Introduction to Machine Learning

Prof. Sotirios A Tsaftaris
WP3.1 Lead in UDRC 3
Chair in Computer Vision and Machine Learning
Canon Medical / Royal Academy of Engineering Research Chair in Healthcare AI
Turing Fellow
http://tsaftaris.com
S.Tsaftaris@ed.ac.uk

DRAFT, released for enhanced learning; final version to be made available electronically on the day.

Some slides adopted from: Mike Davies, A. Katsaggelos, S. Theodoridis, A. Ng

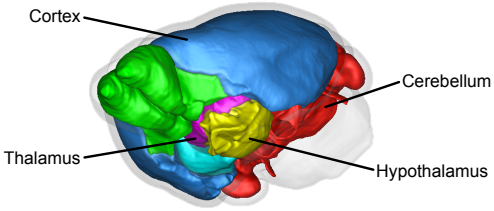
Disclaimers

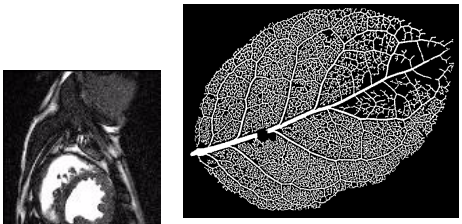
- The following slides contain material from several sources.
- They are to be used by participants of the summer school and cannot be distributed without permission from the lecturer.
- Copyright of figures remains on the copyright holders which may not be solely of the author. Atttribution has been given when possible but has not been exhaustive.

- Material shared maybe more than what we will be covered at delivery.
- Material subject to change.

Our expertise

- **Computer vision & Image Analysis:**
  - Extract information from images
  - Applications: medicine, plants
- **Machine learning & pattern recognition**
  - Shallow or deep representation learning (the process of making sense of data)
- Distributed learning

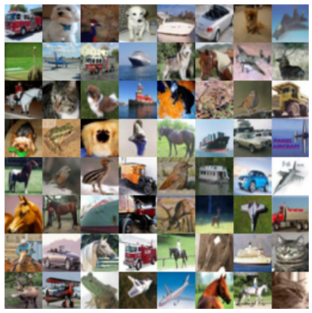




$z = f($  $)$

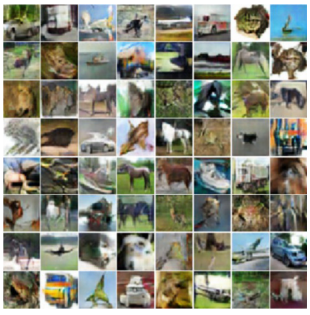
An example of ML in UDRC

- Given data can I learn their distribution?



Real images (CIFAR-10)

- Learn a distribution that can generate that data?



Generated images

What is machine learning?

- What do you think?

Some applications

- Email spam filtering
- Netflix/Amazon recommendations
- Google suggested queries
- The Google index itself
- Predicting stock prices
- Classifying threats in images
- etc

Extreme(…) applications

- MIT flight
- http://www.youtube.com/watch?v=aiNX-vpDhMo
- Robot in the dessert
- http://www.youtube.com/watch?v=OIOtOmyySQo
- Google car
- http://www.youtube.com/watch?v=cdgQpa1pUUE

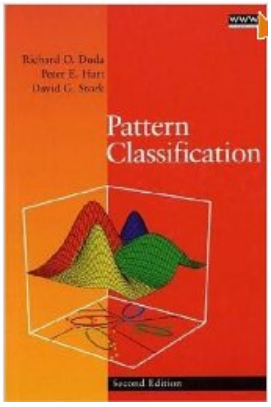
What is machine learning?

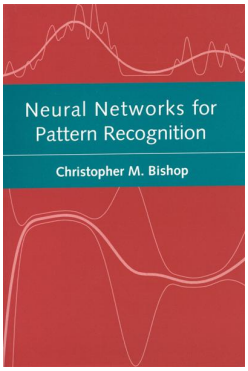
- Arthur Samuel [1959] (informal definition) Gives computers ability to learn without being explicitly programmed.

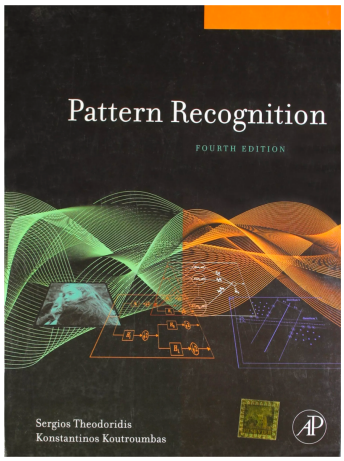
  ➔He built the very first checker's program
- Tom Mitchell [98] (more formal): A well-posed learning problem is defined as follows:
  - A computer program is set to learn from an **experience E** with respect to some **task T** and some **performance measure P** if its performance on **T** as measured by P **improves** with experience E.

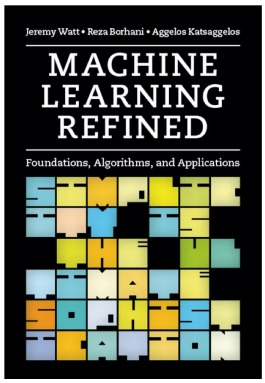
Text Books

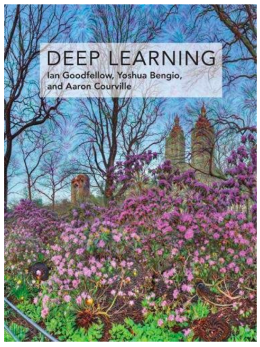
- Useful texts ...











- Most can be found online or libraries: e.g.

https://github.com/jermwatt/machine_learning_refined

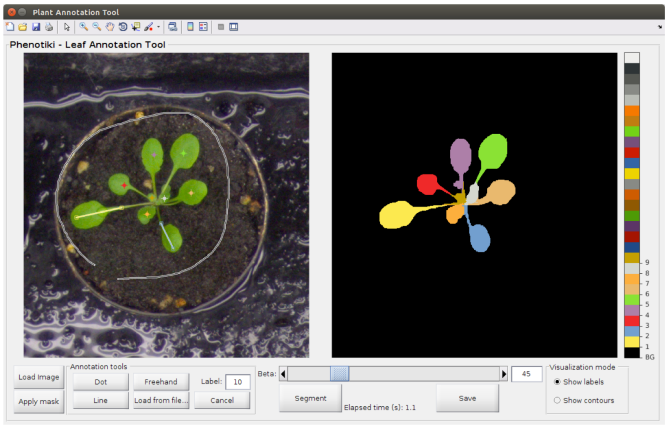
https://www.deeplearningbook.org/

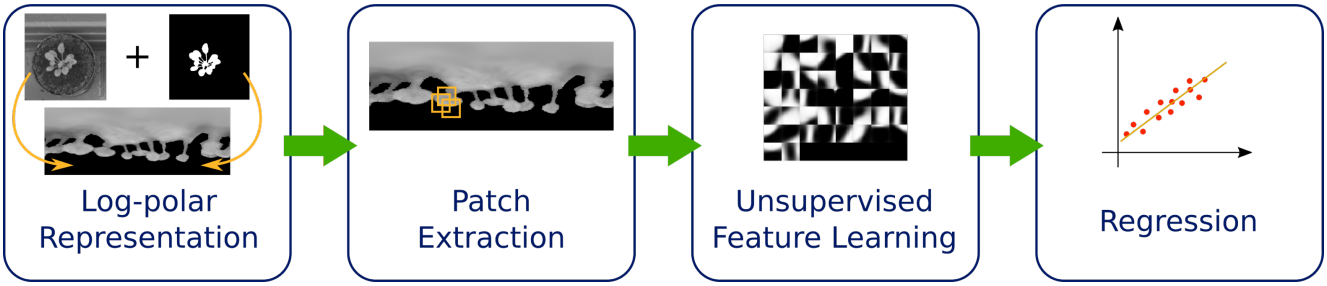
What *we* do with ML…

- We build algorithms to analyze **imaging** data (2D, 3D, 2D+t, 3D+t)
- From a **variety** of domains
- Use machine learning throughout

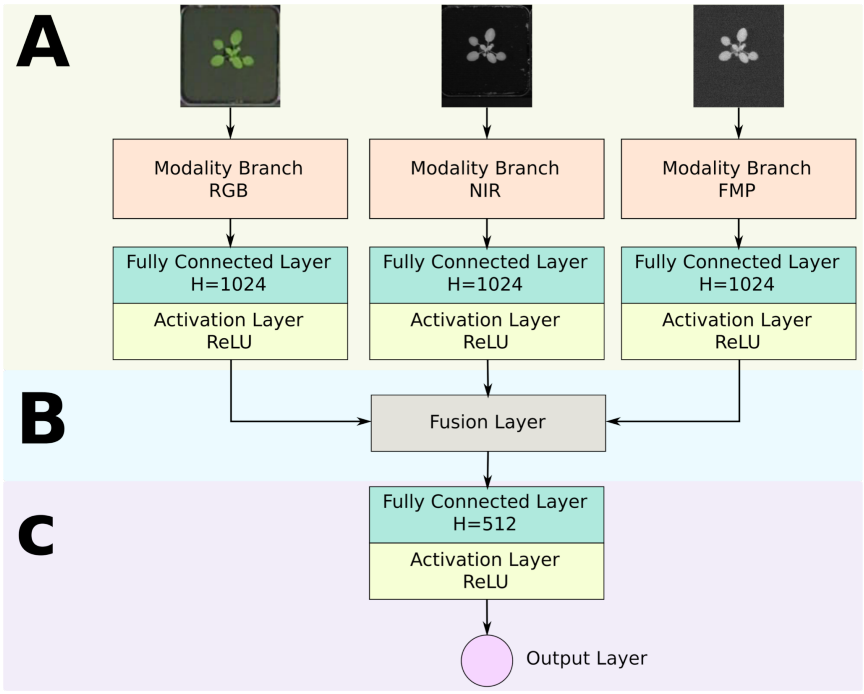
- Some examples…

Counting objects (old school)





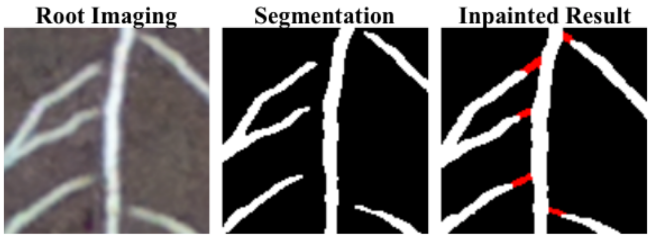
Counting objects (new school)

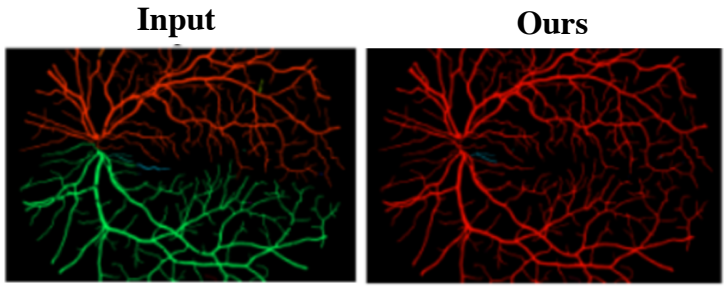


Giuffrida et al. The Plant Journal Pheno-Deep Counter: a unified and versatile deep learning architecture for leaf counting

Restoring faults in images

- **Recover gaps** from images of plant roots
- A similar problem is present in medical imaging as well
  ➔Retina fundus



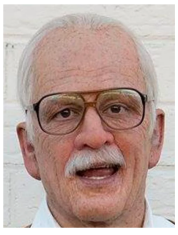


Chen et al. “Adversarial Large-scale Root Gap Inpainting ”, CVPRW - CVPPP 2019

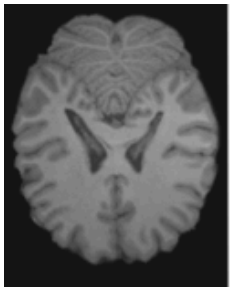
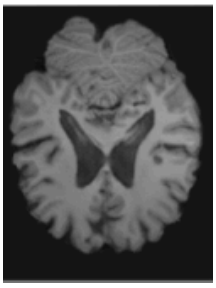
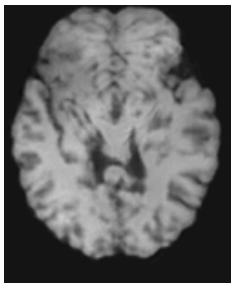
Learning to age

- How would I look in 30 years?



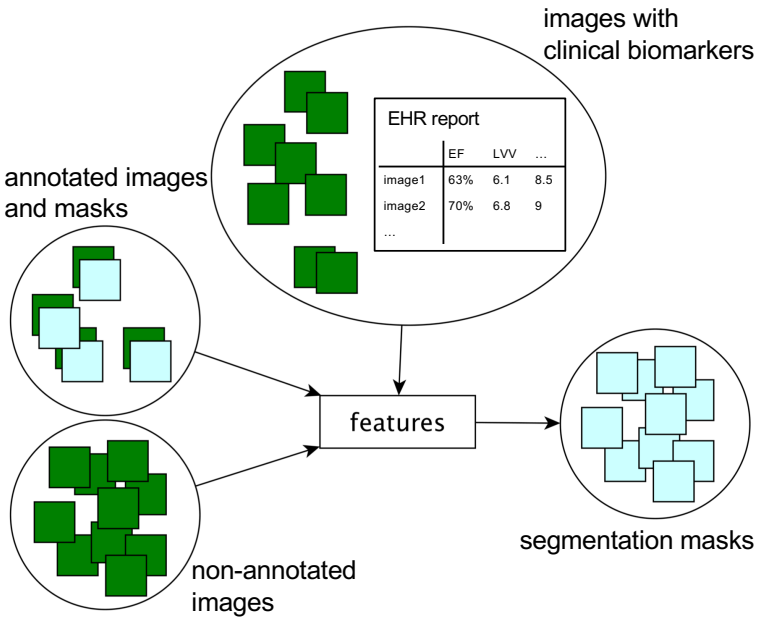


- A much harder task is how would my brain look?

Doing more with less

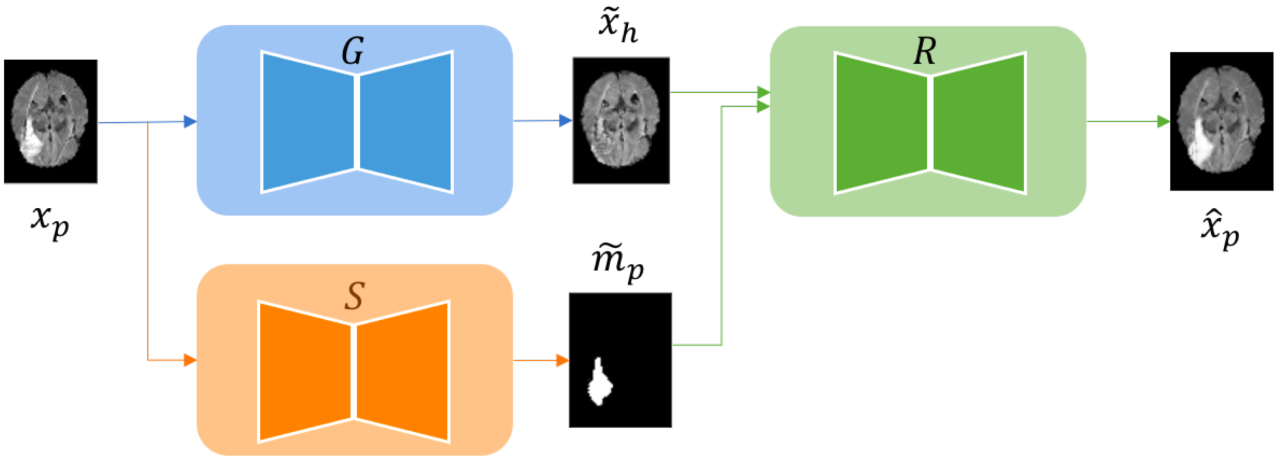
- Build systems that learn from few data, few annotations
  - Understand the world
  - Build intuition about the world



Chartsias et al. "Factorised Representation Learning in Cardiac Image Analysis", ArXiv 2019

Find anomalies in images?

