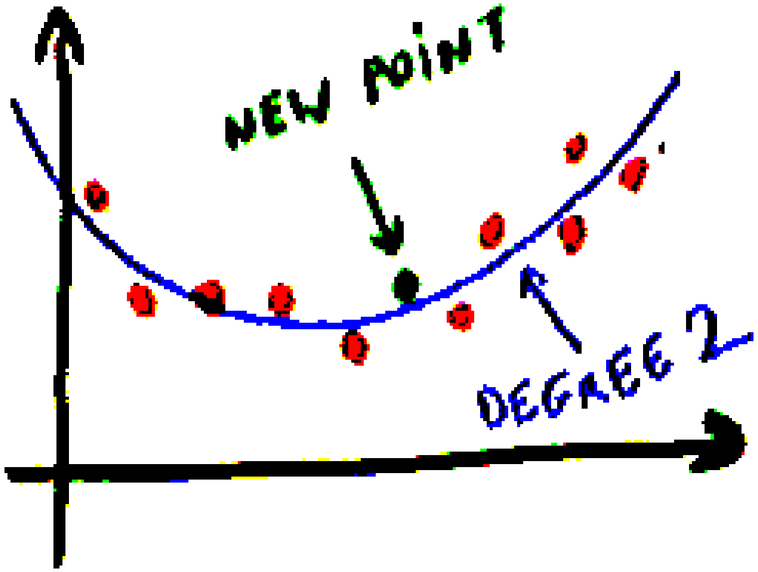
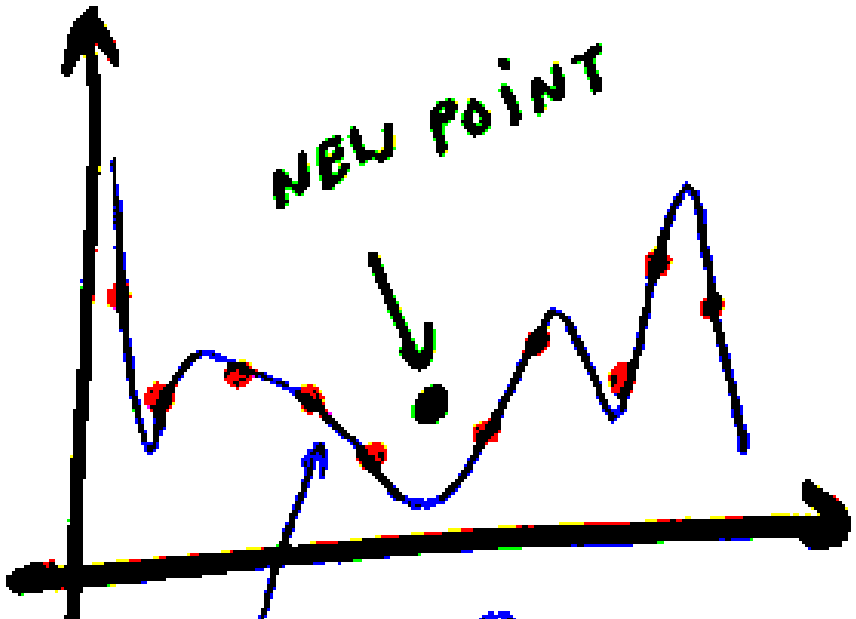
MACHINE LEARNING AND PATTERN RECOGNITION

Spring 2005, Lecture 03b, part II: Learning and Generalization, Regularization

Yann LeCun
The Courant Institute,
New York University
http://yann.lecun.com

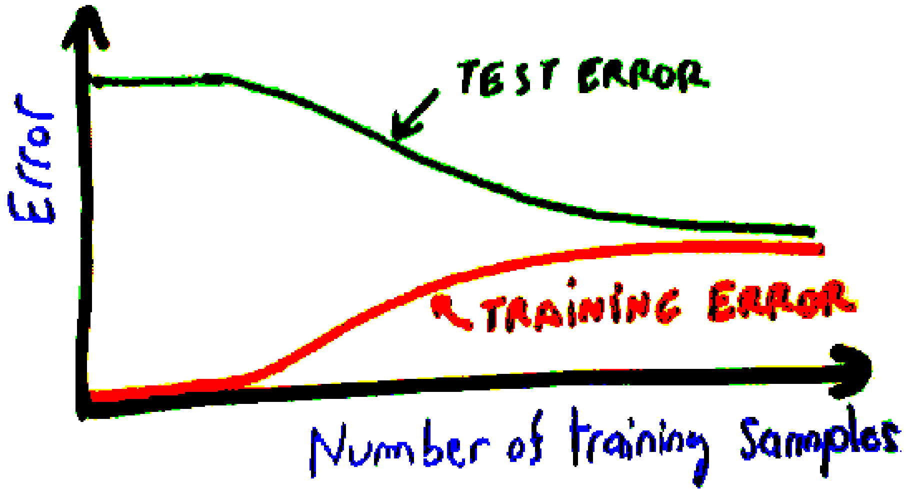
Generalization





The purpose of learning is produce a good prediction for unseen inputs. Interpolation is a form of generalization.
With too many parameters, a learning machine will learn the entire training set, but may perform poorly on new data.

