MACHINE LEARNING AND PATTERN RECOGNITION

Fall 2005 Lecture 1: Introduction and Basic Concepts

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

Before we get started...

- Course web site: http://www.cs.nyu.edu/ yann/2005f-G22-2565-001/index.html
- Evaluation: Assignements (mostly small programming projects) [65%] + larger final project [35%].
- Course mailing list: http://www.cs.nyu.edu/mailman/listinfo/g22_2565_001_fa05
- Text Books: mainly "Pattern Classification" by Duda, Hart, and Stork, but a number of other books can be used reference material: "Neural Networks for Pattern Recognition" by Bishop, and "Element of Statistical Learning" by Hastie, Tibshirani and Friedman.
- ... but we will mostly use resarch papers and tutorials.
- formal prerequisite: linear algebra. You might want to brush up on probability theory, multivariate calculus (partial derivatives ...), optimization (least square method...), and the method of Lagrange multipliers for constrained optimization. We will review those topics in class.
- Programming projects: can be done in any language, but I **STRONGLY** recommend to use Lush (http://lush.sf.net). Skeleton code in Lush will be provided for most projects.

What is Learning?

- Learning is acquiring and improving performance through experience.
- Pretty much all animals with a central nervous system are capable of learning (even the simplest ones).
- What does it mean for a computer to learn? Why would we want them to learn? How do we get them to learn?
- We want computers to learn when it is too difficult or too expensive to program them directly to perform a task.
- Get the computer to program itself by showing examples of inputs and outputs.
- In reality: we will write a "parameterized" program, and let the learning algorithm find the set of parameters that best approximates the desired function or behavior.

Different Types of Learning

- Supervised Learning: given training examples of inputs and corresponding outputs, produce the “correct” outputs for new inputs. Example: character recognition.
- Reinforcement Learning (similar to animal learning): an agent takes inputs from the environment, and takes actions that affect the environment. Occasionally, the agent gets a scalar reward or punishment. The goal is to learn to produce action sequences that maximize the expected reward (e.g. driving a robot without bumping into obstacles). I won’t talk much about that in this course.
- Unsupervised Learning: given only inputs as training, find structure in the world: discover clusters, manifolds, characterize the areas of the space to which the observed inputs belong (e.g.: clustering, probability density estimation, novelty detection, compression, embedding).

Related Fields

- Statistical Estimation: statistical estimation attempts to solve the same problem as machine learning. Most learning techniques are statistical in nature.
- Pattern Recognition: pattern recognition is when the output of the learning machine is a set of discrete categories.
- Neural Networks: neural nets are now one many techniques for statistical machine learning.
- Data Mining: data mining is a large application area for machine learning.
- Adaptive Optimal Control: non-linear adaptive control techniques are very similar to machine learning methods.
- Machine Learning methods are an essential ingredient in many fields: bio-informatics, natural language processing, web search and text classification, speech and handwriting recognition, fraud detection, financial time-series prediction, industrial process control, database marketing....