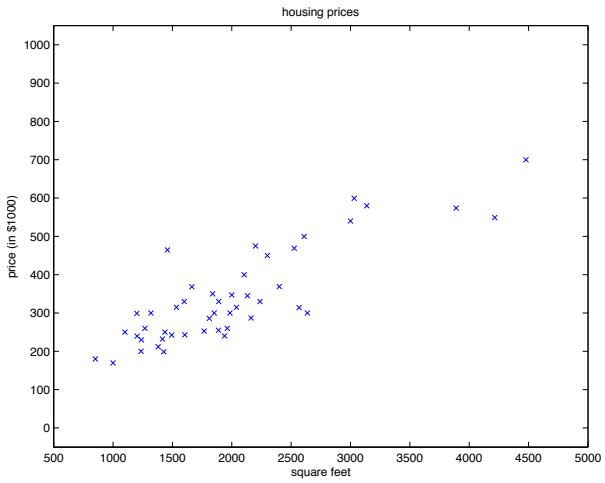
- Find **disease** and also create an artificial image that looks like as if it did not have disease

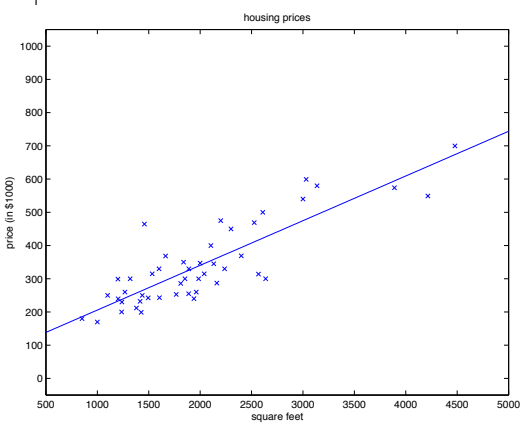


- The major directions of learning are:
  - **Supervised**: Patterns whose class is known a-priori are used for training.
  - **Unsupervised**: The number of classes/groups is (in general) unknown and no training patterns are available.
  - **Semisupervised**: A mixed type of patterns is available. For some of them, their corresponding class is known and for the rest is not.

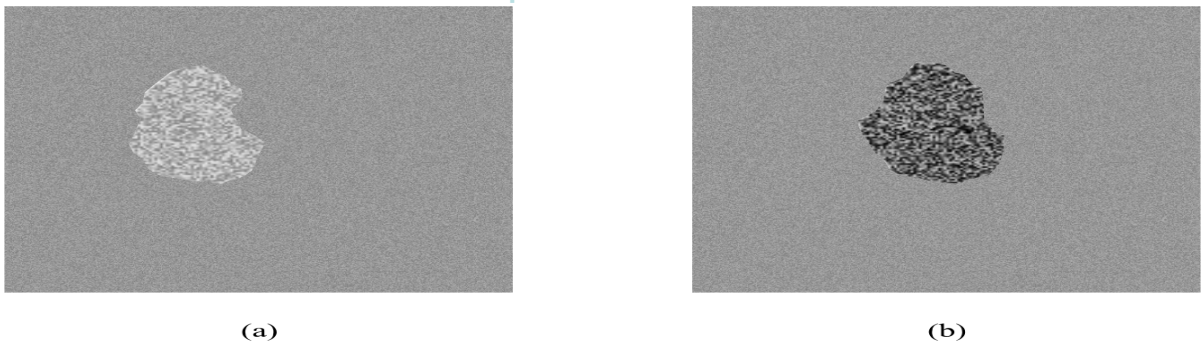
Supervised example

Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

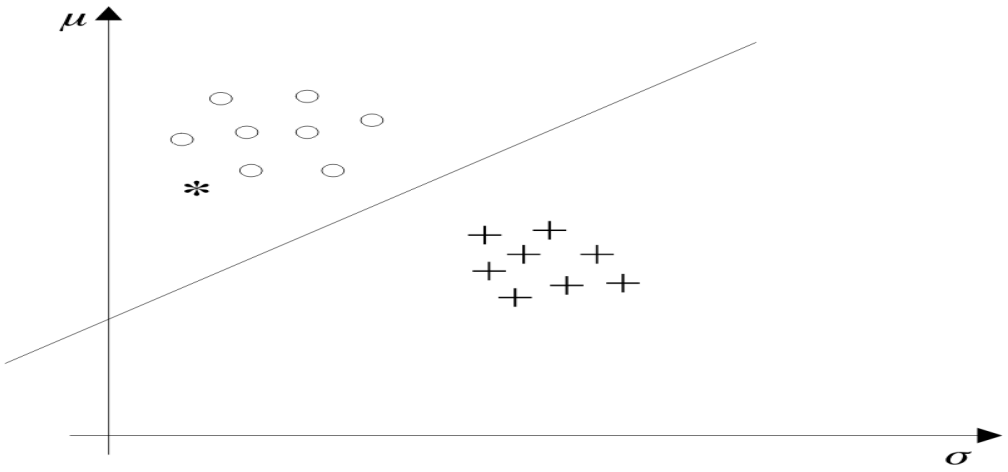




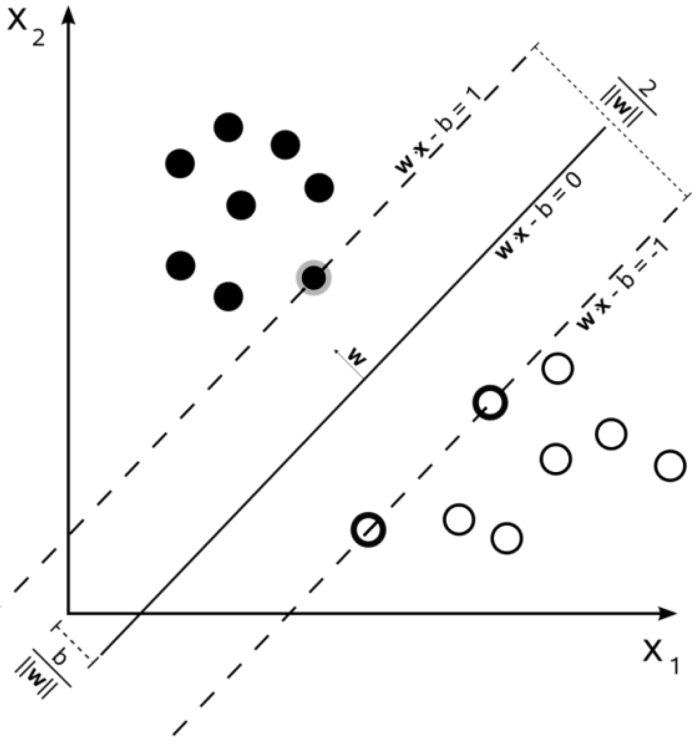
A classification example:



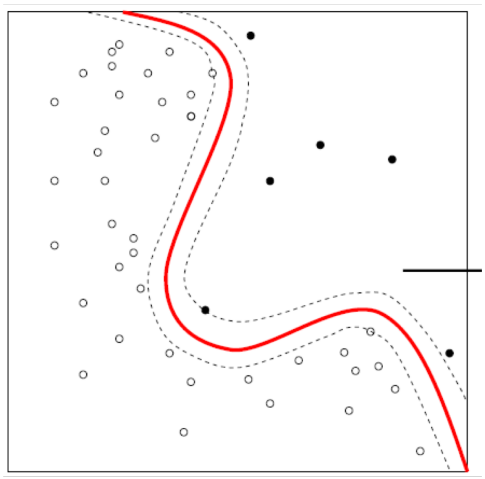
(a) (b)



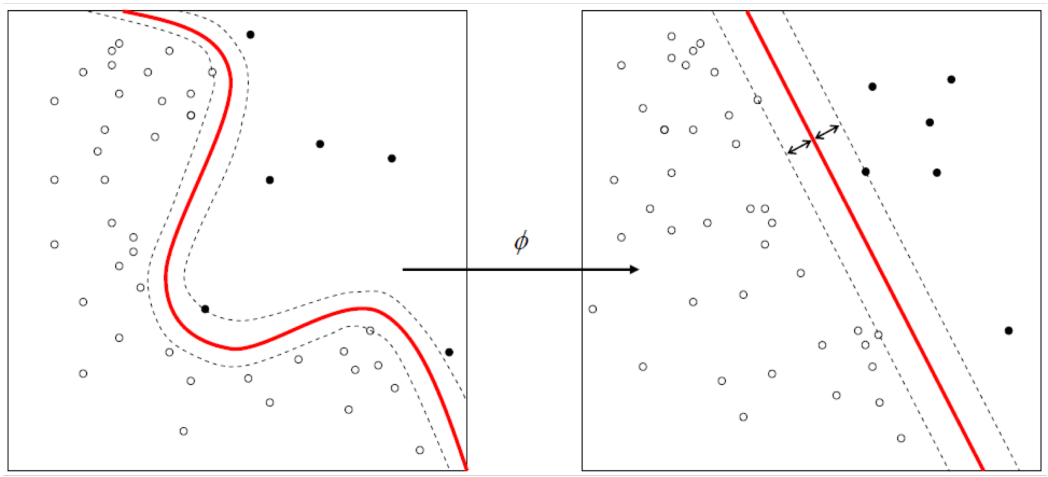
Another one



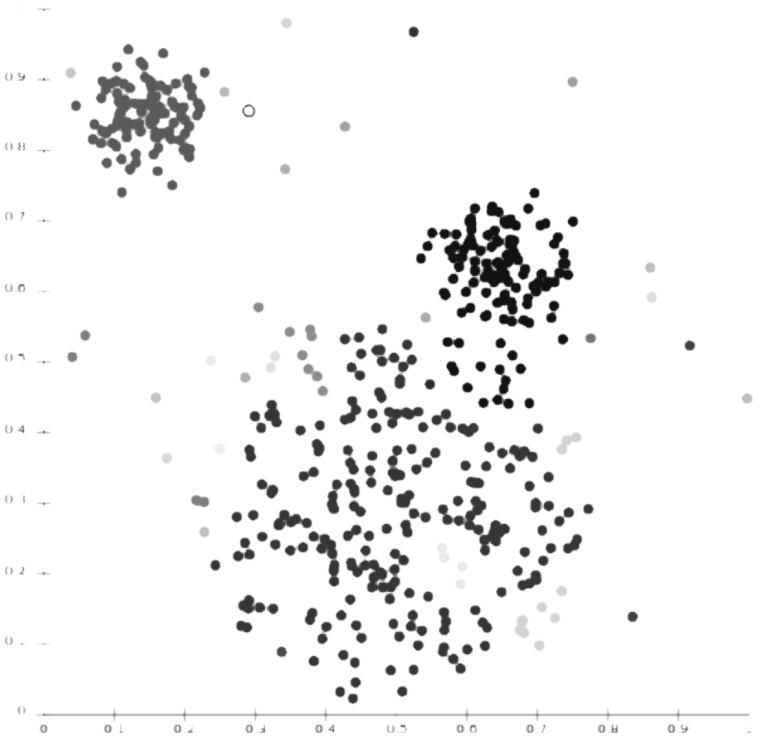
Hmm this is harder



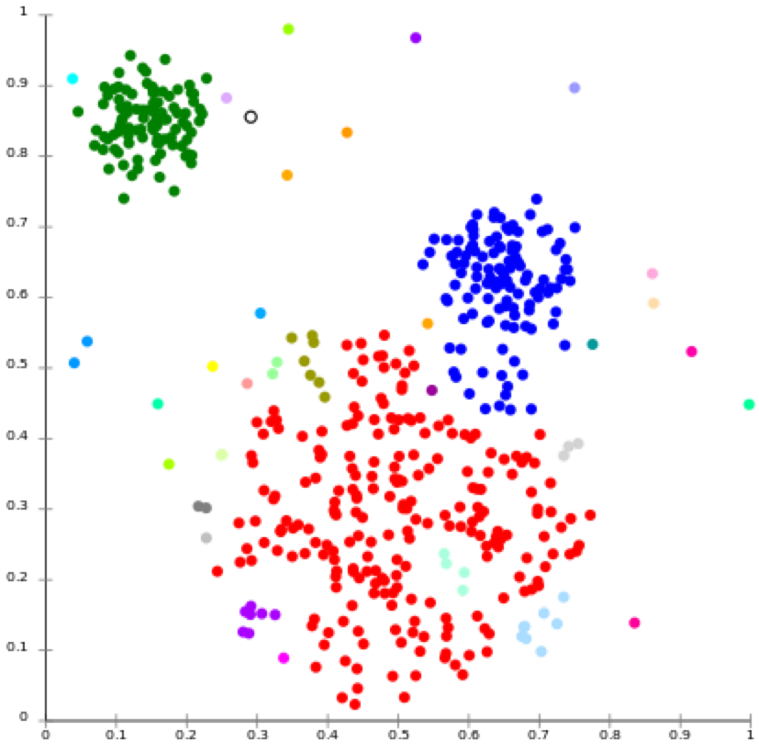
Hmm this is harder



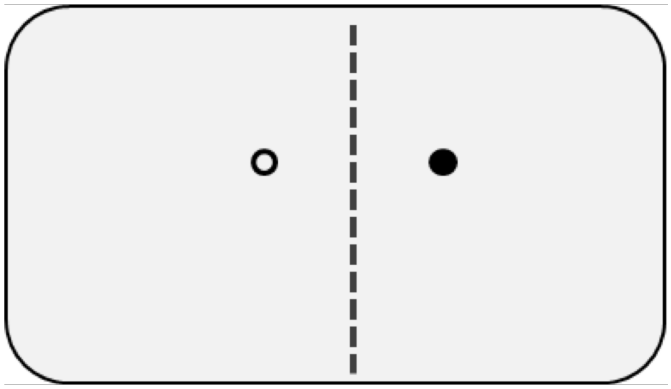
Unsupervised clustering



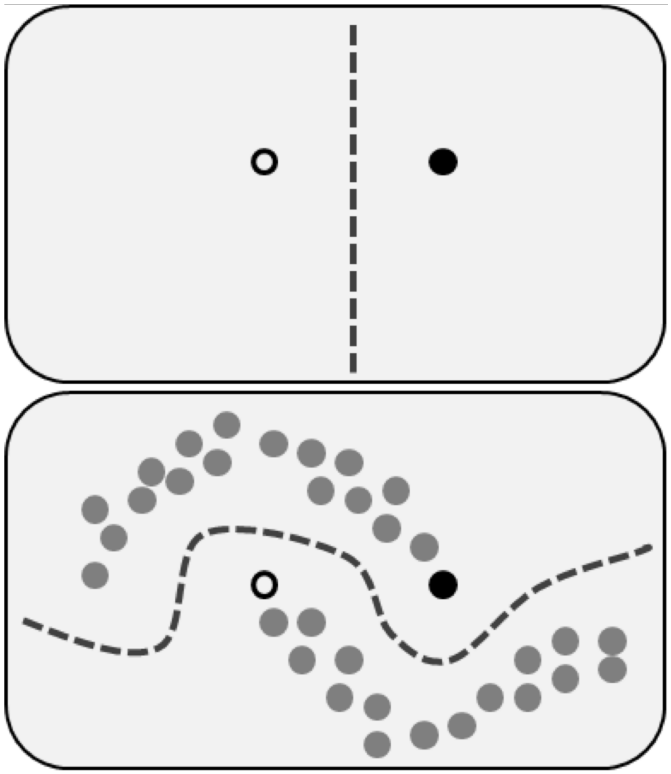
Unsupervised clustering



Semi-supervised learning



Semi-supervised learning

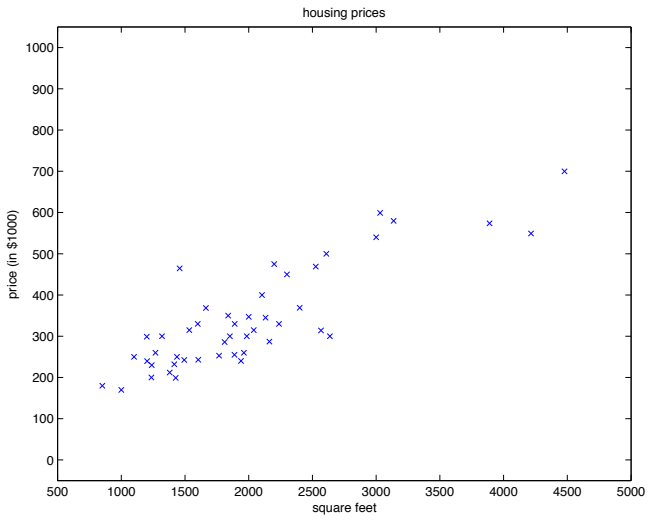


Topics that we will try to cover

- **Supervised** methods
  - Linear regression
  - Logistic regression
  - Lasso and elastic net regression
  - Support vector machines (linear and non linear, and even SVR)
  - Perceptron Classifier
- **Unsupervised** (and dimensionality reduction)
  - PCA, kernel PCA
- **Learning theory** (simple view)
- **Deep Neural Networks I:** Introduction; simple feed forward neural network architecture; how to train neural network; backpropagation theory; introduction to convolutional neural networks.

