

# Predictive Learning from Data

## LECTURE SET 5

### Nonlinear Optimization Strategies

Cherkassky, Vladimir, and Filip M. Mulier. *Learning from data: concepts, theory, and methods*. John Wiley & Sons, 2007.

Source: Dr. Vladimir Cherkassky (revised by Dr. Hsiang-Han Chen)

---

PLEASE DO NOT DISTRIBUTE WITHOUT AUTHOR'S PERMISSION.

# OUTLINE

- Nonlinear optimization in predictive learning
- Overview of ANN
- Stochastic approximation (gradient descent)
- Iterative methods
- Greedy optimization
- Summary and discussion

# Optimization

- **Optimization** is concerned with the problem of determining **extreme values** (**maxima** or **minima**) of a **function** on a given domain.
- Let  $f(\mathbf{x})$  be a real-valued function of real variables  $x_1, x_2, \dots, x_m$ ,
- If  $\mathbf{x}^*$  minimizes unconstrained function  $f(\mathbf{x})$ , then the **gradient of  $f(\mathbf{x})$**  evaluated at  $\mathbf{x}^*$  is **zero**

$$\nabla f(\mathbf{x}) = 0$$

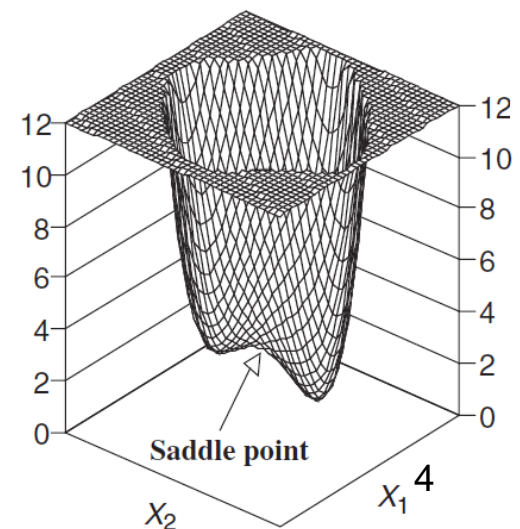
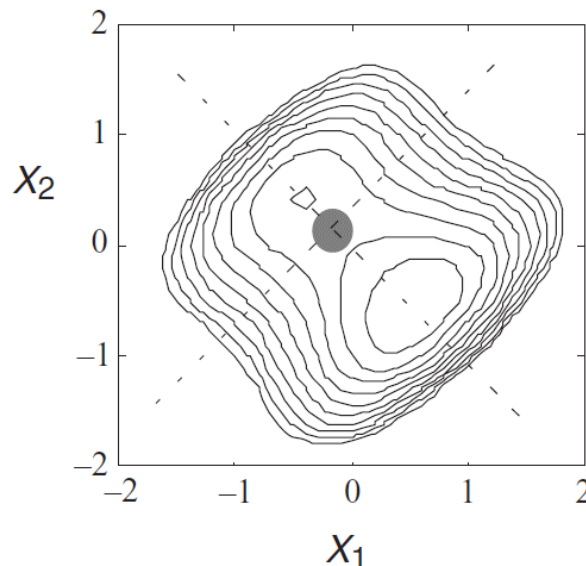
- That is,  $\mathbf{x}^*$  is a solution of the system of equations

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = 0 \quad (i = 1, 2, \dots, m)$$

# Optimization

- The point where  $\nabla f(\mathbf{x}) = \mathbf{0}$  is called a **stationary** or **critical point**;
- It can be a (local) **minimum**, a **maximum**, or a **saddle point** of  $f(\mathbf{x})$

The saddle point is a **local minimum** along the line  $x_2 = -x_1$  and a **local maximum** in the orthogonal direction.



# Optimization

A critical point  $\mathbf{x}^*$  can be further checked for optimality by considering the **Hessian matrix** of **second partial derivatives**:

$$\{\mathbf{H}_f(\mathbf{x})\}_{ij} = \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j}$$

evaluated at  $\mathbf{x}^*$ . At a critical point  $\mathbf{x}^*$  where the gradient is zero:

- If  $\mathbf{H}_f(\mathbf{x})$  is positive definite, then  $\mathbf{x}^*$  is a minimum of  $f$
- If  $\mathbf{H}_f(\mathbf{x})$  is negative definite, then  $\mathbf{x}^*$  is a maximum of  $f$
- If  $\mathbf{H}_f(\mathbf{x})$  is indefinite, then  $\mathbf{x}^*$  is a saddle point of  $f$

# Optimization

## Second Derivative Test for Saddle Points

Let  $f(x, y)$  is a function in two variables for which the first and second-order partial derivatives are continuous on some disk that contains the point  $(a, b)$ .