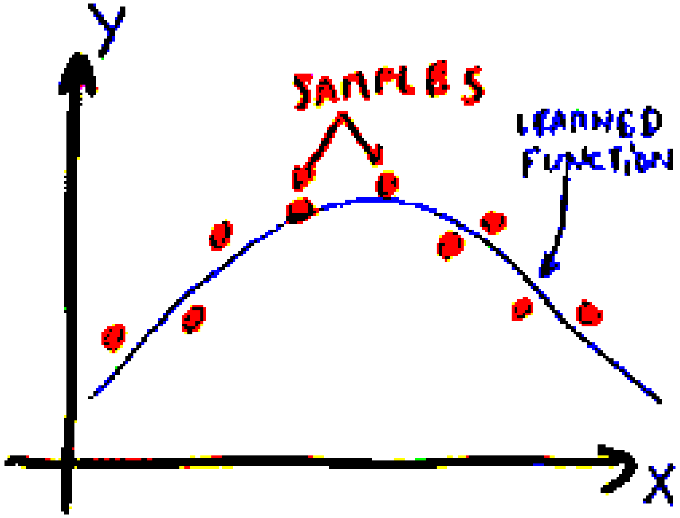
Applications

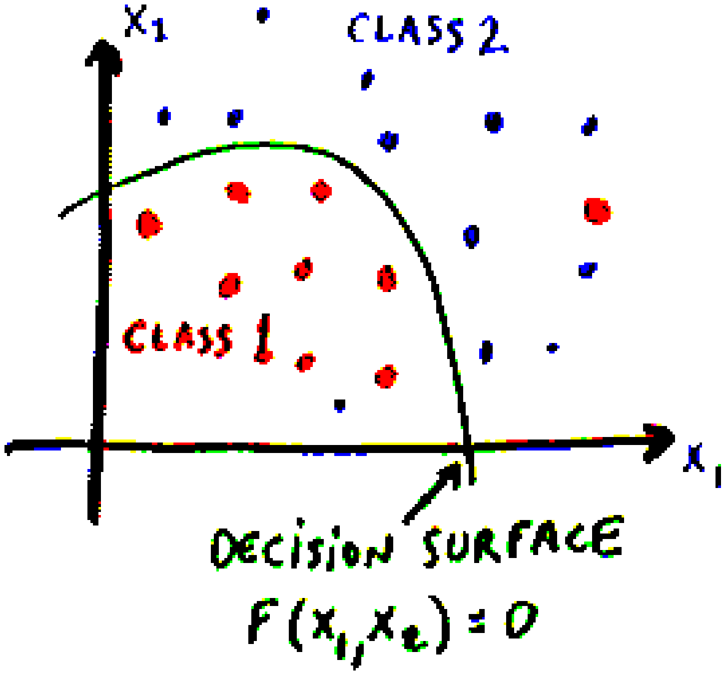
- handwriting recognition, OCR: reading checks and zipcodes, handwriting recognition for tablet PCs.
- speech recognition, speaker recognition/verification
- security: face detection and recognition, event detection in videos.
- text classification: indexing, web search.
- computer vision: object detection and recognition.
- diagnosis: medical diagnosis (e.g. pap smears processing)
- adaptive control: locomotion control for legged robots, navigation for mobile robots, minimizing pollutant emissions for chemical plants, predicting consumption for utilites...
- fraud detection: e.g. detection of "unusual" usage patterns for credit cards or calling cards.
- database marketing: predicting who is more likely to respond to an ad campaign.
- (...and the antidote) spam filtering.
- games (e.g. backgammon).
- Financial prediction (many people on Wall Street use machine learning).

Demos / Concrete Examples

- Handwritten Digit Recognition: supervised learning for classification
- Handwritten Word Recognition: weakly supervised learning for classification with many classes
- Face detection: supervised learning for detection (faces against everything else in the world).
- Object Recognition: supervised learning for detection and recognition with highly complex variabilities
- Robot Navigation: supervised learning and reinforcement learning for control.

Two Kinds of Supervised Learning

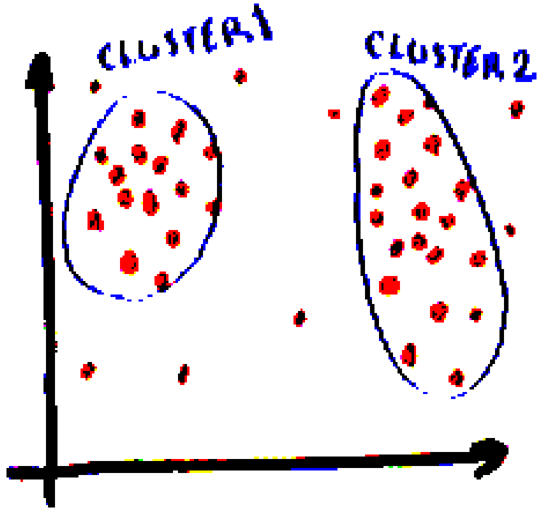


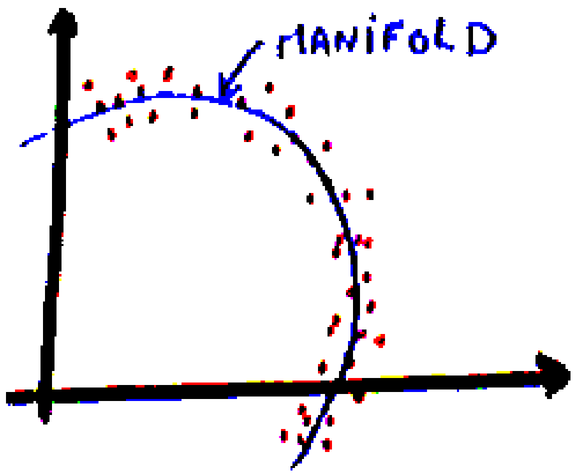


- Regression: also known as "curve fitting" or "function approximation". Learn a continuous input-output mapping from a limited number of examples (possibly noisy).
- Classification: outputs are discrete variables (category labels). Learn a decision boundary that separates one class from the other. Generally, a "confidence" is also desired (how sure are we that the input belongs to the chosen category).

Unsupervised Learning

Unsupervised learning comes down to this: if the input looks like the training samples, output a small number, if it doesn't, output a large number.



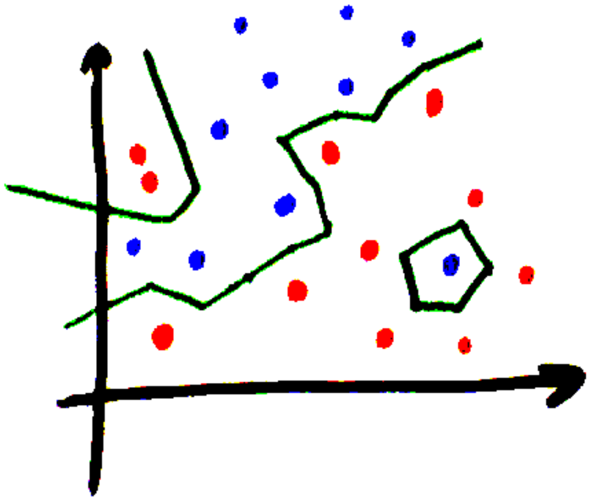


- This is a horrendously ill-posed problem in high dimension. To do it right, we must guess/discover the hidden structure of the inputs. Methods differ by their assumptions about the nature of the data.
- A Special Case: Density Estimation. Find a function f such $f(X)$ approximates the probability density of X, $p(X)$, as well as possible.
- Clustering: discover "clumps" of points
- Embedding: discover low-dimensional manifold or surface near which the data lives.
- Compression/Quantization: discover a function that for each input computes a compact "code" from which the input can be reconstructed.