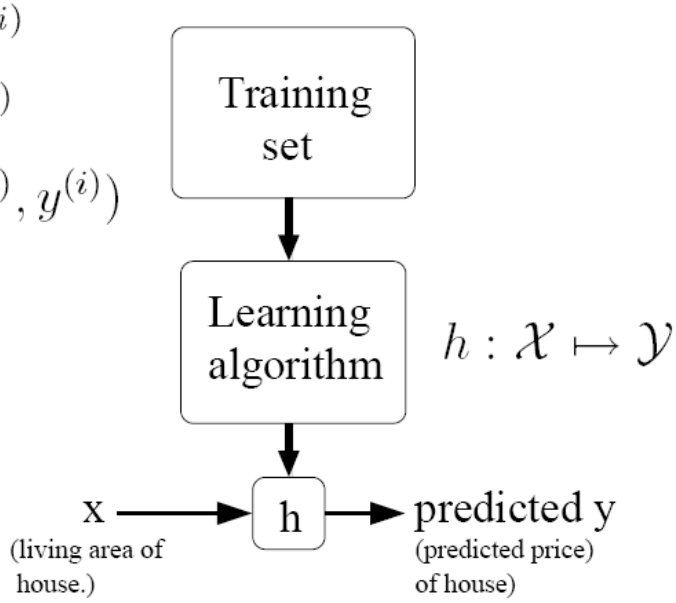
Supervised learning

Living area (feet^2)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮



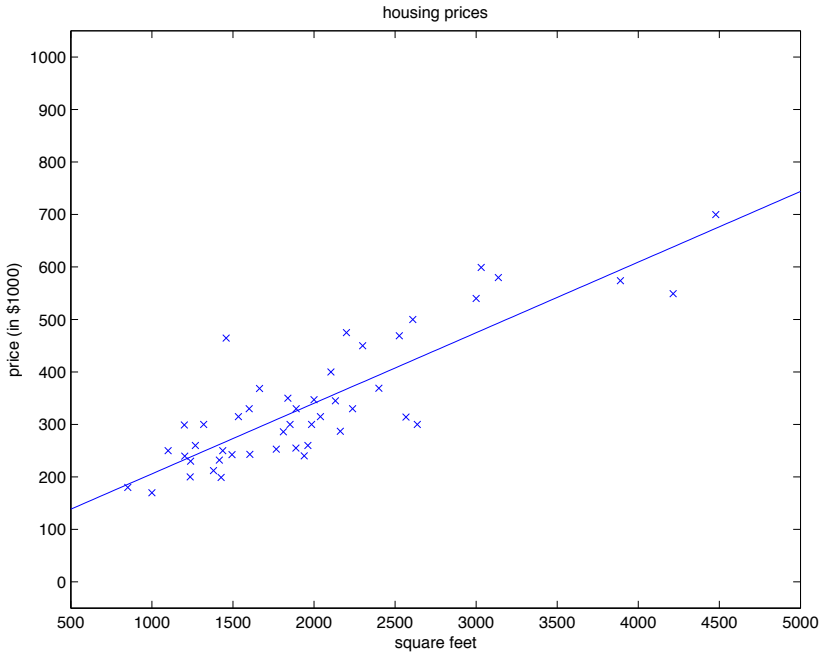
A typical formulation

- Input or features: $x^{(i)}$
- Output or target: $y^{(i)}$
- Training example: $(x^{(i)}, y^{(i)})$
- Training set== List of m examples: $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$
- Space: $\mathcal{X}$ $\mathcal{Y}$



Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
$\vdots$	$\vdots$	$\vdots$

Linear fit



Objective function:
How do we find the solution to this problem?

We are given some data. We are given a desired form of the line but we want to find the best line.

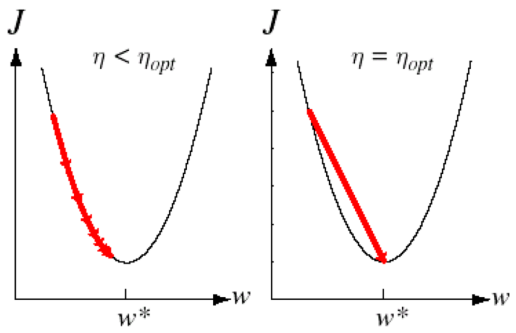
We need an objective function = training cost =measure of performance
That we can optimize.

Optimization

Many ML techniques need *optimization*, e.g.

- Minimizing error in a neural network/adaptive system
- Maximizing probability in Bayesian inference

From simple "steepest descent" to more advanced techniques (conjugate gradient,…)



Solving the linear regression problem

- The LMS algorithm

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 \qquad h(x) = \sum_{i=0}^{n} \theta_i x_i = \theta^T x,$$

- Define a cost: $J(\theta) = \frac{1}{2} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2.$

- Optimise for the cost $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta).$

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\
&= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\
&= (h_\theta(x) - y) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= (h_\theta(x) - y)\, x_j
\end{aligned}$$

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

For a dataset of size 1

Batch vs stochastic

- Batch

Repeat until convergence {

$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$ (for every j)

}

- Stochastic

Loop {

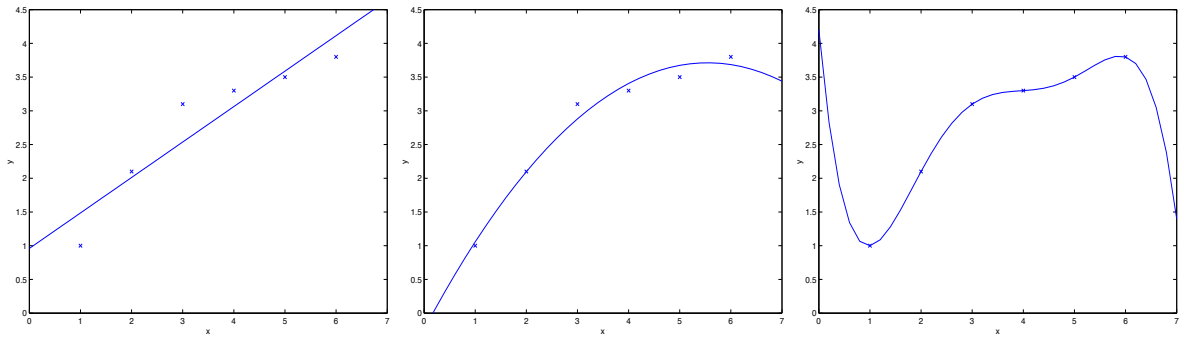
for i=1 to m, {

$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$ (for every j)

}

}

Higher model complexity



Model choice:
How do we tell what is the right choice?
Theoretical background: the bias variance dilemma
The practical solution: cross validation

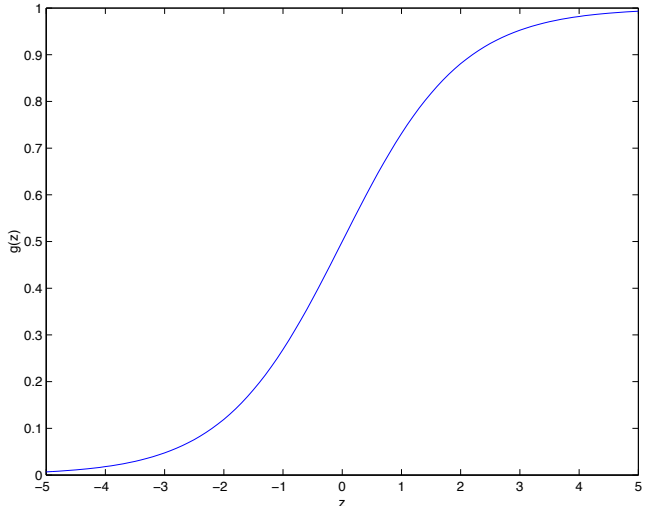
Logistic Regression

- Can we solve the classification problem as linear regression?

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

- The logistic function (sigmoid function) ➔

$$g(z) = \frac{1}{1 + e^{-z}}$$



Self-learning

Why?

- It has some nice property

$$
\begin{aligned}
g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\
&= \frac{1}{(1+e^{-z})^2}\left(e^{-z}\right) \\
&= \frac{1}{(1+e^{-z})}\cdot\left(1-\frac{1}{(1+e^{-z})}\right) \\
&= g(z)(1-g(z)).
\end{aligned}
$$

How to solve the problem (find the theta's)?

- A maximum likelihood view
- Assume

$$
\begin{aligned}
P(y=1 \mid x;\theta) &= h_\theta(x) \\
P(y=0 \mid x;\theta) &= 1-h_\theta(x)
\end{aligned}
$$

- Or…

$$p(y \mid x;\theta) = (h_\theta(x))^y\,(1-h_\theta(x))^{1-y}$$

- Assuming the training examples were generated independently

$$
\begin{aligned}
L(\theta) &= p(\vec{y} \mid X;\theta) \\
&= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)};\theta) \\
&= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}}\left(1-h_\theta(x^{(i)})\right)^{1-y^{(i)}}
\end{aligned}
$$

Objective function

$$
\begin{aligned}
\ell(\theta) &= \log L(\theta) \\
&= \sum_{i=1}^{m} y^{(i)}\log h(x^{(i)}) + (1-y^{(i)})\log(1-h(x^{(i)}))
\end{aligned}
$$

Contd...

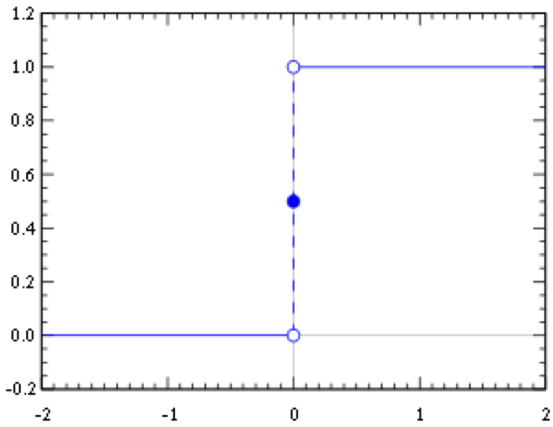
- Lets use gradient ascent

$$\theta := \theta + \alpha \nabla_\theta \ell(\theta)$$

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\
&= \left(y \frac{1}{g(\theta^T x)} - (1-y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\
&= \left(y(1 - g(\theta^T x)) - (1-y) g(\theta^T x) \right) x_j \\
&= \left(y - h_\theta(x) \right) x_j
\end{aligned}$$

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

What if we take an extreme sigmoidal

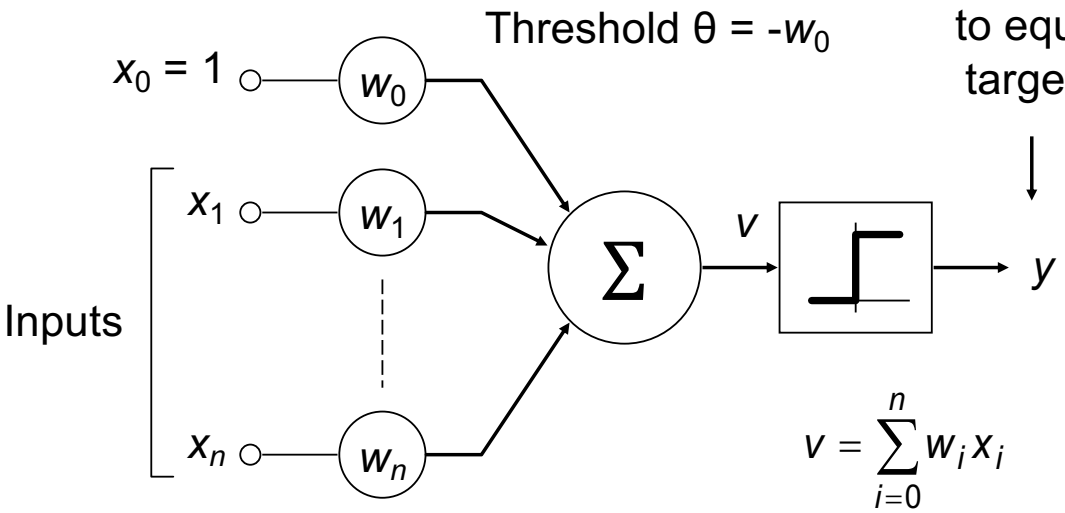


$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

The Perceptron: A Simple Learning Neuron

Rosenblatt (1958)

We want output y to equal target t

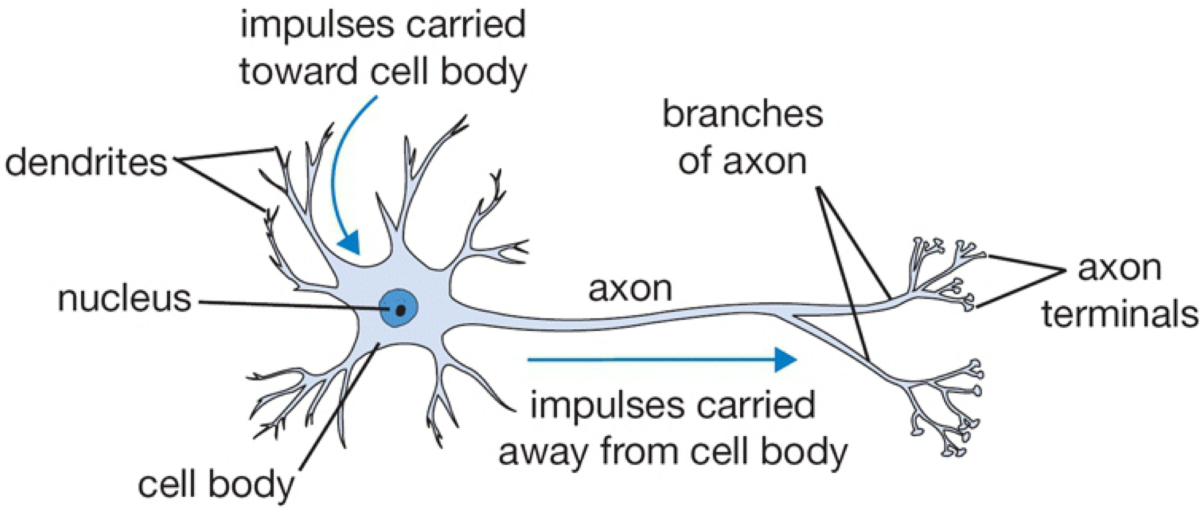


$$v = \sum_{i=0}^{n} w_i x_i$$

$$y = f(v)$$

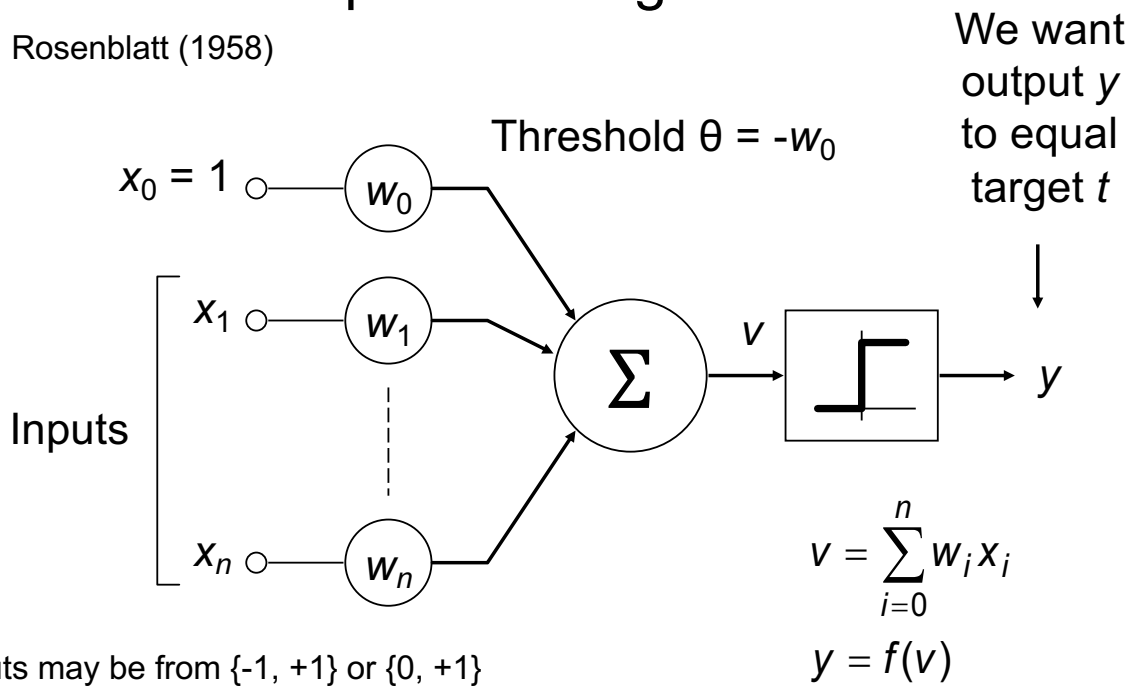
Inputs may be from {-1, +1} or {0, +1}

The Perceptron: A Simple Learning Neuron



The Perceptron: A Simple Learning Neuron

Rosenblatt (1958)



Inputs may be from {-1, +1} or {0, +1}

Perceptron Learning Algorithm

One example of a learning algorithm (presents samples one at a time)

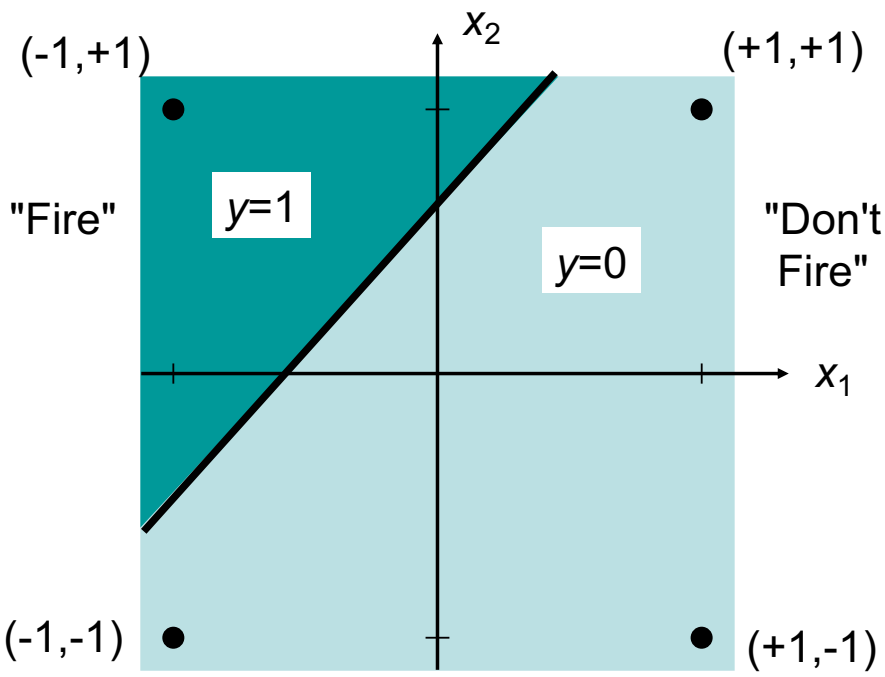
For all input vectors in training set:

1) Present input vector x
2) Calculate y=1 if $w^T x \geq 0$, y=0 if $w^T x < 0$
3) Compare y with target output t
   a) If t=1 but y=0, set new w = old w + ηx [punish]
   b) If t=0 but y=1, set new w = old w – ηx [punish]
   c) Otherwise (If y=t), do nothing [reward]

Repeat until correct for all input vectors.