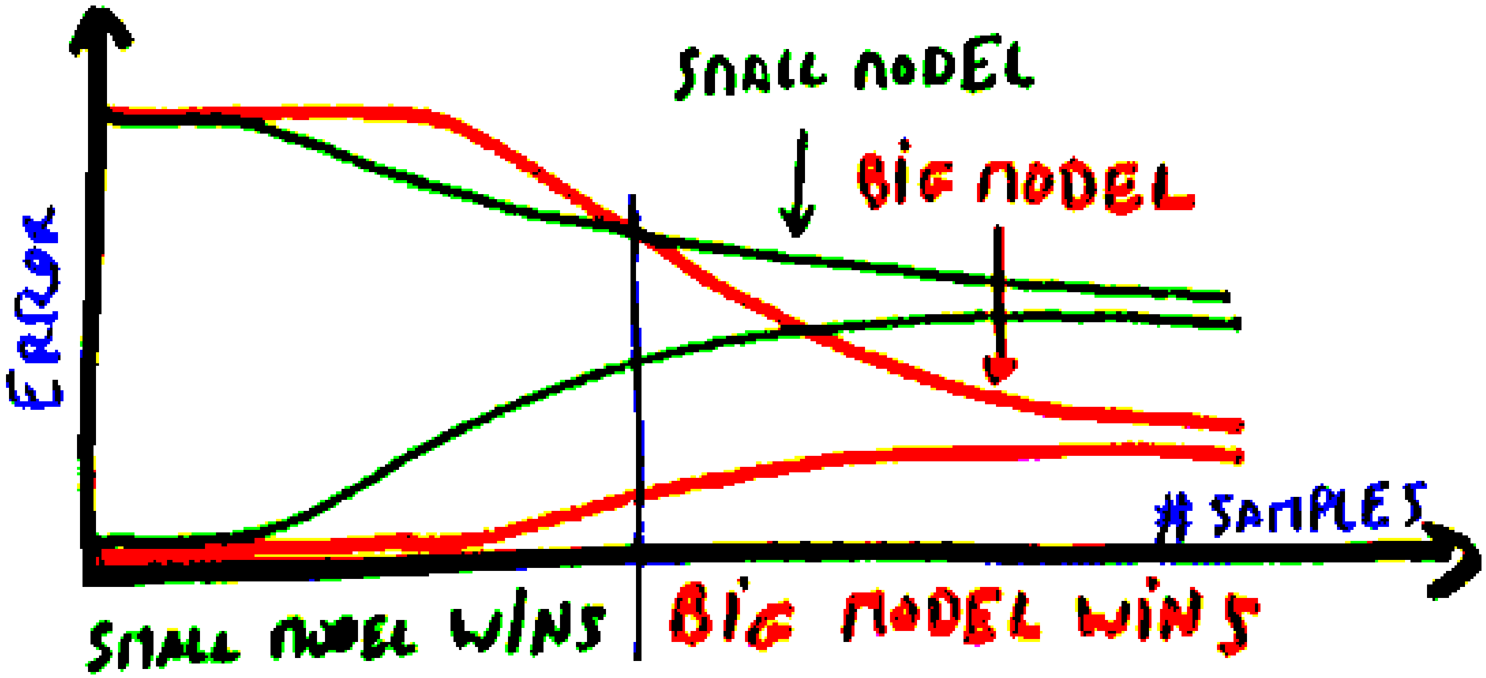
Training Error, Test Error



What we are **really** interested in is good performance on unseen data. In practice, we often partition the dataset into two subsets: a training set and a test set. We train the machine on the training set, and measure its performance on the test set.