Learning is NOT Memorization

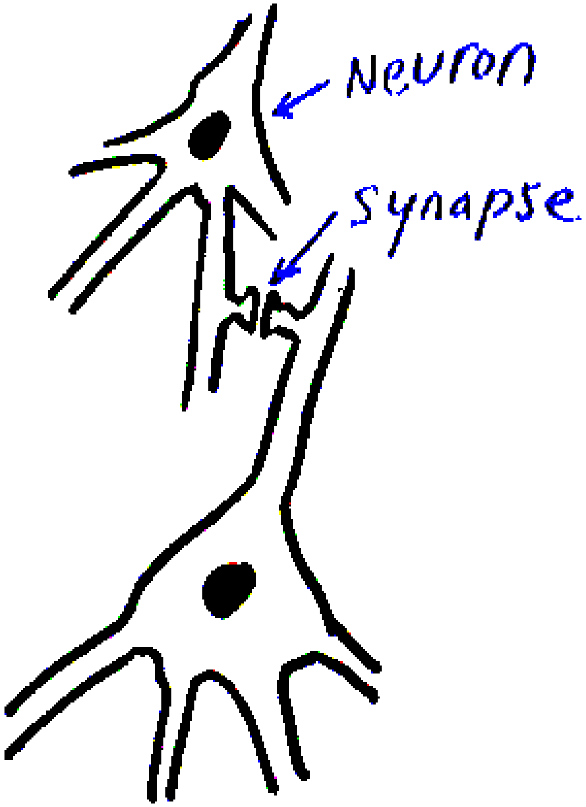
- rote learning is easy: just memorize all the training examples and their corresponding outputs.
- when a new input comes in, compare it to all the memorized samples, and produce the output associated with the matching sample.
- PROBLEM: in general, new inputs are different from training samples.
- The ability to produce correct outputs or behavior on previously unseen inputs is called GENERALIZATION.
- rote learning is memorization without generalization.
- The big question of Learning Theory (and practice): how to get good generalization with a limited number of examples.

A Simple Trick: Nearest Neighbor Matching



- Instead of insisting that the input be exactly identical to one of the training samples, let's compute the "distances" between the input and all the memorized samples (aka the prototypes).
- 1-Nearest Neighbor Rule: pick the class of the nearest prototype.
- K-Nearest Neighbor Rule: pick the class that has the majority among the K nearest prototypes.
- PROBLEM: What is the right distance measure?
- PROBLEM: This is horrendously expensive if the number of prototypes is large.
- PROBLEM: do we have any guarantee that we get the best possible performance as the number of training samples increases?

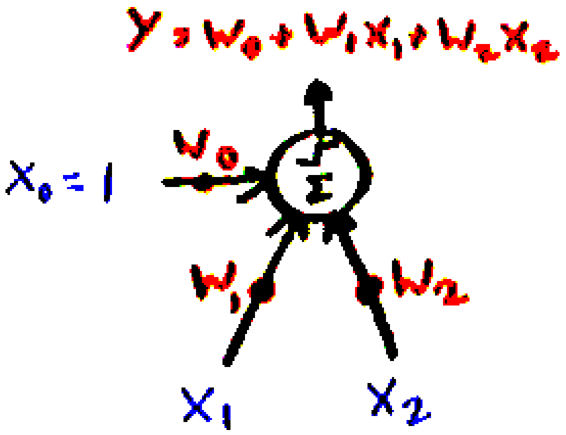
How Biology Does It

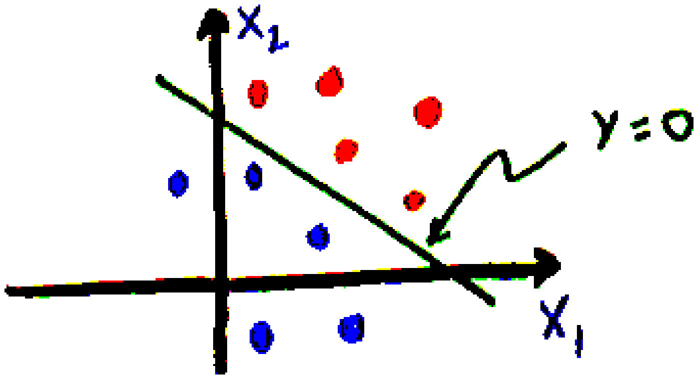


- The first attempts at machine learning in the 50's, and the development of artificial neural networks in the 80's and 90's were inspired by biology.
- Nervous Systems are networks of neurons interconnected through synapses
- Learning and memory are changes in the "efficacy" of the synapses
- HUGE SIMPLIFICATION: a neuron computes a weighted sum of its inputs (where the weights are the synaptic efficacies) and fires when that sum exceeds a threshold.
- Hebbian learning (from Hebb, 1947): synaptic weights change as a function of the pre- and post-synaptic activities.
- orders of magnitude: each neuron has 10^3 to 10^5 synapses. Brain sizes (number of neurons): house fly: 10^5; mouse: 5.10^6, human: 10^{10}.