• If  $f_x(a, b) = 0$  and  $f_y(a, b) = 0$ ,

we define  $D = f_{xx}(a, b) f_{yy}(a, b) - [f_{xy}(a, b)]^2$  then:

- If  $D > 0$  and  $f_{xx}(a, b) > 0$ ,  $f$  has a **local minimum** at  $(a, b)$
- If  $D > 0$  and  $f_{xx}(a, b) < 0$ , has a **local maximum** at  $(a, b)$
- If  $D < 0$ ,  $f$  has a **saddle point**  $(a, b)$
- If  $D = 0$ , the test fails.

# Optimization

- With nonlinear optimization, there is always a possibility of **several local minima** and **saddle points**.
- This has two important implications:
  1. An optimization algorithm can find, at best, only a local minimum.
  2. The local minimum found by an algorithm is likely to be close to an initial point  $x_0$ .

The chances for obtaining globally optimal solution can be **improved** (but not assured) by **brute-force** computational techniques.

EX: **Restarting** optimization with **many (randomized) initial points** and/or using **simulated annealing** to escape from local minima.

# Optimization in predictive learning

- Optimization is used in learning methods for parameter estimation.
- Recall implementation of SRM:
  - fix complexity (VC-dimension)
  - minimize empirical risk
- Two related issues:
  - parameterization (of possible models)
  - optimization method
- Many learning methods use dictionary parameterization

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=0}^m w_i g(\mathbf{x}, \mathbf{v}_i)$$

- Optimization methods vary



# Nonlinear Optimization

- The ERM approach

$$R_{emp}(\mathbf{V}, \mathbf{W}) = \frac{1}{n} \sum_{i=1}^n L(y_i, f(\mathbf{x}_i, \mathbf{V}, \mathbf{W}))$$

where the model

$$f(\mathbf{x}, \mathbf{V}, \mathbf{W}) = \sum_{j=1}^m w_j g_j(\mathbf{x}, \mathbf{v}_j)$$