Factor η is called the *learning rate*

Decision Boundary



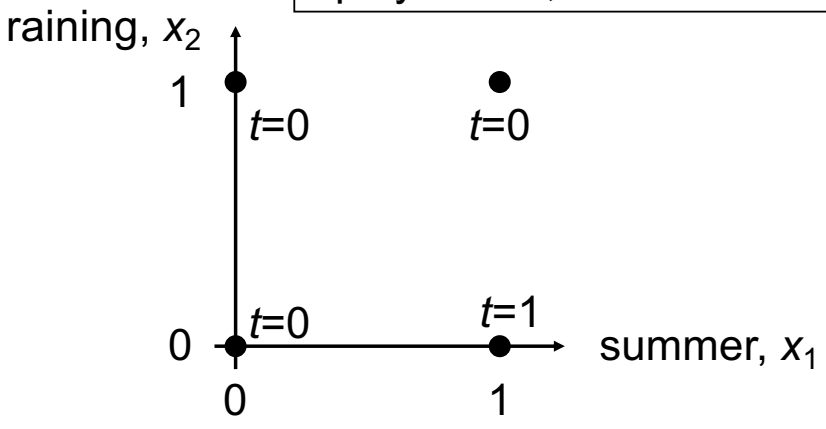
Self-learning

Simple Example

"If summer and not raining, play tennis"

Training set.
Specifies target t
for different inputs

(threshold, x_0	1	1	1	1)
summer, x_1	0	0	1	1
raining, x_2	0	1	0	1
play tennis, t	0	0	1	0



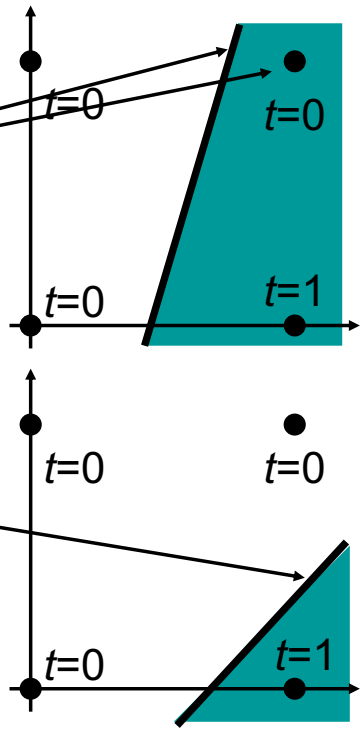
Self-learning

Simple Example (cont)

Suppose initially
$\mathbf{w} = (w_0, w_1, w_2) = (-0.5, +2.5, -1.5)$
Try input $\mathbf{x} = (1,1,1)$:
$\mathbf{w}^T\mathbf{x} = -0.5+2.5-1.5>0$
so y=1: Wrong

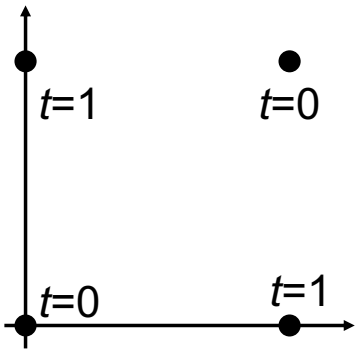
Using η=0.5,
subtract η**x** from **w** to give us
w = (-1.0, +2.0, -2.0)

Perceptron decision boundary is now correct for all inputs.



Perceptron Limitations

- Problem must be *linearly separable*
- Classic non-linearly separable problem: *XOR problem*



- Minsky & Pappert (1969) - conjectured this limitation would *not* be overcome.
- But it was…

From linear discriminants to GLD

Linear Discriminant Functions

Suppose we wished to decide whether some data

$$\mathbf{x} = (x_1, x_2, \ldots, x_d)$$

belonged to one of two categories

One way to do this is to construct a *Discriminant function.* Let g(**x**) define the categories as:

$$\omega(\mathbf{x}) = \begin{cases} \omega_1, g(\mathbf{x}) > 0 \\ \omega_2, g(\mathbf{x}) \leq 0 \end{cases}$$

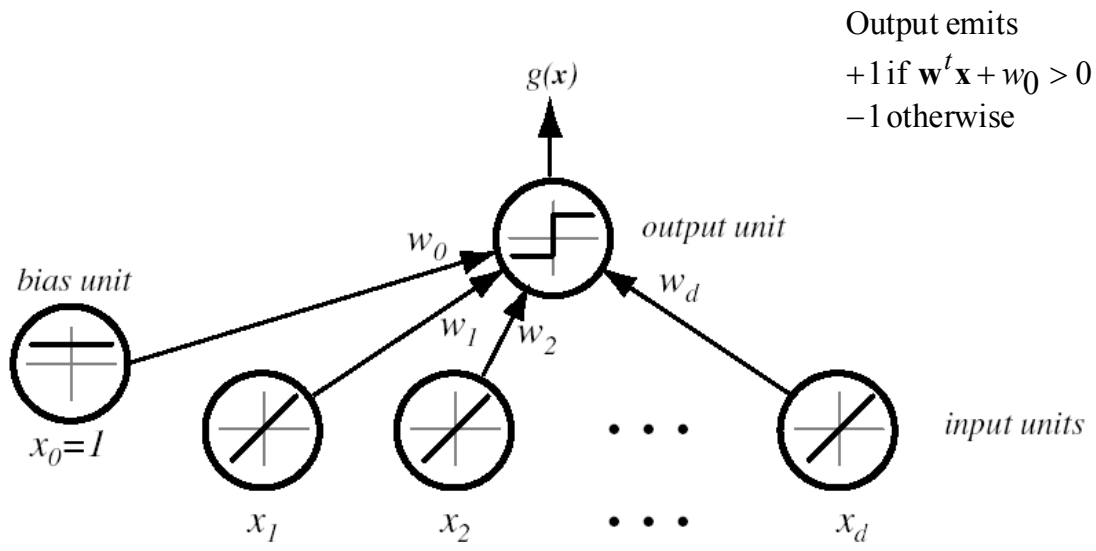
If g(x) is linear we can write:

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = \sum_{i=1}^{d} w_i x_i + w_0$$

where **w** = {w_1, … , w_d} are called the weights and w_0 is called the bias or threshold weight.

See ch 5.2 Linear Discriminant Functions
Duda Hart

Simple Linear Classifier



Each unit shows its effective input-output function.

Decision surface

g(**x**) = 0 defines a *decision surface* which separates points into ω_1 and ω_2. If g(**x**) is linear, this decision surface is a *hyperplane*.

The hyperplane divides the space into two regions:

R_1 : g(**x**) > 0, hence **x** is in ω_1 and

R_2 : g(**x**) ≤ 0, hence **x** is in ω_2

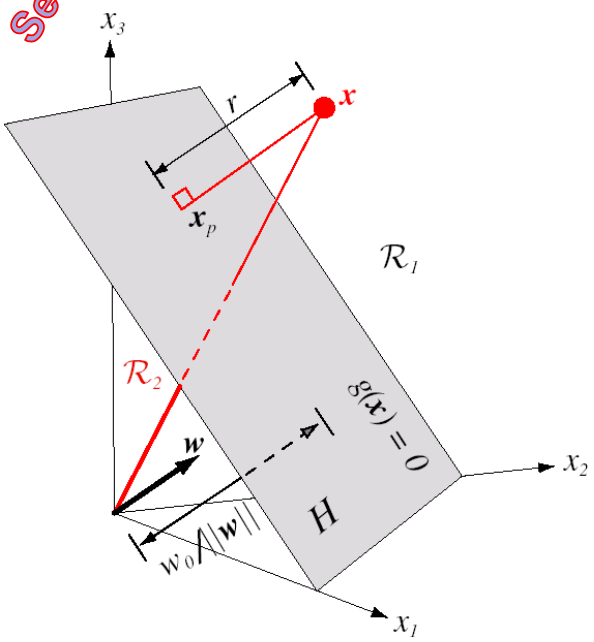
Suppose x_1 and x_2 are both on the decision surface. Then:

$$\mathbf{w}^t\mathbf{x}_1 + w_0 = \mathbf{w}^t\mathbf{x}_2 + w_0 \quad \text{i.e.} \quad \mathbf{w}^t(\mathbf{x}_1 - \mathbf{x}_2) = 0.$$

Therefore **w** is normal (orthogonal) to the hyperplane.

Self-learning

Hyperplane decision surface



Let us write

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$$

where $\mathbf{x}_p$ is normal projection

of $\mathbf{x} \rightarrow H$ and r is distance from H

($r > 0$ on $+$ve side, $r < 0$ on -ve)

Since $g(\mathbf{x}_p) = 0$,

$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = \mathbf{w}^t(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}) + w_0 =$$

$$= \ldots = r\|\mathbf{w}\|$$

$$\Rightarrow r = g(\mathbf{x})/\|\mathbf{w}\|$$

$g(\mathbf{x})$ measures dist from $\mathbf{x}$ to H.

Self-learning

Generalized Linear Discriminants

We can generalize g(**x**) by adding terms x_ix_j to give a quadratic (nonlinear) discriminant function:

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i + \sum_{i=1}^{d}\sum_{j=1}^{d} w_{ij} x_i x_j$$

Can generalize to cubic, etc.

We can view this as a linear discriminant function in a **new space**.

Let $y_i(\mathbf{x})$ define a new variable as a (nonlinear) function of **x**.

$$g(\mathbf{x}) = \sum_{i=1}^{\hat{d}} a_i y_i(\mathbf{x})$$

Note we have absorbed the bias weight in this formulation - A process called *augmentation*.

See ch 5.3 Generalized Linear Discriminant Functions
Duda Hart

Quadratic discriminant: 1-d example

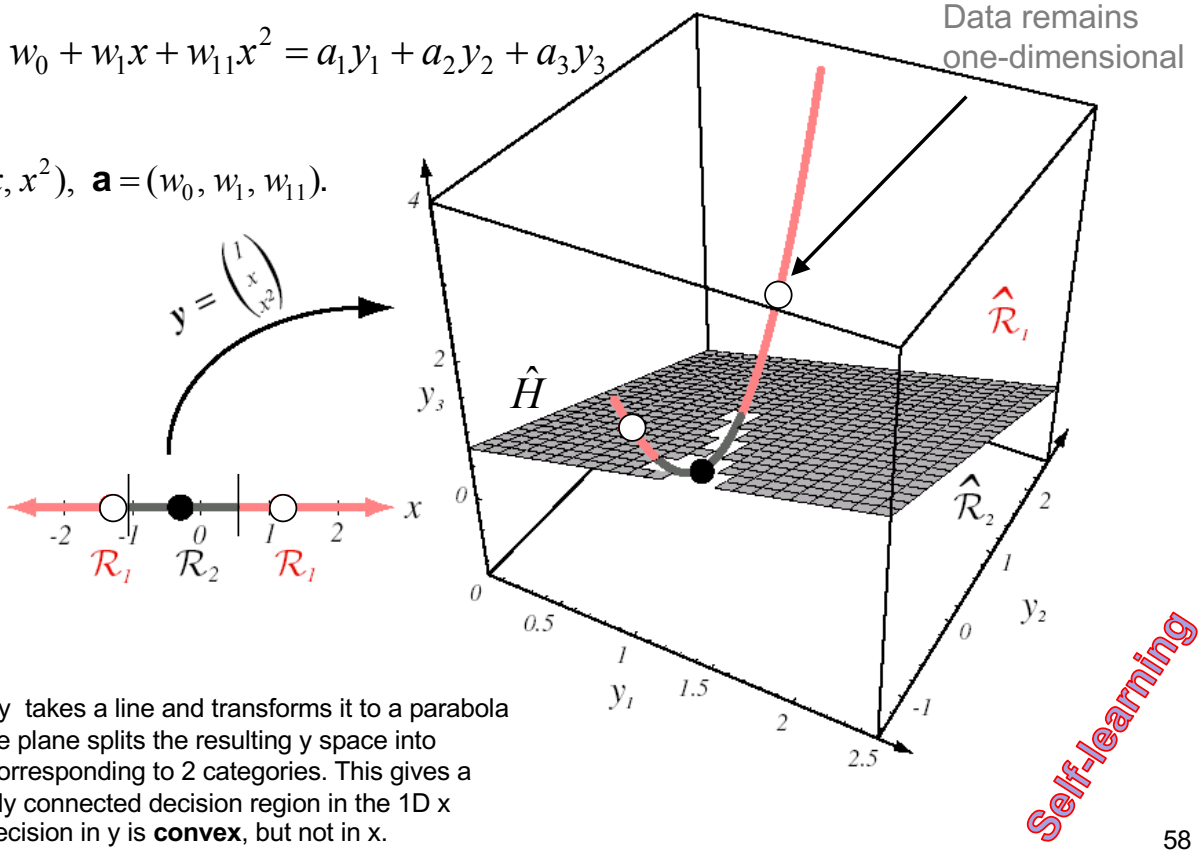
e.g.

$$g(\mathbf{x}) = w_0 + w_1 x + w_{11} x^2 = a_1 y_1 + a_2 y_2 + a_3 y_3$$

with

$$\mathbf{y} = (1, x, x^2), \quad \mathbf{a} = (w_0, w_1, w_{11}).$$

Data remains one-dimensional



Mapping y takes a line and transforms it to a parabola in 3D. The plane splits the resulting y space into regions corresponding to 2 categories. This gives a non-simply connected decision region in the 1D x space. Decision in y is **convex**, but not in x.

Self-learning

Linearly Separable Case (2 category)

Suppose we have n samples $y_1,\ldots y_n$, each labelled either ω_1 or ω_2 and we wish to *learn* a discriminant function $g(\mathbf{y}) = \mathbf{a}^T\mathbf{y}$, that correctly classifies the data.

Sample $\mathbf{y}_i$ is correctly classified if

$$\mathbf{a}^T\mathbf{y}_i > 0 \quad \text{when } \mathbf{y}_i \text{ is labelled } \omega_1$$

or

$$\mathbf{a}^T\mathbf{y}_i < 0 \text{ when } \mathbf{y}_i \text{ is labelled } \omega_2$$

A data set is called *linearly separable* if there exists a vector a which correctly classifies all samples. This is called a *separating vector* or *solution vector.*

[The case of $g(\mathbf{y}) = \mathbf{a}^T\mathbf{y}+a_0$ with augmentation: $\mathbf{y}' \equiv [\mathbf{y},1]^T, \mathbf{a}' \equiv [\mathbf{a}, a_0]^T$]

See Ch 5.4 The Two-Category Linearly-Separable Case Duda Hart

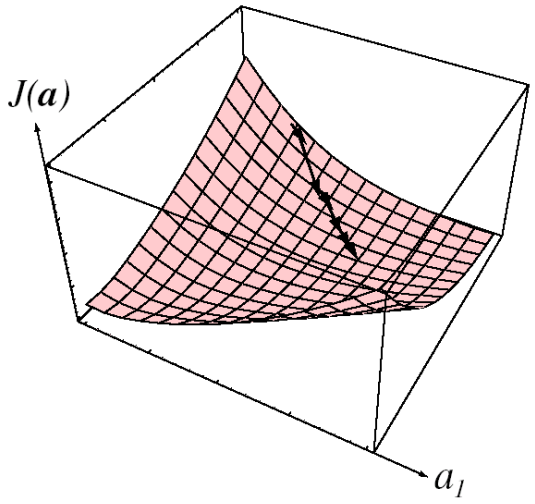
How to find **a**

- Great now we have understood what the **a** must satisfy (linear inequalities) and some properties.
- **However, we still do not know how to find a**
- We are given some data and their labels and we need a procedure to find a
- We need to find a criterion that when optimized we have a solution vector **a**.
  - Lets call this criterion **J(a)**. Observe it is a scalar function of **a : returns a value** pending on **a**
  - If we make good choices of J() we can use **optimization** theory. [We will talk about this a lot later in the class.]
  - Learning a classifier is then reduced to an optimization problem
  - We will consider for now a simple approach: Gradient Descent

Gradient Descent

Simple concept: Consider I am at some point in my function **$J(\mathbf{a}_k)$**. I need to move to a new point **$\mathbf{a}_{k+1}$. What is a good point?**

Gradient: Gradient points in the direction of the greatest rate of increase of the function. (Generalization of derivative in multivariate functions)

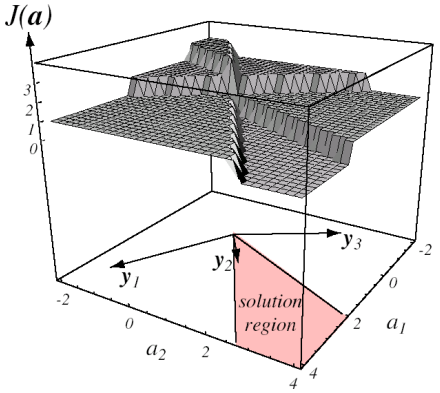


Update : $\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k)\nabla J(\mathbf{a}(k))$
Learning rate $\eta(k)$
Gradient vector $\nabla J(\cdot)$

Basic gradient descent algorithm :
1. Initialize $k \leftarrow 0, \mathbf{a}(1), \theta, \eta(.)$
2. $k \leftarrow k+1$
3. $\mathbf{a} \leftarrow \mathbf{a} - \eta(k)\nabla J(\mathbf{a})$
4. If $|\eta(k)\nabla J(\mathbf{a})| > \theta$, repeat from 2.