The Linear Classifier

Historically, the Linear Classifier was designed as a highly simplified model of the neuron (McCulloch and Pitts 1943, Rosenblatt 1957):





$$y = f\left(\sum_{i=0}^{i=N} w_i x_i\right)$$

With f is the threshold function: $f(z) = 1$ iff $z > 0$, $f(z) = -1$ otherwise. x_0 is assumed to be constant equal to 1, and w_0 is interpreted as a bias.
In vector form: $W = (w_0, w_1 ....w_n)$, $X = (1, x_1...x_n)$:

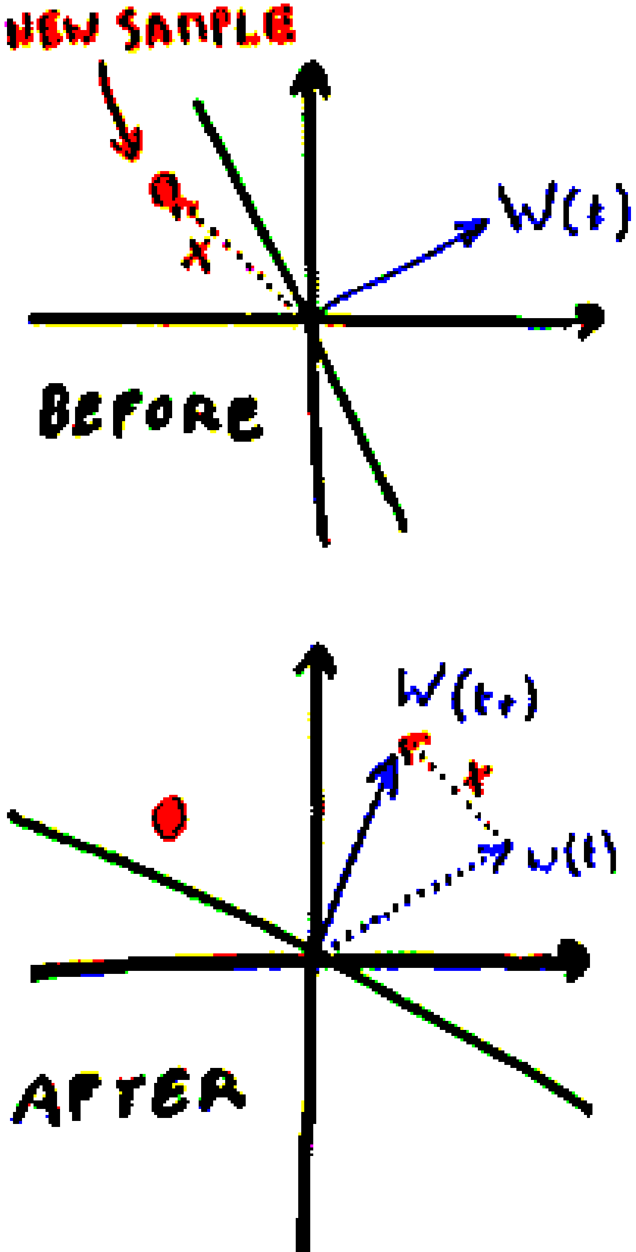
$$y = f(W'X)$$

The hyperplane $W'X = 0$ partitions the space in two categories. W is orthogonal to the hyperplane.

Vector Inputs

With vector-based classifiers such as the linear classifier, we must represent objects in the world as vectors.
Each component is a measurement or a feature of the the object to be classified.
For example, the grayscale values of all the pixels in an image can be seen as a (very high-dimensional) vector.

A Simple Idea for Learning: Error Correction



We have a **training set** $\mathcal{S}$consisting of P input-output pairs: $\mathcal{S} = (X^1, y^1), (X^2, y^2), ...(X^P, y^P)$.
A very simple algorithm:
- show each sample in sequence repetitively
- if the output is correct: do nothing
- if the output is -1 and the desired output +1: increase the weights whose inputs are positive, decrease the weights whose inputs are negative.
- if the output is +1 and the desired output -1: decrease the weights whose inputs are positive, increase the weights whose inputs are negative.

More formally, for sample p:

$$w_i(t+1) = w_i(t) + (y_i^p - f(W'X^p))x_i^p$$

This simple algorithm is called the Perceptron learning procedure (Rosenblatt 1957).

The Perceptron Learning Procedure

Theorem: If the classes are linearly separable (i.e. separable by a hyperplane), then the Perceptron procedure will converge to a solution in a finite number of steps.
Proof: Let's denote by W^* a normalized vector in the direction of a solution. Suppose all X are within a ball of radius R. Without loss of generality, we replace all X^p whose y^p is -1 by $-X^p$, and set all y^p to 1. Let us now define the margin $M = min_p W^* X^p$. Each time there is an error, $W.W^*$ increases by at least $X.W^* \geq M$. This means $W_{final}.W^* \geq NM$ where N is the total number of weight updates (total number of errors). But, the change in square magnitude of W is bounded by the square magnitude of the current sample X^p, which is itself bounded by R^2. Therefore, $|W_{final}|^2 \leq NR^2$. combining the two inequalities $W_{final}.W^* \geq NM$ and $|W_{final}| \leq \sqrt{N}R$, we have

$$W_{final}.W^*/|W_{final}| \geq \sqrt{(}N)M/R$$

. Since the left hand side is upper bounded by 1, we deduce

$$N \leq R^2/M^2$$

Good News, Bad News

The perceptron learning procedure can learn a linear decision surface, **if such a surface exists** that separates the two classes. If no perfect solution exists, the perceptron procedure will keep wobbling around.