*Two factors contributing to nonlinear optimization*

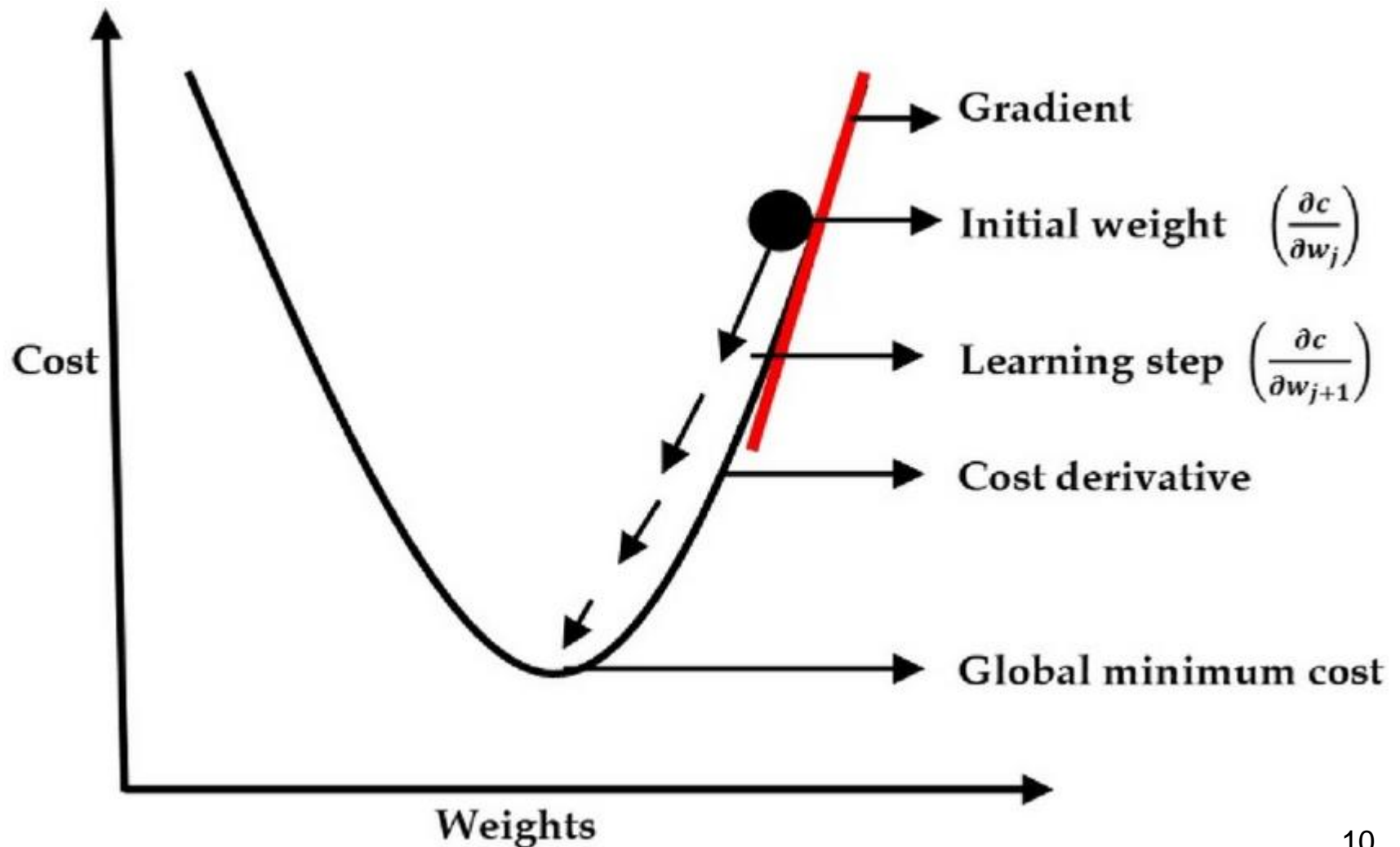
- nonlinear basis functions  $g_j(\mathbf{x}, \mathbf{v}_j)$
- non-convex loss function  $L$

*Examples*

convex loss: squared error, least-modulus

non-convex loss: 0/1 loss for classification

# Nonlinear Optimization - convex



# Unconstrained Minimization

- ERM or SRM (with squared loss) lead to unconstrained convex minimization
- Minimization of a real-valued function of many input variables  
→ Optimization theory
- We discuss only *popular optimization strategies* developed in statistics, machine learning and neural networks
- These optimization methods have been introduced in various fields and usually use specialized terminology

# OUTLINE

- Nonlinear optimization in predictive learning
- Overview of ANN
- Stochastic approximation (gradient descent)
- Iterative methods
- Greedy optimization
- Summary and discussion

# Overview of ANN's

- Huge interest in understanding the nature and mechanism of **biological/human learning**
- **Biologists + psychologists** do not adopt classical parametric statistical learning, because:
  - parametric modeling is not biologically plausible
  - biological info processing is clearly different from algorithmic models of computation

# Overview of ANN's (Conti.)

- **Mid 1980's:** growing interest in applying biologically inspired computational models to:
  - - developing computer models (of human brain)
  - - various engineering applications
- → New field **Artificial Neural Networks** (~1986 – 1987)
- ANN's represent **nonlinear estimators** implementing the ERM approach (usually squared-loss function)

# History of ANN

1943	McCulloch-Pitts neuron
1949	Hebbian learning
1960's	Rosenblatt (perceptron), Widrow
60's-70's	dominance of 'hard' AI
1980's	resurgence of interest (PDP group, MLP etc.)
1990's	connection to statistics/VC-theory
2000's	mature field/ lots of fragmentation
2010's	renewed interest ~ <i>Deep Learning</i>

# Deep Learning

- New marketing term or smth different?
  - several successful applications
  - interest from the media, industry etc.
  - very limited theoretical understanding

For critical & amusing discussion see:

Article in IEEE Spectrum on Big Data

<http://spectrum.ieee.org/robotics/artificial-intelligence/machinelearning-maestro-michael-jordan-on-the-delusions-of-big-data-and-other-huge-engineering-efforts>

And follow-up communications:

<https://www.facebook.com/yann.lecun/posts/10152348155137143>

<https://amplab.cs.berkeley.edu/2014/10/22/big-data-hype-the-media-and-other-provocative-words-to-put-in-a-title/>



# Neural vs Algorithmic computation

- Biological systems do not use principles of digital circuits

	Digital	Biological
Connectivity	1~10	~10,000
Signal	digital	analog
Timing	synchronous	asynchronous
Signal propag.	feedforward	feedback
Redundancy	no	yes
Parallel proc.	no	yes
Learning	no	yes
Noise tolerance	no	yes

# Neural vs Algorithmic computation

- Computers excel at **algorithmic tasks** (well-posed mathematical problems)
- Biological systems are superior to digital systems for **ill-posed problems** with noisy data
- Example: **object recognition** [Hopfield, 1987]

**PIGEON:**  $\sim 10^9$  neurons, cycle time  $\sim 0.1$  sec,  
each neuron sends 2 bits to  $\sim 1\text{K}$  other neurons  
 $\rightarrow 2 \times 10^{13}$  bit operations per sec

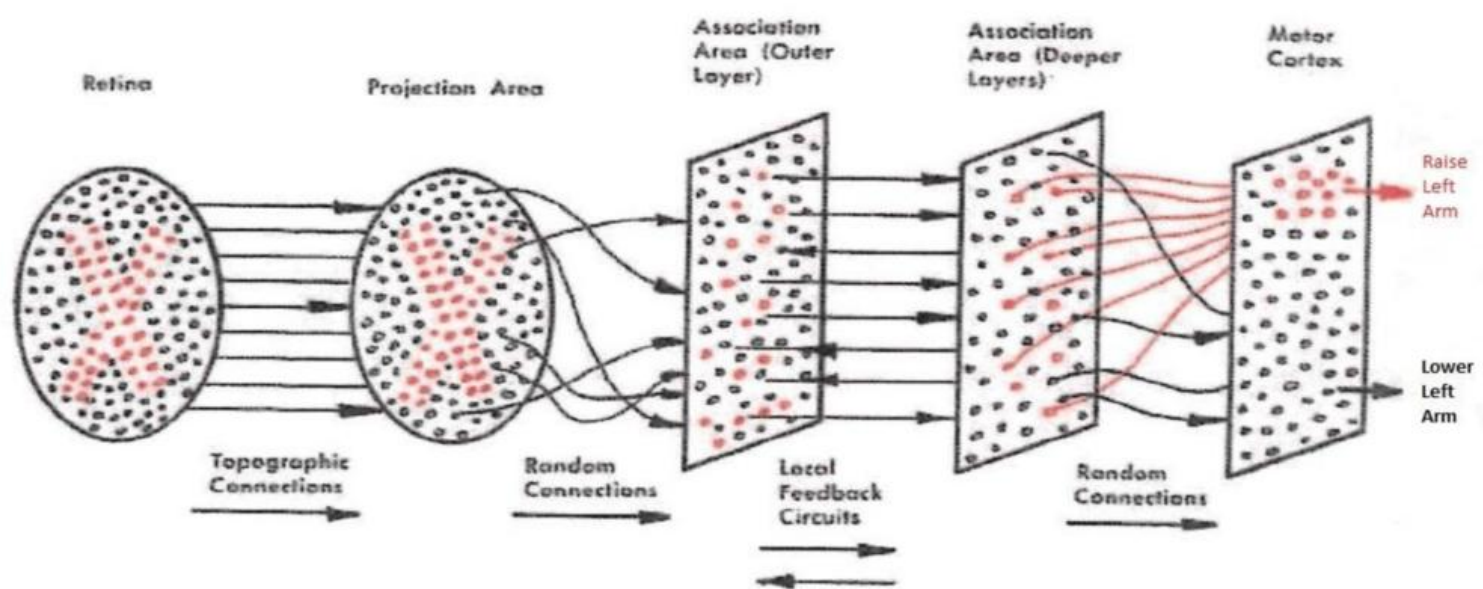
**OLD PC:**  $\sim 10^7$  gates, cycle time  $10^{-7}$ , connectivity=2  
 $\rightarrow 10 \times 10^{14}$  bit operations per sec

Both have similar raw processing capability, but pigeons are better at recognition tasks