Perceptron Revisited

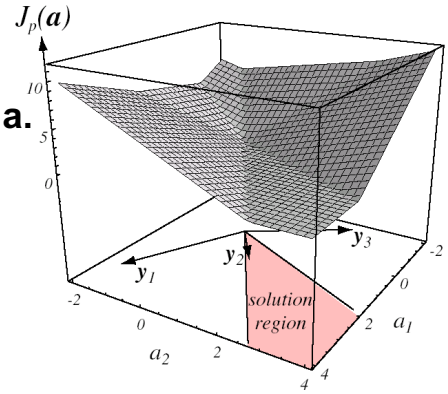
Could J = number of samples misclassified?
No – the function is piecewise constant.
Why is this bad? The cost function has 0 gradient at most points and wherever non-zero has discontinuities. We need a better cost.



The Perceptron

$$J_p(\mathbf{a}) = \sum_{\mathbf{y} \in Y_m} (-\mathbf{a}^T \mathbf{y})$$

warning: Data are assumed normalized



where Y_m is the set of samples misclassified by **a**.

When is $J_p(a)=0$?

Is $J_p(a)$ always positive?

Perceptron Algorithm

$$[\nabla J_p(\mathbf{a})]_i = \partial J_p(\mathbf{a}) / \partial a_i = \sum_{\mathbf{y} \in Y_m} (-y_i)$$

warning: Data are assumed normalized

i.e. $\nabla J_p(\mathbf{a}) = \sum_{\mathbf{y} \in Y_m} (-\mathbf{y})$ so gradient descent rule is :

$$\mathbf{a}(k+1) = \mathbf{a}(k) - \eta(k) \sum_{\mathbf{y} \in Y_m} (-\mathbf{y})$$

where Y_m is set of samples misclassified by $\mathbf{a}(k)$.

Batch Perceptron Algorithm

1. Initialize $\mathbf{a}, \eta(\cdot)$, stopping criterion $\theta, k \leftarrow 0$
2. $k \leftarrow k+1$
3. $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \sum_{\mathbf{y} \in Y_m} \mathbf{y}$
4. If $|\eta(k) \sum_{\mathbf{y} \in Y_m} \mathbf{y}| > \theta$, repeat from 2.

η(k) is the learning rate or step size

Multilayer Neural Networks

Linear discriminants are good for many problems but not **general** enough for demanding applications.

We can get more complex decision surfaces with nonlinear pre-processing,

$$y_i = \varphi_i(\mathbf{x})$$

Where $\varphi_i(.)$, is, for example, a polynomial expansion to some order k.

But **too many free** parameters, so we may not have enough data points to fix them. ➔ *learn* which nonlinearities to use.

The best-known method is based on gradient descent: the so-called *backpropagation* algorithm.

Three-Layer Network

Network has:

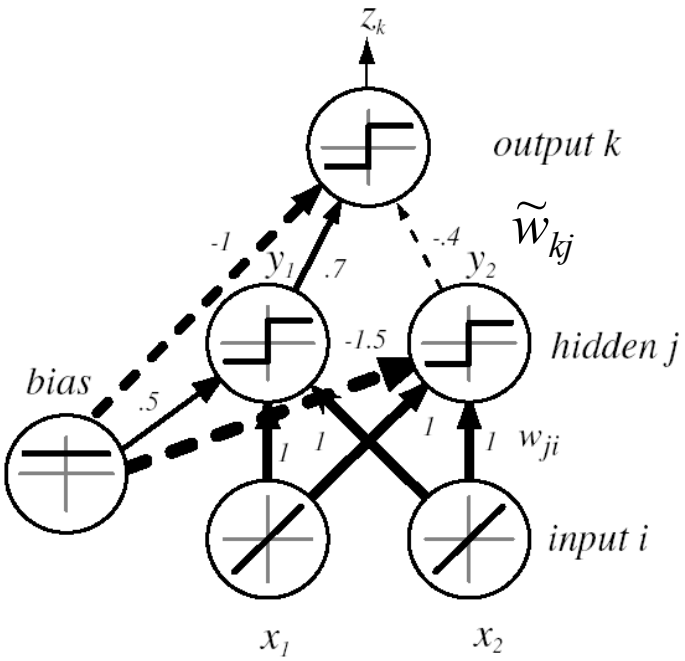
- Input layer
- Hidden layer
- Output layer

with adjustable weights between layers

Also:

- Bias unit

with weights to all hidden and output units.



Biological terms sometimes used:
"neuron" = unit; "synapses" = connection; "synaptic weight" = weight.

Operation

Step 1: each d-dimensional input vector $(x_1,\ldots,x_d)$ is presented to input layer of the network and augmented with a bias term $x_0 = 1$ to give $x = (x_0, x_1,\ldots,x_d)$

Step 2: at each hidden layer we calculate the weighted sum of inputs to give the net activation:

$$net_j = \sum_{i=1}^{d} x_i w_{ji} + w_{j0} = \sum_{i=0}^{d} x_i w_{ji} = \mathbf{w}_j^T \mathbf{x}$$

where w_{ji} is the weight from the input unit *i* to the hidden unit *j*

Step 3: The hidden unit emits the output: $y_j = f(net_j)$

where *f*(.) is some nonlinear *activation function*

Operation (cont)

Step 4: At each output unit we calculate the weighted sum of the hidden layer units it is connected to giving:

$$\widetilde{net}_k = \sum_{j=1}^{n_H} y_j \widetilde{w}_{kj} + \widetilde{w}_{k0} = \sum_{j=0}^{n_H} y_j \widetilde{w}_{kj} = \widetilde{\mathbf{w}}_k^T \mathbf{y}$$

where $\widetilde{w}_{kj}$ is the weight from the hidden unit *j* to the output unit *k*

Step 5: each output unit emits $z_k = f(\widetilde{net}_k)$

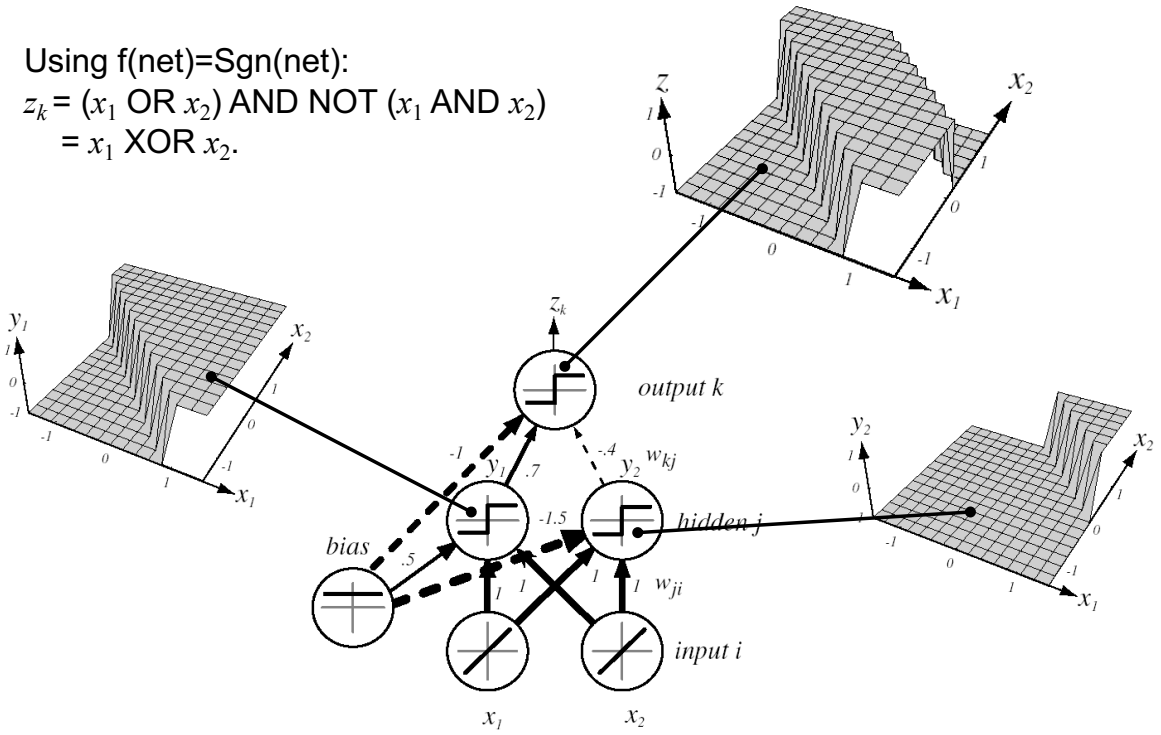
where *f*(.) is again the nonlinear *activation function*

We can therefore think of the network as calculating c discriminant functions:

$$z_k = g_k(\mathbf{x})$$

Example: XOR Problem

Using f(net)=Sgn(net):
z_k = (x_1 OR x_2) AND NOT (x_1 AND x_2)
= x_1 XOR x_2.



General Feedfoward Operation

General form of output discriminant functions:

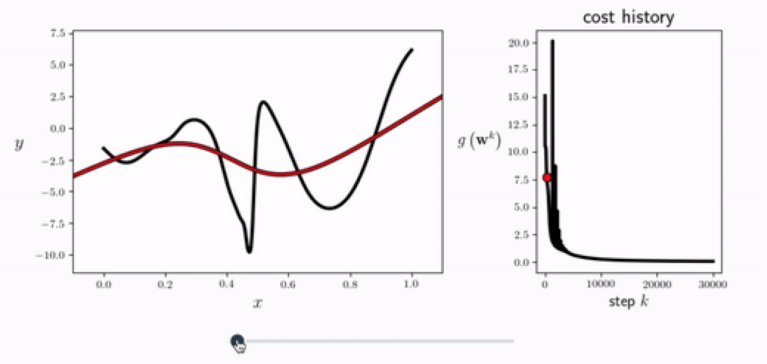
$$g_k(\mathbf{x}) \equiv z_k = f\left(\sum_{j=1}^{n_H} \tilde{w}_{kj} f\left(\sum_{i=1}^{d} w_{ji} x_i + w_{j0}\right) + \tilde{w}_{k0}\right)$$

[Note that the $\tilde{w}_{kj}$s are different from the w_{ji}s.]

Question: can every decision be implemented with a 3-layer network?

Answer [Kolmogorov + others]: (in theory) Yes.

Typically needs very nasty functions at each layer.



See 6.2 and onwards
Duda Hart

Backpropagation

We have seen that we can *approximate* any function but how do we ***learn*** functions?

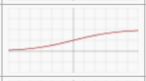
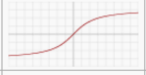
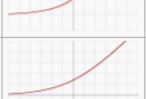
Perceptrons (single layer network): each input affects the output via its weight: so we know which weight to change to reduce errors.

Multilayer networks (a.k.a *multilayer perceptron*, *MLP*): hidden units have no "teacher" – so how reduce the error?

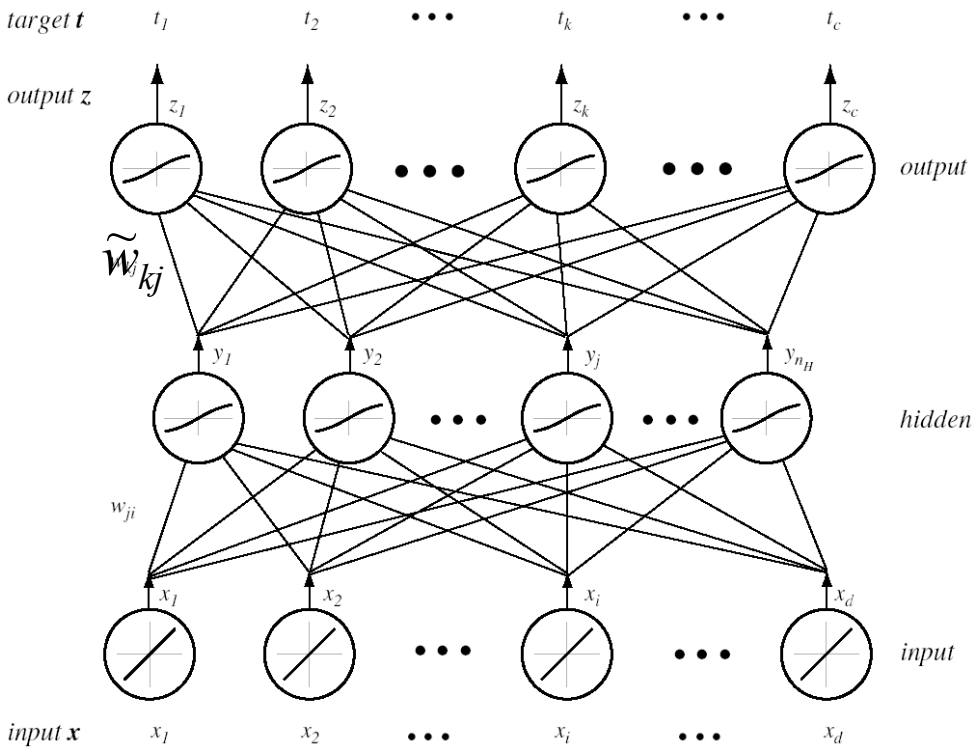
This is known as the *credit assignment problem*.

Backpropagation solves the credit assignment problem using a smooth activation function *f(.)* and gradient descent.

Some known non-linearities

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Backpropagation Network



Backpropagation Outline

Step 1: start with untrained network (random weights)

Step 2: present training data to network and calculate outputs: **z**(n) = g(**x**(n))

This generates a training error:

$$J(\mathbf{w}) = \tfrac{1}{2}\sum_{n}\sum_{k=1}^{c}(t_k(n) - z_k(n))^2 = \tfrac{1}{2}\sum_{n}\| \mathbf{t}(n) - \mathbf{z}(n) \|^2$$

Where **w** represents the set of all weights in the network and **t**(n) are the target outputs

Step 3: change the weights in the direction of the negative gradient:

$$\Delta\mathbf{w} = -\eta\frac{\partial J}{\partial \mathbf{w}} \qquad \text{or} \qquad \Delta w_{pq} = -\eta\frac{\partial J}{\partial w_{pq}}$$

where η is the learning rate (step size).

Step 4: iterate until convergence

See 6.3
Duda Hart

Self-learning

Backpropagation of Errors

Let us calculate the gradient for a 3-layer network. We work backwards, starting at the hidden-to-output weights.