What class of problems is **Linearly Separable**, and learnable by a Perceptron?

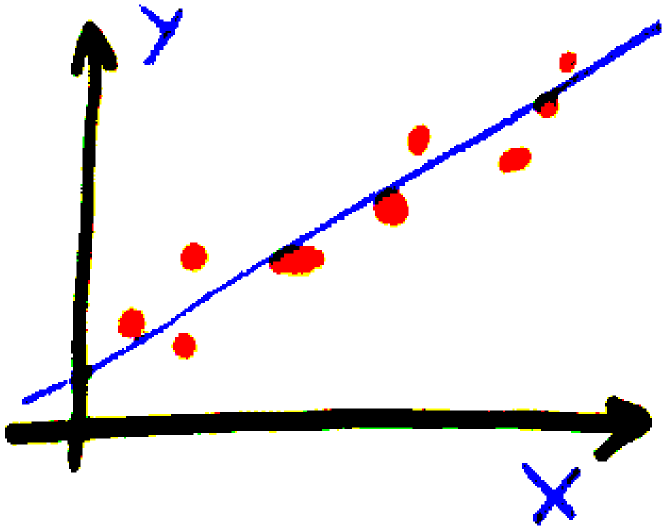
There are many interesting applications where the data can be represented in a way that makes the classes (nearly) linearly separable: e.g. text classification using “bag of words” representations (e.g. for spam filtering).

Unfortunately, the really interesting applications are generally not linearly separable.

This is why most people abandonned the field between the late 60’s and the early 80’s.

We will come back to the linear separability problem later.

Regression, Mean Squared Error



Regression or function approximation is finding a function that approximates a set of samples as well as possible.

Classic example: linear regression. We are given a **training set** $\mathcal{S}$ of input/output pairs $\mathcal{S} = \{(X^1, y^1), (X^2, y^2)....(X^P, y^P)\}$, and we must find the parameters of a linear function that best predicts the y's from the X's in the least square sense. In other words, we must find the parameter W that minimizes the quadratic **loss function** $\mathcal{L}(W, \mathcal{S})$:

$$\mathcal{L}(W, \mathcal{S}) = \frac{1}{P} \sum_{i=1}^{P} L(W, y^i, X^i)$$

where the **per-sample loss function** $L(W, y^i, X^i)$ is defined as:

$$L(W, y^i, X^i) = \frac{1}{2}(y^i - W'X^i)^2$$

Regression: Solution

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

$$W^* = \text{argmin}_W \mathcal{L}(W) = \text{argmin}_W \frac{1}{P}\sum_{i=1}^{P}\frac{1}{2}(y^i - W'X^i)^2$$

At the solution, W satisfies the extremality condition:

$$\frac{d\mathcal{L}(W)}{dW} = 0$$

$$\frac{d\left[\frac{1}{P}\sum_{i=1}^{P}\frac{1}{2}(y^i - W'X^i)^2\right]}{dW} = 0$$

$$\sum_{i=1}^{P}\frac{d\left[\frac{1}{2}(y^i - W'X^i)^2\right]}{dW} = 0$$

A digression on multivariate calculus

- W: a vector of dimension N. W' denotes the transpose of W, i.e. if W is $N \times 1$ (column vector), W' is $1 \times N$ (line vector).
- $F(W)$: a multivariate scalar-valued function (an N-dimensional surface in an $N+1$ dimensional space).
- $$\frac{dF(W)}{dW} = \left[\frac{\partial F(W)}{\partial w_1}, \frac{\partial F(W)}{\partial w_2}, \ldots \frac{\partial F(W)}{\partial w_N}\right]$$

  is the gradient of $F(W)$ with respect to W (it's a line vector).
- The gradient of a function that maps N-dim vectors scalars is a $1 \times N$ line vector.
- example 1: linear function: $F(W) = W'X$ where X is an N-dim vector: $\frac{d(W'X)}{dW} = X'$
- example 2: quadratic function $F(W) = (y - W'X)^2$ where y is a scalar: $\frac{d(y-W'X)^2}{dW} = -2(y - W'X)X'$.