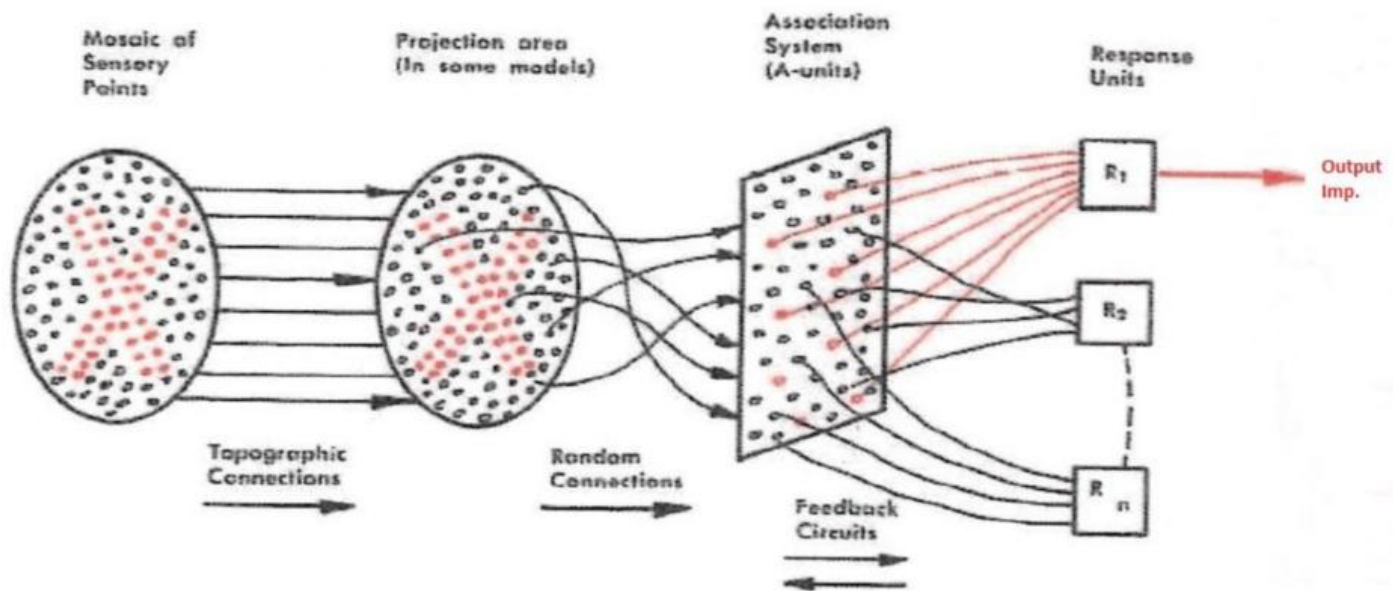
# Goals of ANN's

- Develop models of computation inspired by biological systems
- Study computational capabilities of networks of interconnected neurons
- Apply these models to real-life applications

**Learning in NNs** = modification (adaptation) of synaptic connections (weights) in response to external inputs



**FIG. 1 — Organization of a biological brain.** (Red areas indicate active cells, responding to the letter X.)



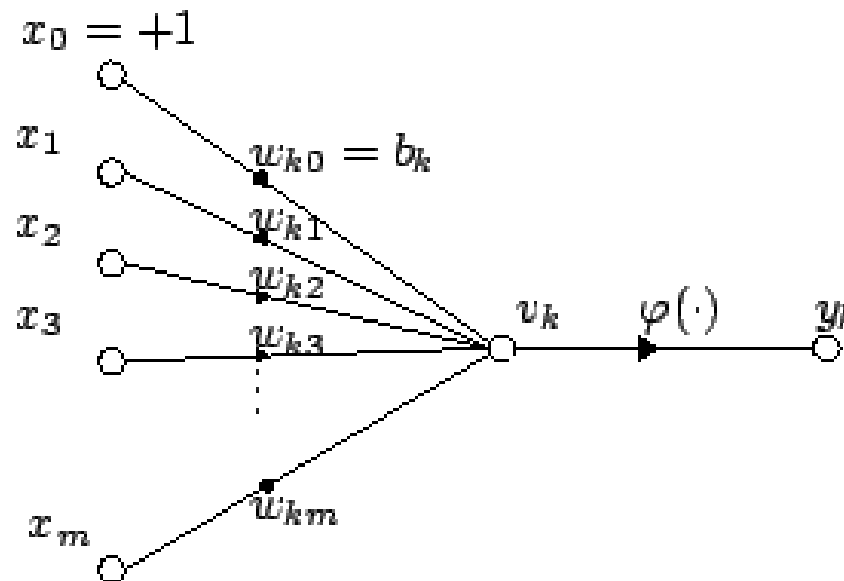
**FIG. 2 — Organization of a perceptron.**

