may local optima
- In this case, the final solution may depend 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.

Maximization procedure: Gradient ascent

- First we compute the gradient of $\log L(\mathbf{w})$ wrt $\mathbf{w}$:

$$\begin{aligned}\nabla \log L(\mathbf{w}) &= \sum_i y_i \frac{1}{h_{\mathbf{w}}(\mathbf{x}_i)} h_{\mathbf{w}}(\mathbf{x}_i)(1 - h_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i \\ &+ (1 - y_i)\frac{1}{1 - h_{\mathbf{w}}(\mathbf{x}_i)} h_{\mathbf{w}}(\mathbf{x}_i)(1 - h_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i(-1) \\ &= \sum_i \mathbf{x}_i(y_i - y_i h_{\mathbf{w}}(\mathbf{x}_i) - h_{\mathbf{w}}(\mathbf{x}_i) + y_i h_{\mathbf{w}}(\mathbf{x}_i)) = \sum_i (y_i - h_{\mathbf{w}}(\mathbf{x}_i))\mathbf{x}_i\end{aligned}$$

- The update rule (because we maximize) is:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \nabla \log L(\mathbf{w}) = \mathbf{w} + \alpha \sum_{i=1}^{m} (y_i - h_{\mathbf{w}}(\mathbf{x_i}))\mathbf{x_i} = \mathbf{w} + \alpha \mathbf{X}^T(\mathbf{y} - \hat{\mathbf{y}})$$

where $\alpha \in (0, 1)$ is a step-size or learning rate parameter

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 for logistic regression, or Fisher scoring

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 almost never necessary
- Computing the product of a Hessian with a vector can be done in linear time (Schraudolph, 1994)

Newton-Raphson for logistic regression

- Leads to a nice algorithm called *iterative recursive least squares*
- The Hessian has the form:

$$\mathbf{H} = \mathbf{\Phi}^T \mathbf{R} \mathbf{\Phi}$$

  where $\mathbf{R}$ is the diagonal matrix of $h(\mathbf{x}_i)(1 - h(\mathbf{x}_i))$ (you can check that this is the form of the second derivative.

- The weight update becomes:

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

Regularization for logistic regression

- One can do regularization for logistic regression just like in the case of linear regression
- Recall regularization makes a statement about the weights, so does not affect the error function
- Eg: L_2 regularization will have the optimization criterions:

$$J(\mathbf{w} = J_D(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

Probabilistic view of logistic regression

- Consider the additive noise model we discussed before:

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

  where ϵ are drawn iid from some distribution

- At first glance, log reg does not fit very well
- We will instead think of a latent variable $\hat{y}_i$ such that:

$$\hat{y}_i = h_{\mathbf{w}}(\mathbf{x}_i) + \epsilon$$

- Then the output is generated as:

$$y_i = 1 \text{ iff } \hat{y}_i > 0$$

Recap

- Machine learning algorithms make choices of hypothesis space, error function and optimization procedure
- In some cases, optimization is easy
- Gradient descent is a general procedure (lots more on this to come)
- All algorithms are affected by bias-variance trade-off (too much variance=overfitting)
- Bayesian interpretation gives us a handle on what the algorithms really do