Regression: Solution

The gradient of $\mathcal{L}$(W) is:

$$\frac{d\mathcal{L}(W)}{dW} = \sum_{i=1}^{P} \frac{d\left[\frac{1}{2}(y^i - W'X^i)^2\right]}{dW} = \sum_{i=1}^{P} -(y^i - W'X^i)X^{i\prime}$$

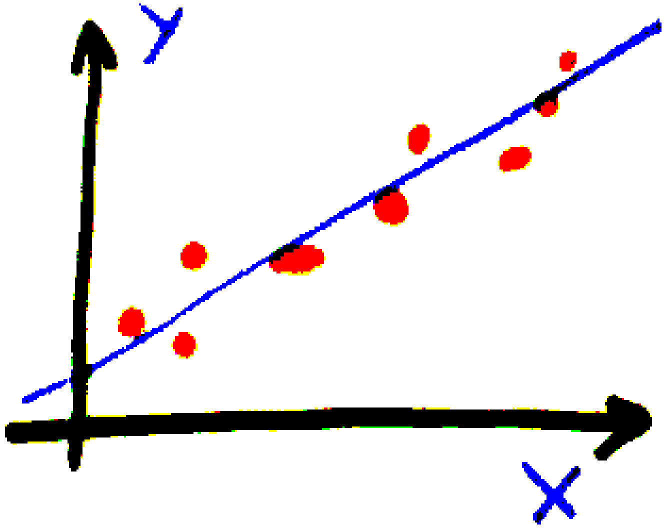
The extremality condition becomes:

$$\frac{1}{P}\sum_{i=1}^{P} -(y^i - W'X^i)X^{i\prime} = 0$$

Which we can rewrite as:

$$\left[\sum_{i=1}^{P} y^i X^i\right] - \left[\sum_{i=1}^{P} X^i X^{i\prime}\right] W = 0$$

Regression: Direct Solution



$$\sum_{i=1}^{P} y^i X^i - [\sum_{i=1}^{P} X^i X^{i'}]W = 0$$

Can be written as:

$$[\sum_{i=1}^{P} X^i X^{i'}]W = \sum_{i=1}^{P} y^i X^i$$

This is a linear system that can be solved with a number of traditional numerical methods (although it may be ill-conditioned or singular).

If the **covariance matrix** $A = \sum_{i=1}^{P} X^i X^{i'}$ is non singular, the solution is:

$$W^* = \left[\sum_{i=1}^{P} X^i X^{i'}\right]^{-1} \sum_{i=1}^{P} y^i X^i$$

Regression: Iterative Solution

Gradient-based minimization: $W(t+1) = W(t) - \eta \frac{d\mathcal{L}(W)}{dW}$
where η is a well chosen coefficient (often a scalar, sometimes diagonal matrix with positive entries, occasionally a full symmetric positive definite matrix).
The k-th component of the gradient of the quadratic loss $\mathcal{L}(W)$ is:

$$\frac{\partial \mathcal{L}(W)}{\partial w_k} = \sum_{i=1}^{P} -(y^i - W(t)'X^i)x_k^i$$

If η is a scalar or a diagonal matrix, we can write the udpate equation for a single component of W: $w_k(t+1) = w_k(t) + \eta \sum_{i=1}^{P}(y^i - W(t)'X^i)x_k^i$
This update rules converges for well-chosen, small-enough values of η (more on this later).

Regression, Online/Stochastic Gradient

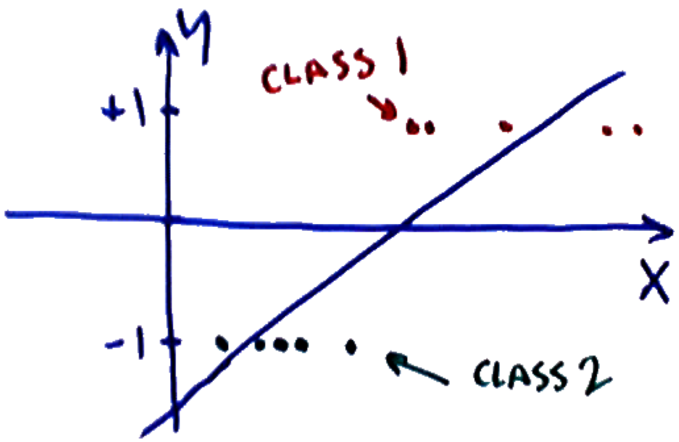
Online gradient descent, aka Stochastic Gradient:

$$W(t+1) = W(t) - \eta \frac{d(W, Y^i, X^i)}{dW}$$

$$w_k(t+1) = w_k(t) + \eta(t)(y^i - W(t)'X^i)x_k^i$$

No sum! The average gradient is replaced by its instantaneous value.
This is called **stochastic gradient descent**. In many practical situation it is **enormously faster** than batch gradient.
But the convergence analysis of this method is very tricky.
One condition for convergence is that $\eta(t)$ must be decreased according to a schedule such that $\sum_t \eta(t)^2$ converges while $\sum_t \eta(t)$ diverges.
One possible such sequence is $\eta(t) = \eta_0/t$.
We can also use second-order methods, but we will keep that for later.