# Neural terminology and artificial neurons

Some general descriptions of ANN's:

[http://en.wikipedia.org/wiki/Neural\\_network](http://en.wikipedia.org/wiki/Neural_network)

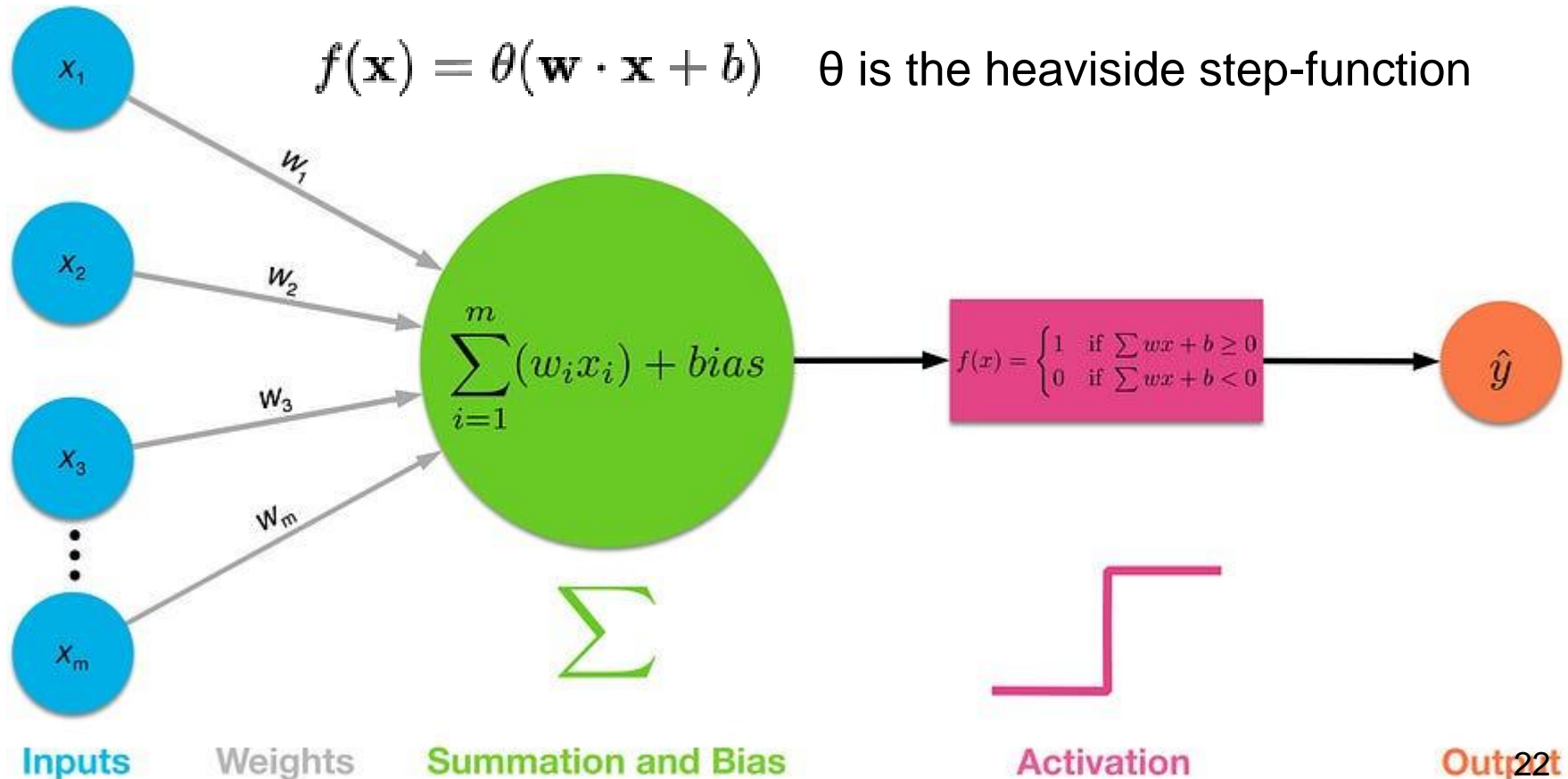
- McCulloch-Pitts neuron (1943)