The *chain rule* gives:

$$\frac{\partial J}{\partial \tilde{w}_{kj}} = \sum_n \frac{\partial J}{\partial n\tilde{e}t_k(n)} \frac{\partial n\tilde{e}t_k(n)}{\partial \tilde{w}_{kj}} = -\sum_n \tilde{\delta}_k(n) \frac{\partial n\tilde{e}t_k(n)}{\partial \tilde{w}_{kj}}$$

where $\tilde{\delta}_k(n) = -\partial J / \partial n\tilde{e}t_k(n)$ is termed the *sensitivity* of *J* to the net activation of *k*. Applying the *chain rule* again:

$$\tilde{\delta}_k(n) = -\frac{\partial J}{\partial n\tilde{e}t_k(n)} = -\frac{\partial J}{\partial z_k(n)} \frac{\partial z_k(n)}{\partial n\tilde{e}t_k(n)} = \underbrace{(t_k(n) - z_k(n))}_{\text{error}} f'(n\tilde{e}t_k(n))$$

where *f'(.) is the derivative of f(.).* Noting that:

$$n\tilde{e}t_k(n) = \sum_j \tilde{w}_{kj} y_j(n) \Rightarrow \frac{\partial n\tilde{e}t_k(n)}{\partial \tilde{w}_{kj}} = y_j(n)$$

Hence:

$$\Delta \tilde{w}_{kj} = -\eta \partial J / \partial \tilde{w}_{kj} = \eta \sum_n \tilde{\delta}_k(n) y_j(n) = \eta \sum_n (t_k(n) - z_k(n)) f'(n\tilde{e}t_k(n)) y_j(n)$$

Self-learning

Backpropagation to Hidden Units

Again using the chain rule for input-to-hidden units we have

$$\frac{\partial J}{\partial w_{ji}} = \sum_n \frac{\partial J}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial net_j(n)} \frac{\partial net_j(n)}{\partial w_{ji}(n)}$$

Note that $\partial J / \partial y_j(\mathrm{n})$ involves all outputs *k*

$$\frac{\partial J}{\partial y_j(n)} = \frac{\partial}{\partial y_j(n)} \left[\frac{1}{2} \sum_{k=1}^{c} (t_k(n) - z_k(n))^2 \right] = -\sum_{k=1}^{c} (t_k(n) - z_k(n)) \frac{\partial z_k(n)}{\partial y_j(n)}$$

$$= -\sum_{k=1}^{c} (t_k(n) - z_k(n)) \frac{\partial z_k(n)}{\partial n\tilde{e}t_k(n)} \frac{\partial n\tilde{e}t_k(n)}{\partial y_j(n)} = -\sum_{k=1}^{c} (t_k(n) - z_k(n)) f'(n\tilde{e}t_k(n)) \tilde{w}_{kj}$$

$$= -\sum_{k=1}^{c} \tilde{\delta}_k(n) \tilde{w}_{kj}(n)$$

since $\tilde{\delta}_k(n) = (t_k(n) - z_k(n)) f'(n\tilde{e}t_k(n))$

Self-learning

Hidden Units (cont)

We also have $\partial y_j(n)/\partial net_j(n) = f'(net_j(n))$ so we can define

$$\delta_j(n) \equiv -\frac{\partial J}{\partial net_j(n)} = -\frac{\partial J}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial net_j(n)} = f'(net_j(n))\sum_{k=1}^{c}\tilde{w}_{kj}\tilde{\delta}_k(n)$$

as the *hidden unit sensitivities*. Note that the output sensitivities are propagated back to the hidden unit sensitivites – hence *"back-propagation of errors"*

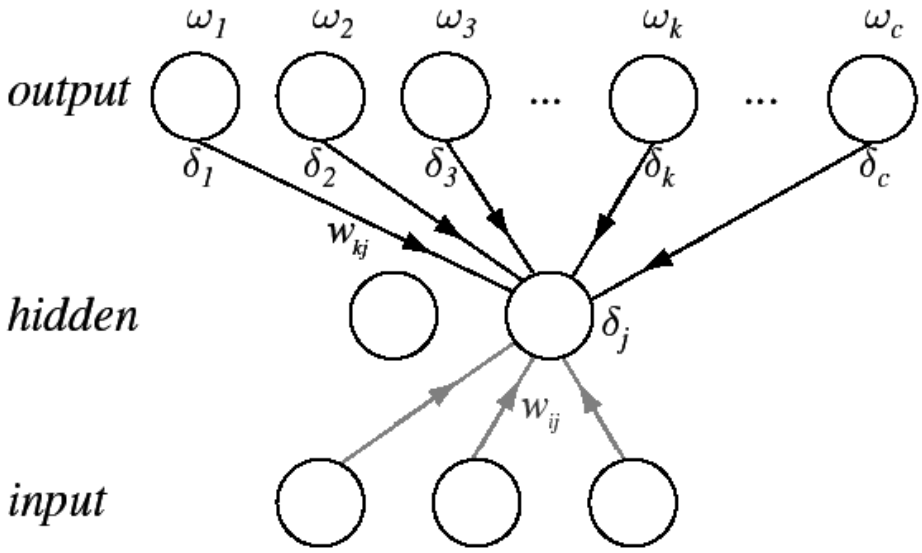
Finally we have:

$$\partial net_j(n)/\partial w_{ji} = x_i(n)$$

So the update rule for the input-to-hidden weights is:

$$\Delta w_{ji} = -\eta\frac{\partial J}{\partial w_{ji}} = \eta\sum_n x_i(n)\delta_j(n) = \eta\sum_n\left[\sum_{k=1}^{c}\tilde{w}_{kj}\tilde{\delta}_k(n)\right]f'(net_j(n))x_i(n)$$

Backwards propagation



The backpropagation algorithm can be easily generalized to any network with feed-forward connections, e.g. more layers, different nonlinearities in different layers, etc.

Criticisms of MLPs

MLPs provide one way to achieve the required expressive power needed to build general classifiers. However they do have their weaknesses:

- Possibility of multiple (local) minima
- g(**x**) is nonlinear in terms of the weights: this makes training slow.
- *ad hoc* solution (how many units, hidden layers, etc?)

Other popular discriminant learning structures:

- Support Vector Machines (SVMs)
- Convolutional Networks (CNNs)

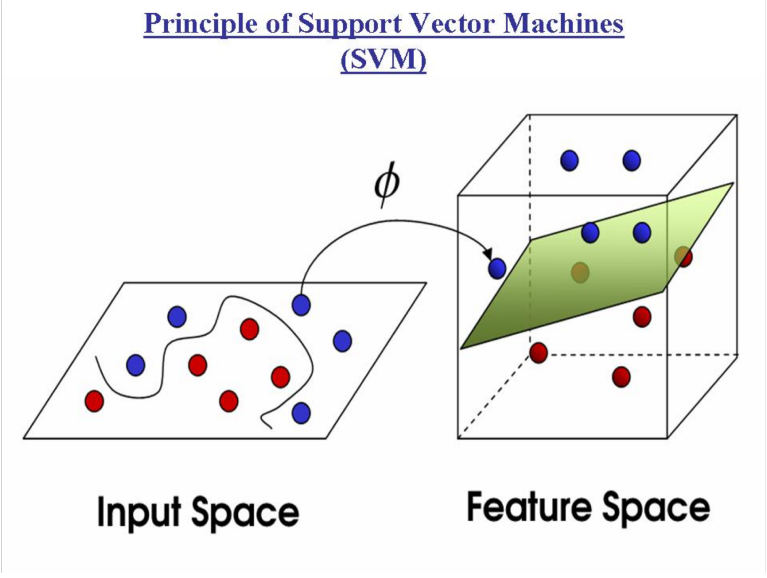
TensorFlow Demo

- Start with the bottom left (linear activation)
  - Observations: linearly separable
  - Start with a very simple net 1 layer 1 node (linear activations)
- Move to upper right still separable but not linear in input space
  - Options to fix: change layer size, add layers, change activation, change inputs?

Support Vector Machines

Support Vector Machines

- SVMs solve a problem in linear or non-linear space by projecting the input space into a new (possibly infinite) space.

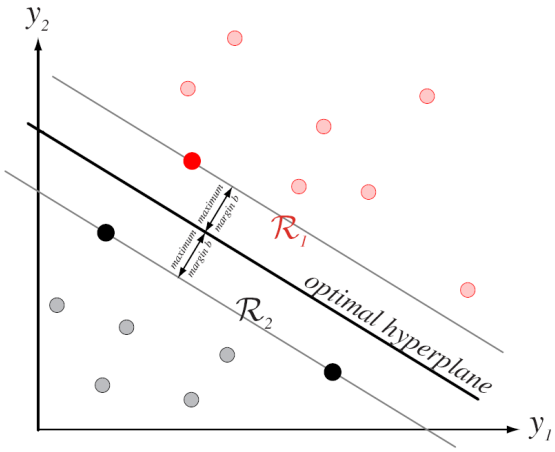


Ch 5.11 Duda

Support Vector Machines

SVMs aim to solve the linearly separable problem. First map feature space into high (possibly $\infty-$) dimensional space.

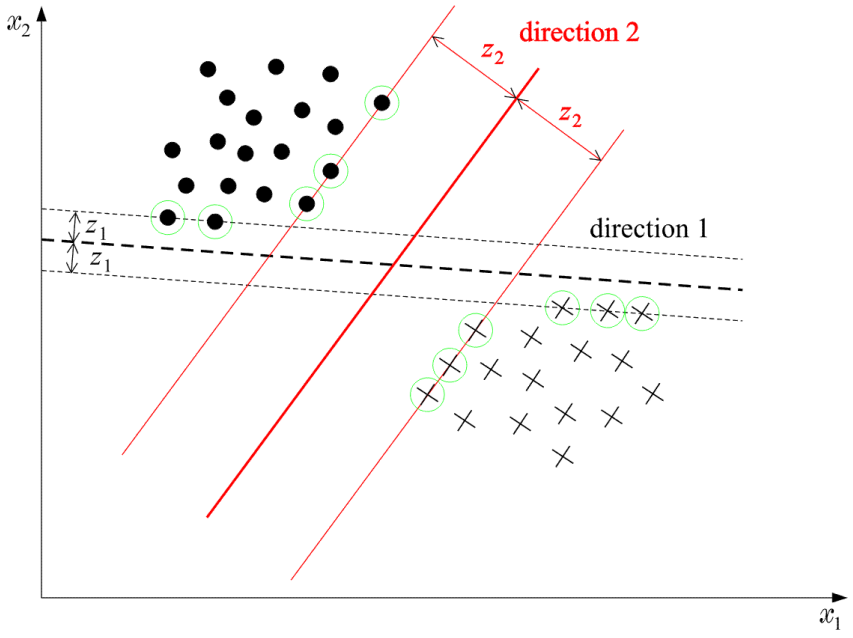
Then find separating hyperplane with **maximal margin**. Recall, intuitively we are more confident in classifying point far away from the decision boundary.



Learning takes the form of a constrained optimization scheme.

Why max margin

- For example these possible hyperplanes. Which one you will choose?
- "Direction 1" has a narrow margin thus on unseen (test) data it has higher likelihood of error.
- Clearly 'direction 2' is preferred



SVMs: Maximizing the Margin

Suppose that we have a margin γ, such that $\omega^{(i)} a^T y_i \geq \gamma$ for all points, $i = 1, 2, \ldots, n$. We therefore want to solve the following:

$$\max \gamma, \text{such that: } \omega^{(i)} a^T y_i \geq \gamma$$
$$\text{and: } \|a\| = 1$$

This is a messy optimization problem. However, equivalently we can solve:

$$\min \|a\|^2, \text{such that: } \omega^{(i)} a^T y_i \geq 1$$

That is, we search for the minimum size weight vector that is able to separate the data with a margin of $\gamma = 1$.

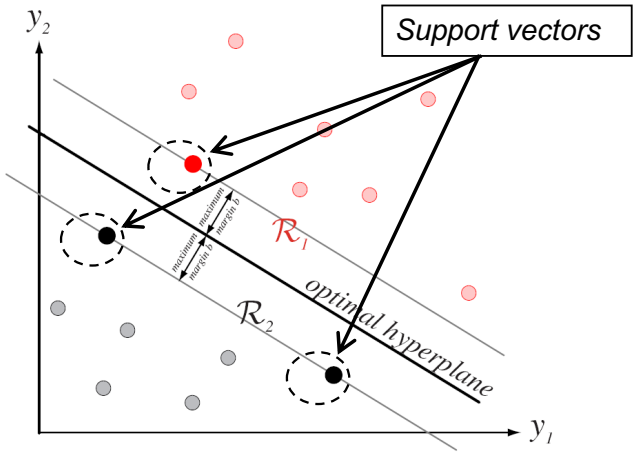
This form of the problem is a constrained quadratic optimization problem. It is convex and (relatively) easy to solve.

The Support Vectors

Interestingly the Max margin solution only depends on a subset of the training data – those that lie exactly on the margin (why?). These are called the **support vectors** (SVs).

Also

- the support vectors also define the equation of the optimal hyperplane: **a ~ weighted sum of $\omega^{(i)}y_i$**
- The hyperplane is unique the SVs are not unique. Why?



SVMs can be shown to generalize well in terms of cross validation error

SVM Generalization Error

- **Cross Validation (CV)** – Break the data into a *training set* and a *testing set*. Use the training data to learn the classifier then evaluate on the test data
- **Leave-one-out CV**: choose one data point at random as the testing set.

$$\{x_1, x_2, \ldots, \cancel{x_p}, x_{p+1}, \ldots x_n\}$$

- Note the LOOCV error will be unaffected unless x_p is a support vector. Therefore we have:

$$error_{LOOCV} \leq \frac{\#\text{ of SVs}}{n}$$

- Therefore the number of SVs tells us how confident we are in the SVM.

Ch 5.11 Duda

FEATURE ENGINEERING

Principal Component Analysis

MLP can learn “functions” of data but with high dimensional inputs they need some help!

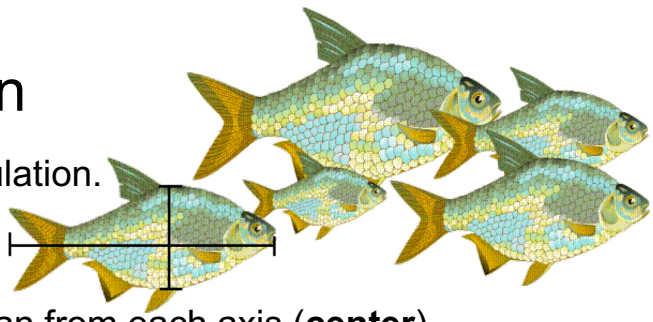
This topic concerns decomposing signals into useful low dimensional subsets:

- For feature space selection in classification
- For redundancy reduction
- To avoid overfitting
- For signal separation

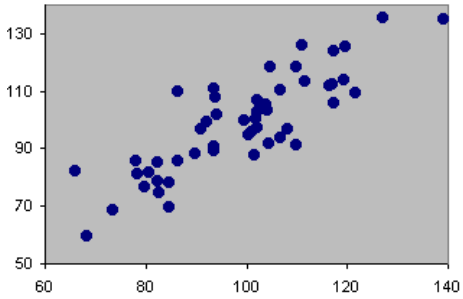
The key aim is to find a linear transform of the data that better represents the underlying information

e.g. Fourier transform of an image concentrates information into low frequencies

PCA – Graphical Intuition

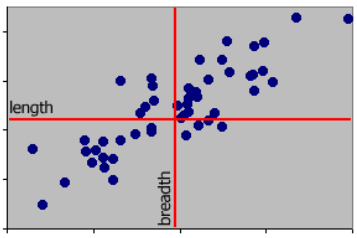


Suppose I want to characterize fish population.
Measure length/breadth and plot.



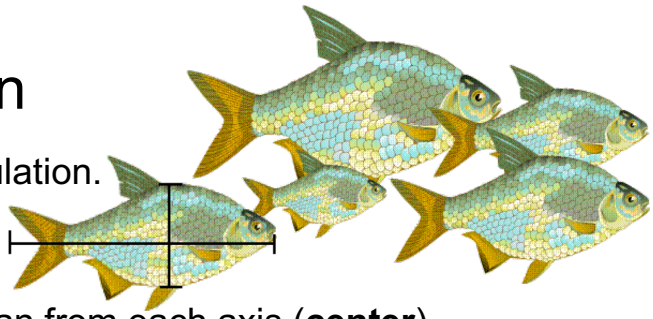
Subtract mean from each axis (**center**)

Note relationship between data matters only**.**

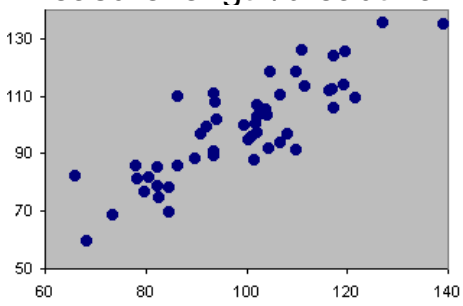


Inspired by http://www.cmbi.ru.nl/edu/bioinf4/prac-microarray/stats/PCA%20graphical%20explanation.htm

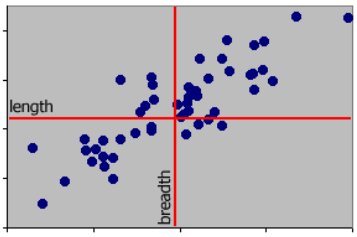
PCA – Graphical Intuition

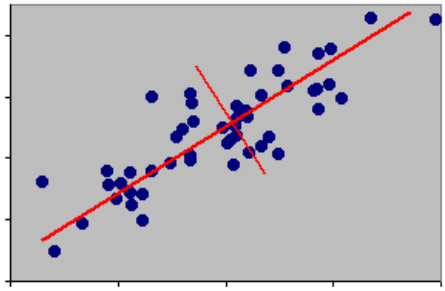


Suppose I want to characterize fish population.
Measure length/breadth and plot.



Subtract mean from each axis (**center**)
Note relationship between data matters only.