Least Mean Squared Error for Classification



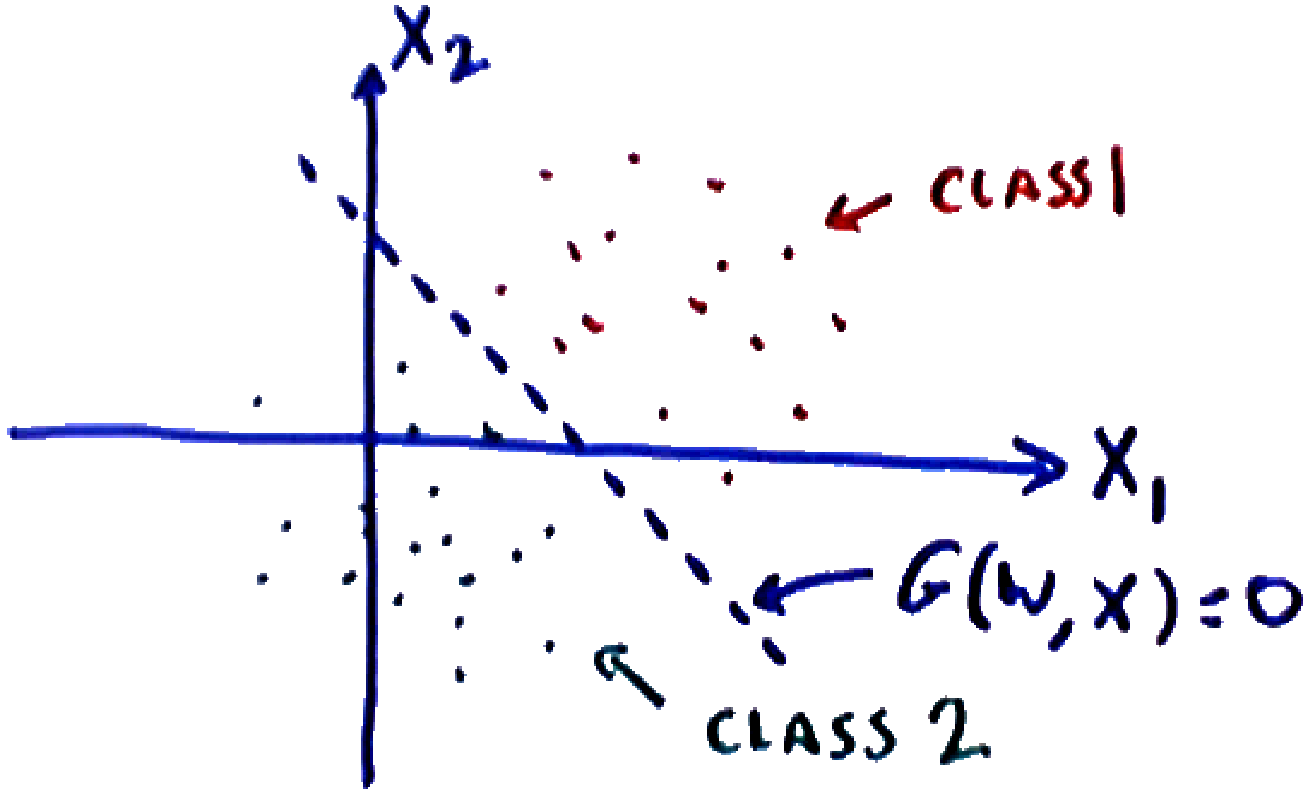
We can use the Mean Squared Error criterion with a linear regressor to perform classification (although this is clearly suboptimal).
We compute a **linear discriminant function** $G(W, X) = W'X$ and compare it to a threshold T. If $G(W, X)$ is larger than T, we classify X in class 1, if it is smaller than T, we classify X n class 2.

- To compute W, we simply minimize the quadratic loss function

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

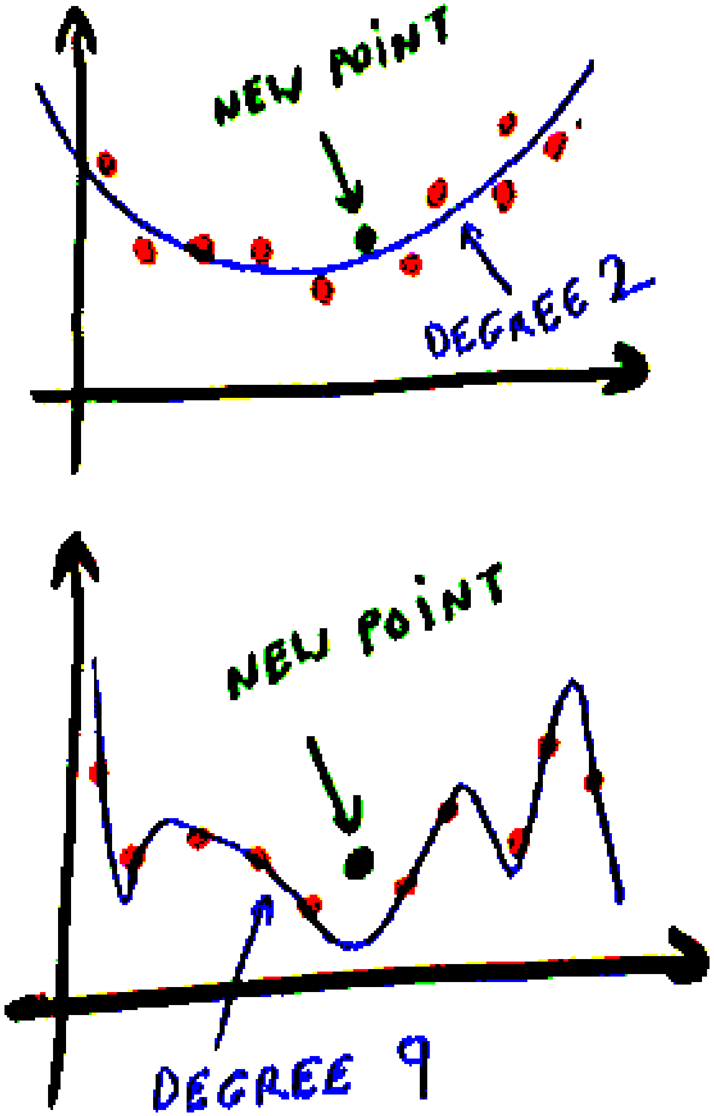
  where $y^i = +1$ if training sample X^i is of class 1 and $y^i = -1$ if training sample X^i is of class 2.
- This is called the Adaline algorithm (Widrow-Hoff 1960).