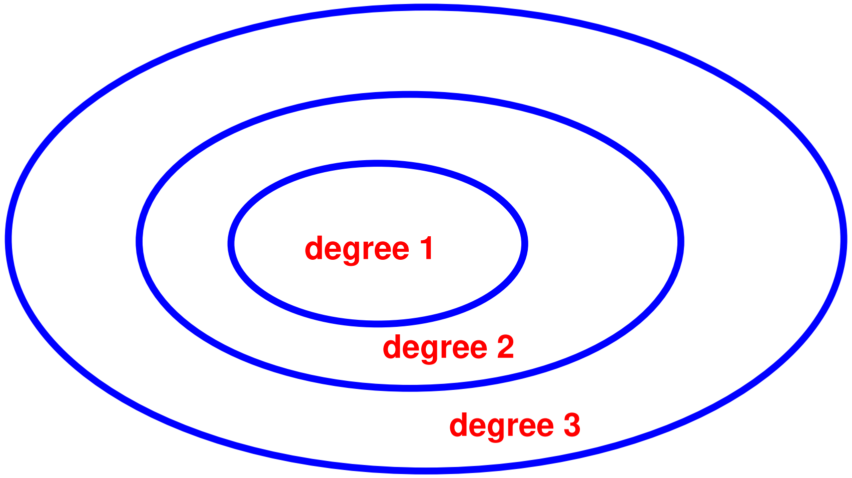
The error on the training set (the average of the loss function) is often called the **emprical risk**. The average loss on an infinite test set drawn from the same source as the training set is often called the **expected risk**.
The number of training samples for which the training error and test error start converging toward each other is called the “capacity” of the learning machine (there are formal definitions for this that we will study later on).

Learning Curves



Small/Simple models: may not do well on the training data, but the difference between training and test error quickly drops.
Big/Rich models: will learn the training data, but the difference between training and test error can be large.
How much a model deviates from the desired mapping on average is called the **model bias** of the family of functions. How much the output of a model varies when different drawings of the training set are used is called the **model variance**. There is a dilemma between bias and variance.

Model Selection



In many situations we can construct a sequence of models of increasing capacity. The capacity of a model is related to the size of the **family of functions** representable by the model.

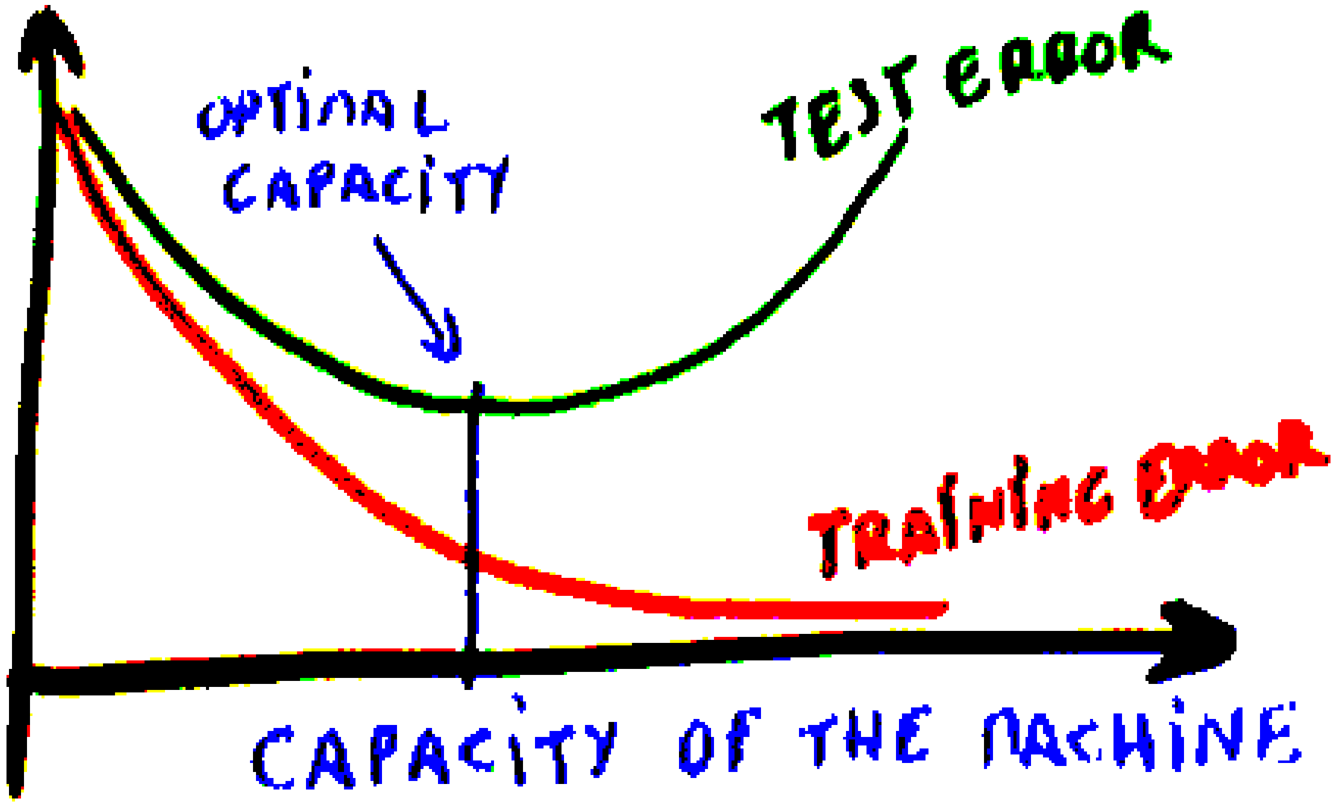
Examples:

- polynomials of degree 1, 2, 3,....
- neural nets with 10, 20, 30.... units in the first layer.
- linear regressors where $||W||^2$ is less than 1, 2, 3, 4....

Occam's Razor

- Occam's Razor: do not multiply hypotheses beyond the strict necessary.
- Occam's Razor: when given the choice between several models that explain the data equally well, choose the "simplest" one.
- Occam's Razor applied to machine learning: choose a trade off between how well the model fits the training data and how "simple" that model is.
- Occam's Razor In other contexts: *Never ascribe to malice, that which can be explained by incompetence* (Napoleon Bonaparte)

Optimal Over-Parameterization



The curve of training error and test error for a given training set size, as a function of the capacity of the machine (the richness of the class of function) has a minimum. The is the optimal size for the machine.

Regularization: Trading-off Accuracy for Simplicity

- We need a new loss function that penalizes solutions taken from "rich" families of function.
- Let's assume that the "complexity" of the family of functions from which a solution W is drawn can be computed by a function $H(W)$.
- We can minimize a loss function of the form:

$$\mathcal{L}(W) = \frac{1}{P}\left[\sum_{i=1}^{P} L(W, y^i, X^i) + \lambda H(W)\right]$$

  where L is the conventional loss function (e.g. squared error, etc) and λ is a well chosen positive constant.
- The minimum of this loss function will be a trade-off between minimizing the training error $\sum_{i=1}^{P} L(W, y^i, X^i)$ and the "simplicity" of the family of function from which W is drawn.
- the coefficient λ controls the trade-off.

Regularization

$$\mathcal{L}(W) = \frac{1}{P}\left[\sum_{i=1}^{P} L(W, y^i, X^i) + \lambda H(W)\right]$$

- $H(W)$ is called a **regularization term**. It penalizes solutions taken from "rich" families of function more than those taken from "leaner" families of functions.
- How we pick this regularization term is entirely up to us!. Despite many claims to the contrary, no theory tells us how to build the regularization function.
- By picking the family of function and the penalty term, we choose an **inductive bias**, i.e. we bias the system toward chosing simple solutions over complex ones.
- Sometimes we do not need to add an explicit regularization term because it is built implicitely into the optimization algorithm (more on this later).

Induction Principles

Assuming our samples are drawn from a distribution $P(X, Y)$, what we really want to minimize with respect to our parameter W is the expected risk (or expected loss):

$$\mathcal{L}_{\text{expected}}(W) = \int L(W, Y, X) P(X, Y) dX dY.$$

but we do not have access to $P(X, Y)$, we only have access to a few training samples drawn from it.
The method we will employ to replace the expected risk by another quantity that we can minimize is called the **induction principle**.
The simplest induction principle is called **Empirical Risk Minimization** and simply consists in minimizing the loss on the training set (training error).
The alternative, which generally consists in using a regularization term to penalize members of "rich" families of functions, is called **Structural Risk Minimization** (SRM).

Examples of Regularization Terms: Ridge Regression

Ridge Regression: penalizes large values of the parameters.

$$\mathcal{L}(W) = \frac{1}{2P}\sum_{i=1}^{P}(y^i - W'X^i)^2 + \lambda|W|^2$$

Direct solution:

$$W* = [\frac{1}{P}\sum_{i=1}^{P} X^i X^{i\prime} + \lambda I]^{-1}\sum_{i=1}^{P} y^i X^i$$