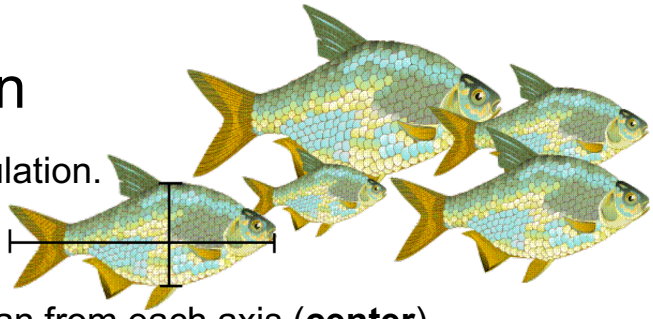


I can **move the axis!** But how to chose them? **I can rotate them.**

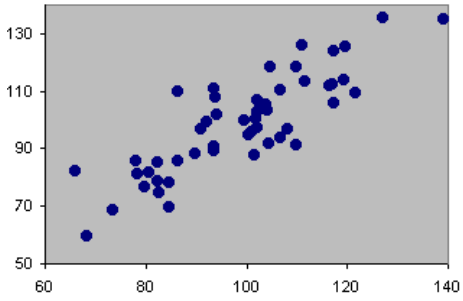
What is my objective: position one axis in such a way that it accounts for the largest proportion of the data's **variance**.

Inspired by http://www.cmbi.ru.nl/edu/bioinf4/prac-microarray/stats/PCA%20graphical%20explanation.htm

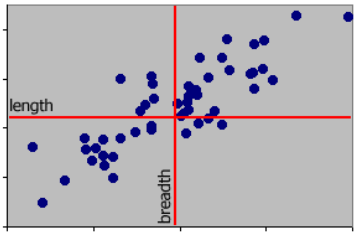
PCA – Graphical Intuition

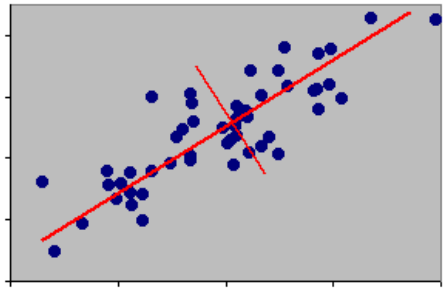


Suppose I want to characterize fish population.
Measure length/breadth and plot.

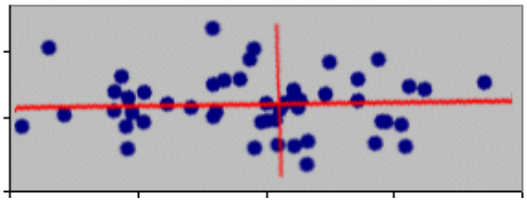


Subtract mean from each axis (**center**)
Note relationship between data matters only.





I can **move the axis!** But how to chose them? **I can rotate them.**



What is my objective: position one axis in such a way that it accounts for the largest proportion of the data's **variance**.

Inspired by http://www.cmbi.ru.nl/edu/bioinf4/prac-microarray/stats/PCA%20graphical%20explanation.htm

PCA – Graphical Intuition

What might you **call** this new axis? **size = length + breadth**

However, we could make one of the **variables** more important.

size = 0.75 x length + 0.25 x breadth

[these are weights and are important! They tell us the "significance" of each of the original variables.

What about the second axis of the ellipse?

Objective: account for as much of the remaining variation as possible but must also be uncorrelated (**orthogonal**) with the first. [trivial in 2D]

Apart from size, how else do the above fish differ?

Not much, minor differences in shape. If we ignore 2nd axis (after rotation) we would lose information about the different shapes, but since they are all very similar in shape we wouldn't lose much information at all.

Thus, in the above example we can reduce the data's dimensionality from two (length and breadth) to one (size), with little information loss.

PCA & Subspace Projections

Introduced by Pearson in 1901 "*On lines and planes of closest fit to a system of points in space.*" (a.k.a. Karhunen-Loéve transform (KLT), or the Hotelling transform,…).

Suppose our data consists of d-dimensional vectors $\mathbf{x}^{(n)} \in R^d$ and we want a low-dimensional approximation for the data.

Let $\boldsymbol{u}_i$ be an **orthonormal** basis for R^d (i.e. $U^T U = I$) then we can approximate x by:

$$\tilde{\mathbf{x}} = \sum_{i=1}^{M} (\mathbf{u}_i^T \mathbf{x}) \mathbf{u}_i + \sum_{j=M+1}^{d} b_j \mathbf{u}_j$$

Each b_j is a constant

let $z_i = \mathbf{u_i}^T \mathbf{x}$

Subspace Projections

x has d degrees of freedom while $\tilde{\mathbf{x}}$ has M deg. of freedom.

Principal Component Analysis – choose $\boldsymbol{u}_i$ and b_j to best approximate x in the LSE sense, ie., minimize E_M:

$$E_M = \frac{1}{2}\sum_{n=1}^{N}\left\|\mathbf{x}^{(n)} - \tilde{\mathbf{x}}^{(n)}\right\|^2 = \frac{1}{2}\sum_{n=1}^{N}\left\|\mathbf{U}^T\mathbf{x}^{(n)} - \mathbf{U}^T\tilde{\mathbf{x}}^{(n)}\right\|^2 = \frac{1}{2}\sum_{n=1}^{N}\sum_{j=M+1}^{d}\left(z_j^{(n)} - b_j\right)^2$$

Taking the derivative with respect to b_j *gives:*

$$\sum_{n=1}^{N}\left(z_j^{(n)} - b_j\right) = 0 \;\Rightarrow\; b_j = \frac{1}{N}\sum_{n=1}^{N} z_j^{(n)} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{u}_j^T\mathbf{x}^{(n)} = \mathbf{u}_j^T\bar{\mathbf{x}}$$

Subspace Projections (cont.)

So we can write:

$$E_M = \frac{1}{2}\sum_{n=1}^{N}\sum_{j=M+1}^{d}\left(\mathbf{u}_j^T\left(\mathbf{x}^{(n)} - \bar{\mathbf{x}}\right)\right)^2$$

$$= \frac{1}{2}\sum_{j=M+1}^{d}\sum_{n=1}^{N}\mathbf{u}_j^T\left(\mathbf{x}^{(n)} - \bar{\mathbf{x}}\right)\left(\mathbf{x}^{(n)} - \bar{\mathbf{x}}\right)^T\mathbf{u}_j = \frac{1}{2}\sum_{j=M+1}^{d}\mathbf{u}_j^T\mathbf{R}_{\mathbf{x}}\mathbf{u}_j$$

where $\mathbf{R}_{\mathbf{x}}$ is the sample covariance matrix for $\{\mathbf{x}^{(n)}\}$

Minimizing E_M with respect to $\boldsymbol{u}_j$ is satisfied by:

$$\mathbf{R}_{\mathbf{x}}\,\mathbf{u}_j = \lambda\mathbf{u}_j, \text{ for } \mathrm{j} = 1,...,\mathrm{M}$$

i.e. the M basis vectors are the principal eigenvectors of the sample covariance matrix.

PCA example: eigenfaces

In face recognition a common practice is to first project data (after alignment) onto a low dimensional PCA space,

e.g. images from images AT&T Laboratories Cambridge.

u_1 u_2 u_3 u_4

Eigenfaces capture appearance and lighting conditions quite well.

PCA Algorithm Summary and Projection on Principal Directions

The PCA algorithm is summarized as follows:

1. subtract mean $\bar{\mathbf{x}}$ from data
2. Calculate sample covariance matrix, $\mathbf{R}_\mathbf{x}$ for $\{\mathbf{x}_n - \bar{\mathbf{x}}\}$
3. Perform eigenvalue decomposition: $\mathbf{R}_x = \mathbf{U}\Lambda\mathbf{U}^T$
4. Approximate data by the first M components that have the largest eigenvalue:

$$\mathbf{x}_n \approx \bar{\mathbf{x}} + \sum_{i=1}^{m} (\mathbf{u}_i^T (\mathbf{x}_n - \bar{\mathbf{x}}))\mathbf{u}_i$$

Recall by design eigenvectors are **ORTHOGONAL** with each other $\mathbf{u_i}^T\mathbf{u_j}=0$ (i # j) and have length 1 ($\mathbf{u_i}^T\mathbf{u_i}=1$).

An interactive demo

- http://setosa.io/ev/principal-component-analysis/

Kernel PCA

(linear) PCA method is built upon the eigenanalysis of (n number of data points)

$$R_X = \frac{1}{n} X^T X = \frac{1}{n} \sum_{i=1}^{n} \underline{x}_i \underline{x}_i^T$$

There is an equivalent built upon the eigenanalysis of

$$K = XX^T = \begin{bmatrix} \underline{x}_1^T \underline{x}_1 & \cdots & \underline{x}_1^T \underline{x}_n \\ \vdots & \ddots & \vdots \\ \underline{x}_n^T \underline{x}_1 & \cdots & \underline{x}_n^T \underline{x}_n \end{bmatrix}$$

known as the Gram matrix.

The kernel PCA method is the kernelized version of this, where inner products are replaced by kernel operations.

➢ The kernel PCA algorithm

- Compute the Gram matrix.

$$K(i,j) = K(\underline{x}_i, \underline{x}_j) \ , \quad i,j = 1,2,\ldots,n$$

- Compute the m dominant eigenvalues / eigenvectors.

$$\lambda_k \ , \underline{a}_k \ , \ k = 1,2,\ldots,m$$

- Perform normalization to unity.

$$1 = n\lambda_k \underline{a}_k^T \underline{a}_k \ , \ k = 1,2,\ldots,m$$

- Given a vector $\underline{x}$, perform the following "nonlinear mapping".

$$y(k) = \sum_{i=1}^{n} a_k(i) K(\underline{x}_i, \underline{x}) \ , \ k = 1,2,\ldots,m$$

Self-learning

➢ Remark

- The kernel PCA is equivalent with performing a (linear) PCA in a Reproducing kernel Hilbert space (RKHS) H, after a mapping

$$\underline{x} \rightarrow \phi(\underline{x}) \in H$$

- It can be shown that the dominant eigenvectors of

$$\frac{1}{n} \sum_{i=1}^{n} \phi(\underline{x}_i) \phi^T(\underline{x}_i)$$

are given in terms of the dominant eigenvectors of the Gram matrix, i.e.,

$$\underline{v}_k = \sum_{i=1}^{n} a_k(i) \phi(\underline{x}_i) \ , \ k = 1,2,\ldots,m$$

Hence the projection of $\phi(\underline{x})$ on $\underline{v}_k$ is given by:

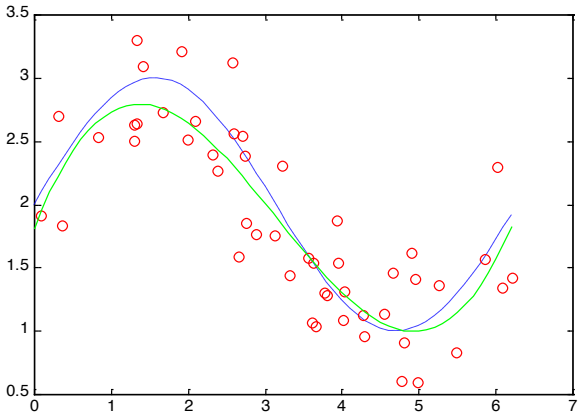
$$\langle \underline{v}_k, \phi(\underline{x}) \rangle = \sum_{i=1}^{n} a_k(i) \langle \phi(\underline{x}_i), \phi(\underline{x}) \rangle = \sum_{i=1}^{n} a_k(i) K(\underline{x}_i, \underline{x}),$$

using the properties of the RKHS.

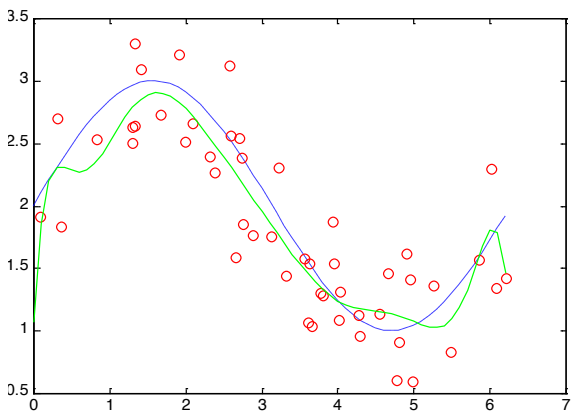
WHY CARE ABOUT FEATURES AND HOW MANY

Overfitting in ML

Given a limited data set maximising the likelihood L(**w**) may lead to *overfitting.* If the model order (dimensions of **w**) is large enough we can 'exactly' fit model to data.



3rd order polynomial fit



10th order polynomial fit

Ch 9.3.1 Duda

Self-learning

Overfitting in ML

Overfitting relates to *Model Order Selection.* As with estimating power spectra the problem comes from a *bias - variance* trade-off. Consider the expected error between an optimal estimator $F(x)$ (ie. the true function) and an empirical estimator $g(x)$. [We use the augmentation trick.]

$$\mathbb{E}_x\left\{(F(x)-g(x))^2\right\} = \mathbb{E}_x\left\{\left(\ F(x) - \mathbb{E}_x\{g(x)\} + \mathbb{E}_x\{g(x)\} - g(x)\ \right)^2\right\}$$

$$= \underbrace{\mathbb{E}_x\left\{\left(F(x)-\mathbb{E}_x\{g(x)\}\right)^2\right\}}_{\text{bias}^2} + \underbrace{\mathbb{E}_x\left\{\left(\mathbb{E}_x\{g(x)\}-g(x)\right)^2\right\}}_{\text{variance}}$$

$$+ \underbrace{2\ \mathbb{E}_x\left\{\left(F(x)-\mathbb{E}_x\{g(x)\}\right)\left(\mathbb{E}_x\{g(x)\}-g(x)\right)\right\}}_{=0}$$

The expectation over all possible different training sets (also known as the ensemble average). It is **not** the E we used to denote as the cost function thus far.

To show why this is 0 do the multiplication and then recall properties of the Expectation of constants!

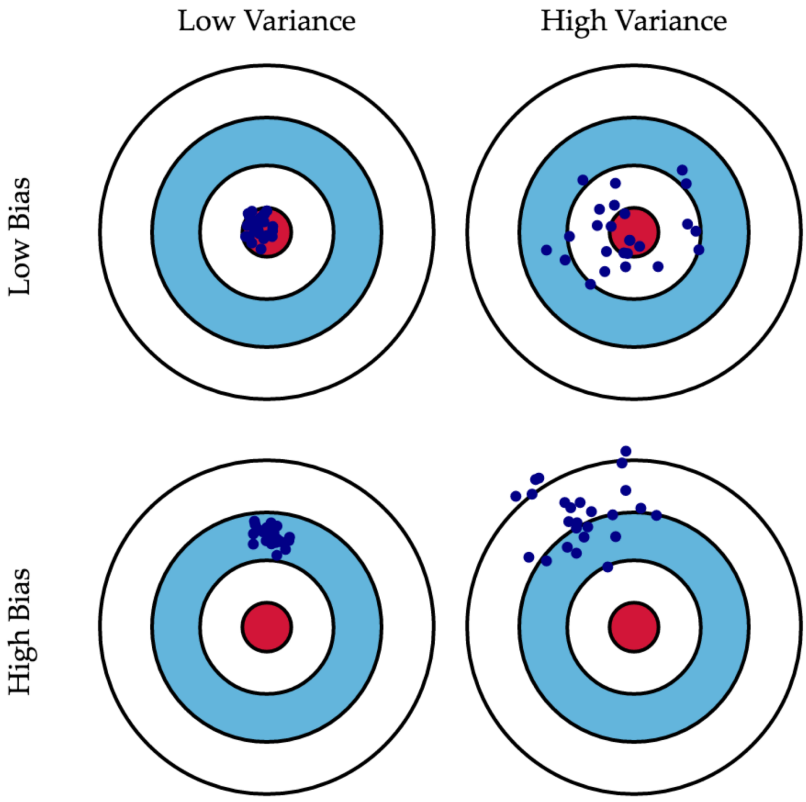
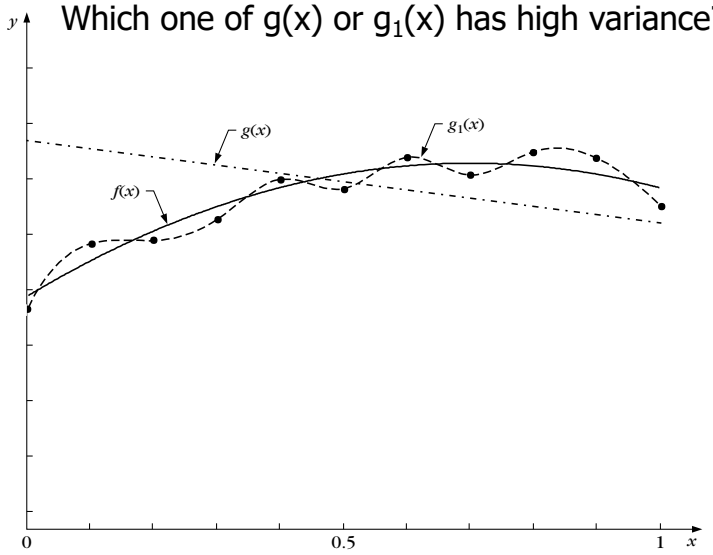


Fig. 1 Graphical illustration of bias and variance.