Linear Classifiers



In multiple dimensions, the linear discriminant function $G(W, X) = W'X$ partitions the space into two half-spaces separated by a hyperplane.

A Richer Class of Functions



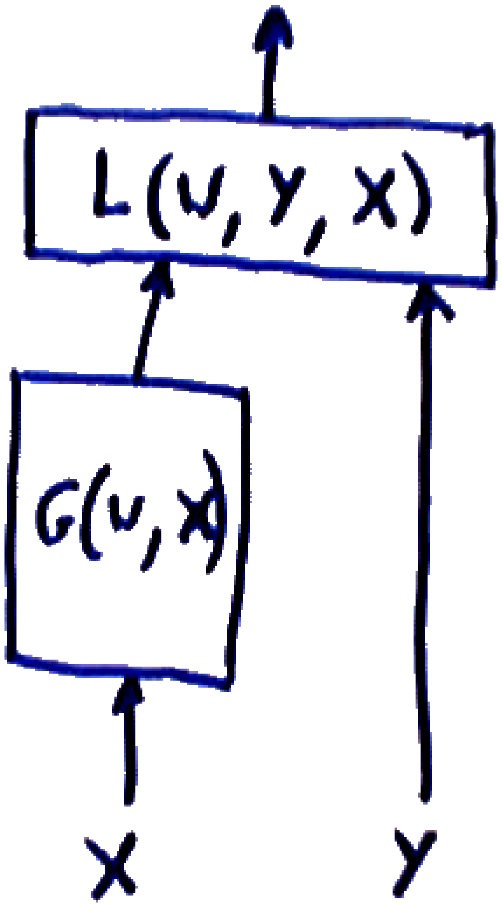
What if we know that the process that generated our samples is non linear? We can use a richer **family of functions**, e.g. polynomials, sum of trigonometric functions....

PROBLEM: if the family of functions is too rich, we run the risk of **overfitting** the data. If the family is too restrictive we run the risk of not being able to approximate the training data very well.

QUESTIONS: How can we choose the richness of the family of functions? Can we predict the performance on new data as a function of the training error and the richness of the family of functions?

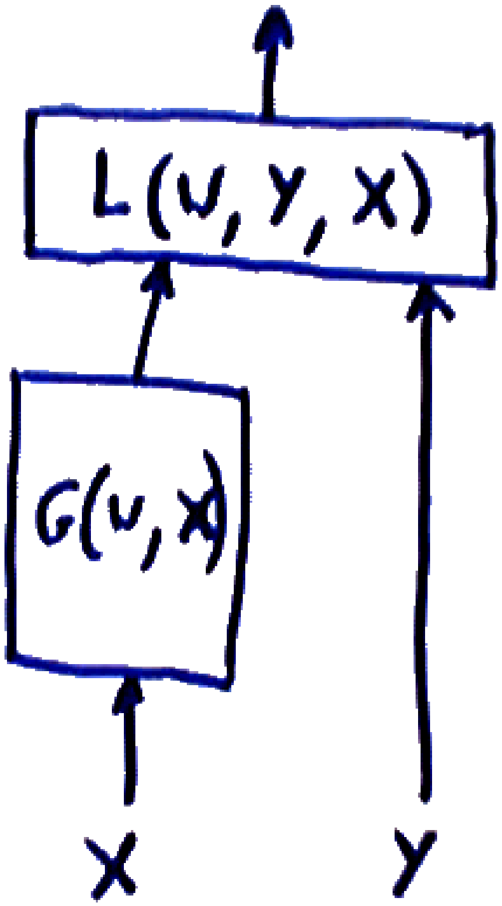
Simply minimizing the training error may not give us a solution that will do well on new data.

Learning as Function Estimation



- pick a **machine** $G(W, X)$ parameterized by W. It can be complicated and non-linear, but it better be differentiable with respect to W.
- pick a **per-sample loss function** $L(Y, G(W, X))$.
- pick a training set $\mathcal{S} = (X^1, Y^1), (X^2, Y^2), ...(X^P, Y^P)$.
- find the W that minimizes $\mathcal{L}(W, \mathcal{S}) = \frac{1}{P} \sum_i L(Y^i, G(W, X^i))$

Learning as Function Estimation (continued)



- If $L(Y^i, G(W, X^i))$ is differentiable with respect to W, use a gradient-based minimization technique:

$$W \leftarrow W - \eta \frac{\partial \mathcal{L}(W)}{\partial W}$$

- or use a stochastic gradient minimization technique:

$$W \leftarrow W - \eta \frac{\partial L(Y^i, G(W, X^i))}{\partial W}$$