Gradient descent update (*weight decay*):

$$W \leftarrow (1 - \eta\lambda)W + \eta(y^i - W'X^i)X^i$$

Examples of Regularization Terms: Lasso

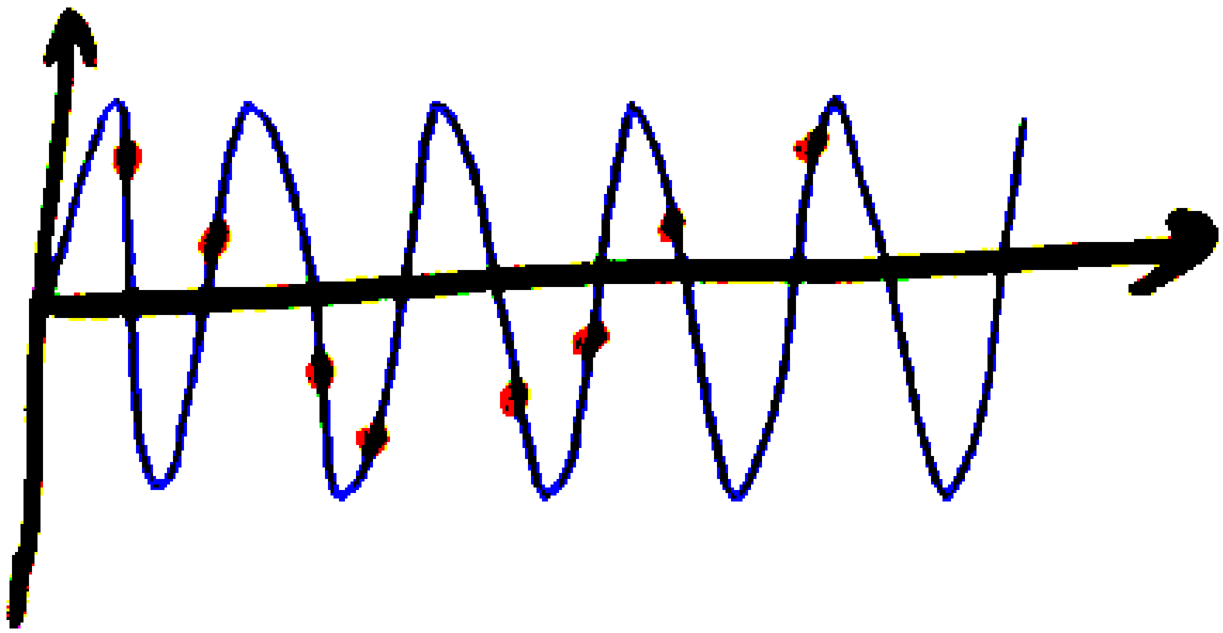
Lasso: penalizes all parameter values with a linear term (this tends to shrink small, useless parameters to 0):

$$\mathcal{L}(W) = \frac{1}{2P} \sum_{i=1}^{P} (y^i - W' X^i)^2 + \lambda |W|$$

Other Penalty Terms

What about a regularization term that simply counts the number of free parameters in the machine?

It works in some cases, but not in others. For example, the function $a.\sin(wx + b)$ has only three parameters but can exactly fit as many points as we want. This is an example of a very high-capacity function with just a few parameters:



The problem is that we need a very high accuracy on the parameters to go through all the points (lots of bits).

Minimum Description Length

A popular way of deriving regularization terms is the Minimum Description Length Principle.

$$\mathcal{L}(W) = \frac{1}{P}\left[\sum_{i=1}^{P} L(W, y^i, X^i) + H(W)\right]$$

The idea is to interpret the loss function as an expression for the *number of bits* necessary to transmit the training data.
The regularization term counts the number of bits to code the hypothesis (e.g. the value of the parameter vector), and the error term counts the number of bits to code the residual error (i.e. the difference between the predicted output and the real output.
Using efficient coding, the length of the code for a symbol is equal to the log of the probability of that symbol.

MDL: Learning as Compression

MDL comes from the idea that "compact" internal representations of a set of data are preferable to non compact ones This is another form of Occam's Razor: do not multiply hypotheses beyond the strict necessary.

Example:

complete this sequence: 01010101010101010.......

now complete that one : 01100010110010001......

The second sequence looks "random", we cannot find a compact "theory" for it.

QUESTION: How do we measure randomness?

Sometimes, a simple underlying rule exists, but it is very hard to find.

Example: 9265358979323846264338328.....

Can you guess?

Measuring Randomness?

The Kolmogoroff/Chaitin/Solomonoff theory of complexity gives us a theoretical framework:

The KCS complexity of a string of bits S relative to computer C is the length of the shortest program that will output S when run on C.

Good news: the complexity given by two different universal computers differ at most by a constant (the size of the program that will make one computer emulate the other).

Bad News 1: that constant can be very, very, very large. So in practice, there is no absolute measure randomness for finite strings.

Bad New 2: the KCS complexity of a string is non-computable in general (you can't enumerate all the programs, because some won't halt).

Although this is a very rich and cool theoretical concept, we can't really use it in practice.

Learning Theory

A better theoretical framework for studying generalization, regularization and structural risk minimization is the so-called Statistical Learning Theory.
We will study that approach later in the course.