Source: http://scott.fortmann-roe.com/docs/BiasVariance.html

Example

Which one of g(x) or $g_1(x)$ has high variance?

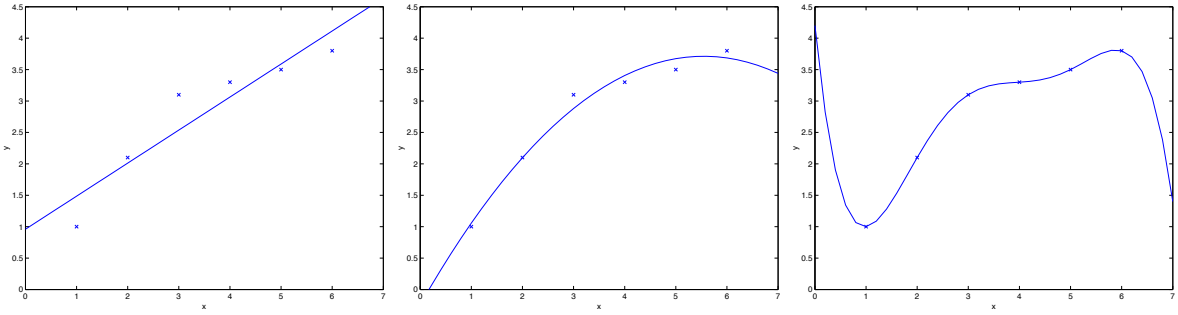


Complex model results in low-bias but a high variance, as one changes from one training set to another.

Simple model results in high bias but low variance.

Example in regression

- Identify high bias / variance cases:



Ignore the data ➔

* Big approximation errors (high bias)
* No variation between data sets (no variance)

Use all data ➔

* No approximation errors (zero bias)
* Variation between data sets (high variance)

Overfitting in ML

These examples tell us:
- Too **complex** a model → **high variance**.
- Too **simple** a model → **high bias** (the simplest model is the fixed value).

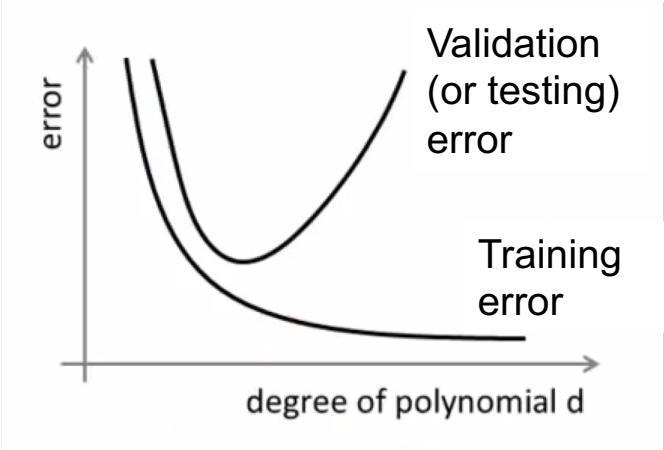
Possible solution on diagnosis for models and data (size)
Cross-validation: Break the data into **3 pieces**: *training*, *validation* and *testing* sets. Set the testing set apart, do not touch it, till the end.

Use the training data to learn the model parameters.
But identify the best model order (and other parameters such as step size, regularizers, type of activation function etc) using the validation ("out-of-sample") set.

If you are happy with the performance on the validation set, then you can now take the testing set and run the algorithm on the testing set.
Then report results on the training and testing sets.

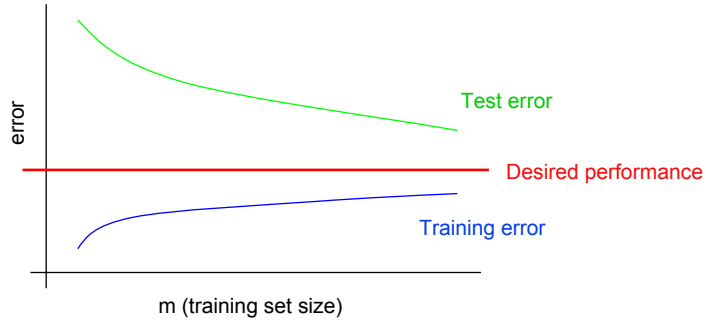
Diagnosing bias/variance: the practical picture

- Finding a good model size



Bias vs variance: on data size

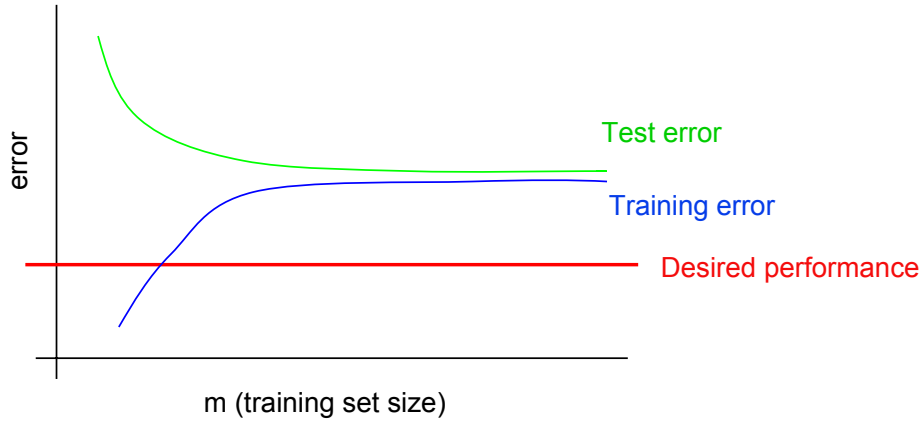
- Typical curve with high variance



- Test error decreases as m increases ➔ larger training set may help
- There is a gap between training & test error

Bias vs. variance: on data size

- Typical curve with high bias

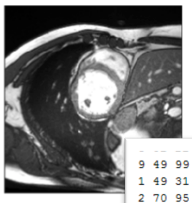


- We are in trouble: training error is high too
- Small gap between the two errors

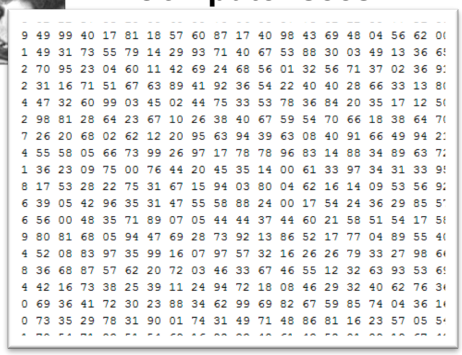
DEEP LEARNING

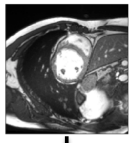
From “hand-designed” feature spaces to data-driven ones…

We see



Computer sees





Extract features

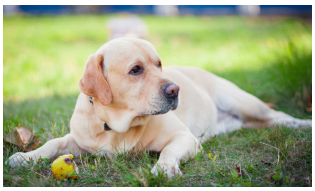
Human engineering
(e.g. input x and x^2)

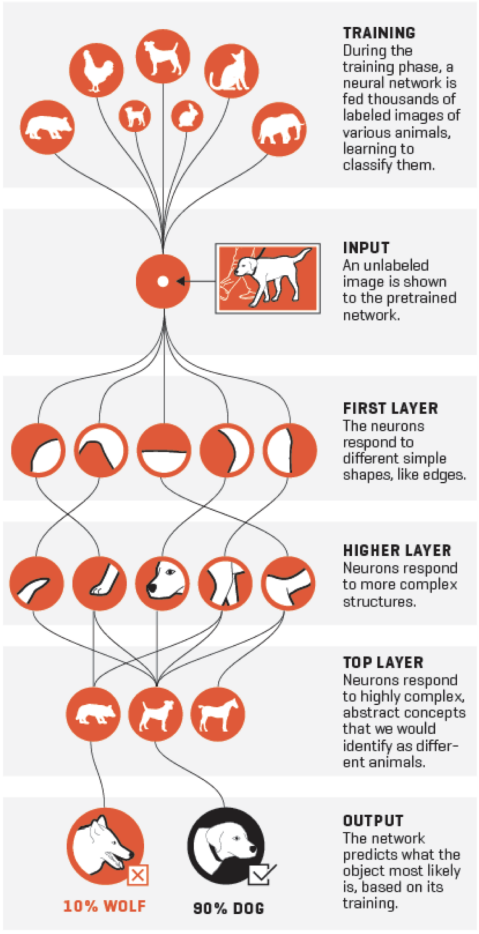
Decision function

decision

e.g. SVM, GLD, MLP

What if we could “extract” automatically good features….

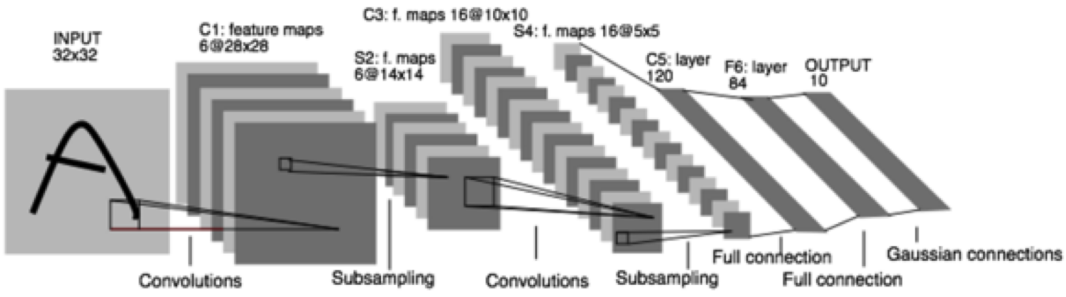




http://fortune.com/ai-artificial-intelligence-deep-machine-learning/

Convolutional Networks (CNNs)

Consider the problem of building a classifier that is insensitive to translation (or scale, rotation, etc.). A Convolutional Network encodes the invariance within the MLP structure [LeCun 1998].



LeNet: a layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten digits. [LeNet]

Ch 6.10.4 Duda and Inspired by Goodfellow et al Deep learning book

Convolutional Networks (CNNs)

Consider the problem of building a classifier that is insensitive to translation (or scale, rotation, etc.). A Convolutional Network encodes the invariance within the MLP structure [LeCun 1998].

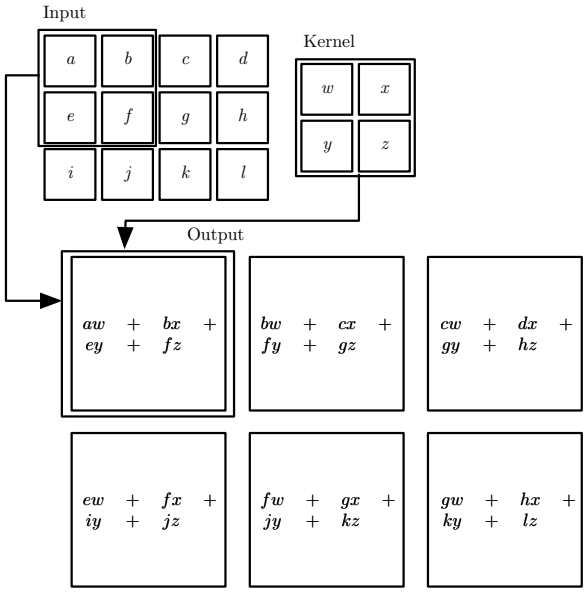
Let us first review 2D convolution ➔

Can I write *w conv I* as a matrix multiplication?

Next layer O = W*I

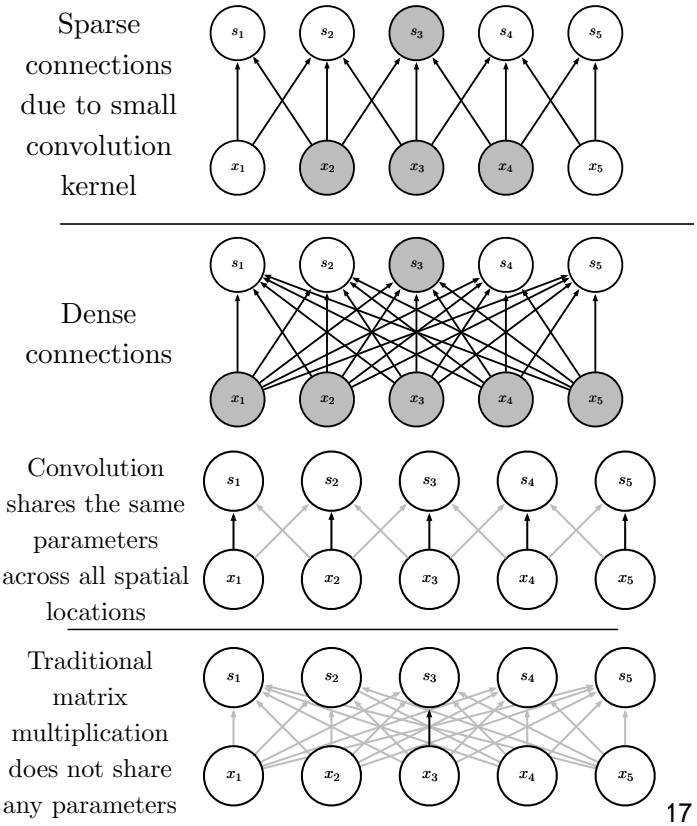
W must have a correct shape and

I: a vector (input layer)!



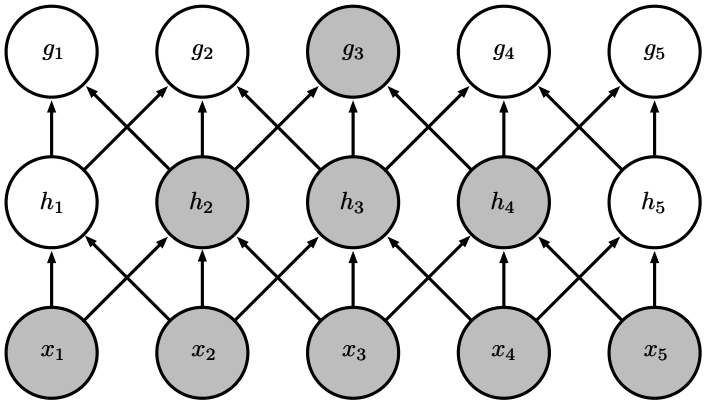
Convolutional Networks (CNNs)

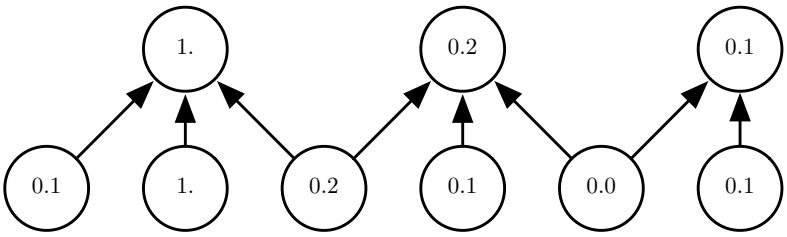
- Connections are **restricted**: hidden units are connected identically to neighbours to encode shift/delays
- Training using back prop. but with ties weights across shifted units (***weight sharing***)
- The resulting MLP has much fewer weights to train than a traditional MLP

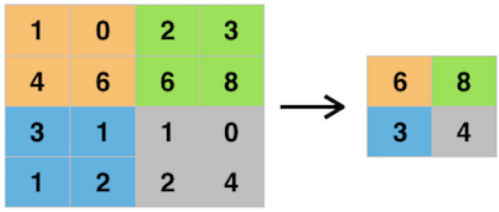


Convolutional Networks (CNNs)

- Growing "*influence*" with depth ➔
  (the top layer is influenced by a large span of inputs despite only connected to 3 previous units)
- *Max pooling* helps with invariance and *down sampling* reduces numbe of parameters

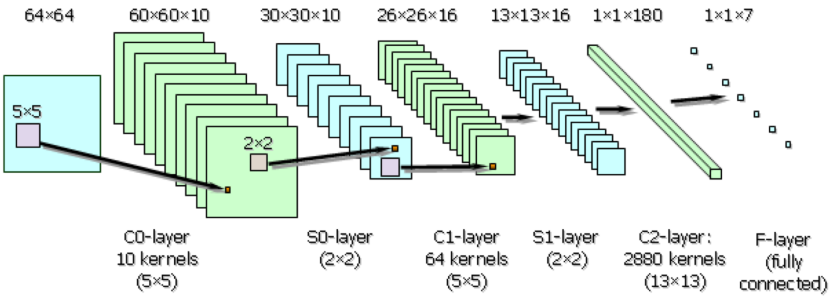






CNNs & Deep Learning

- CNNs are often considered the ancestors of Deep Learning.
- The idea is to use MLP with many (≥ 3) hidden layers. This involves lots of 'tricks' to make training work: convolution, subsampling, pooling of outputs,…, pretraining (helps to start the weights with good initial values)



Application to face detection and pose estimation

but exhibit state-of-the-art performance…

e.g. see - http://www.cs.nyu.edu/~yann/research

More Demos

- http://cs.stanford.edu/people/karpathy/convnetjs/