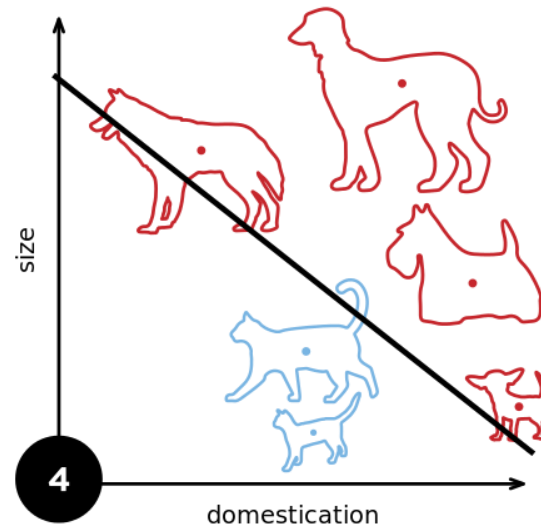
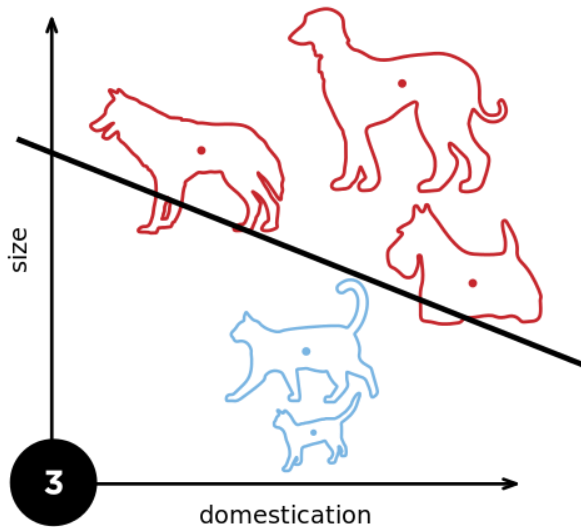
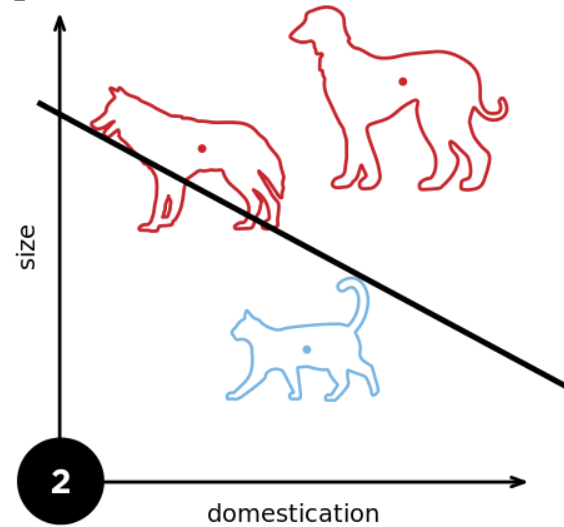
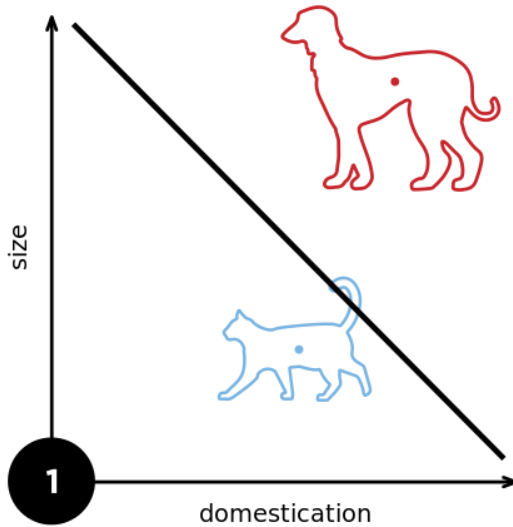
- Threshold (indicator) function of weighted sum of inputs

# Perceptron

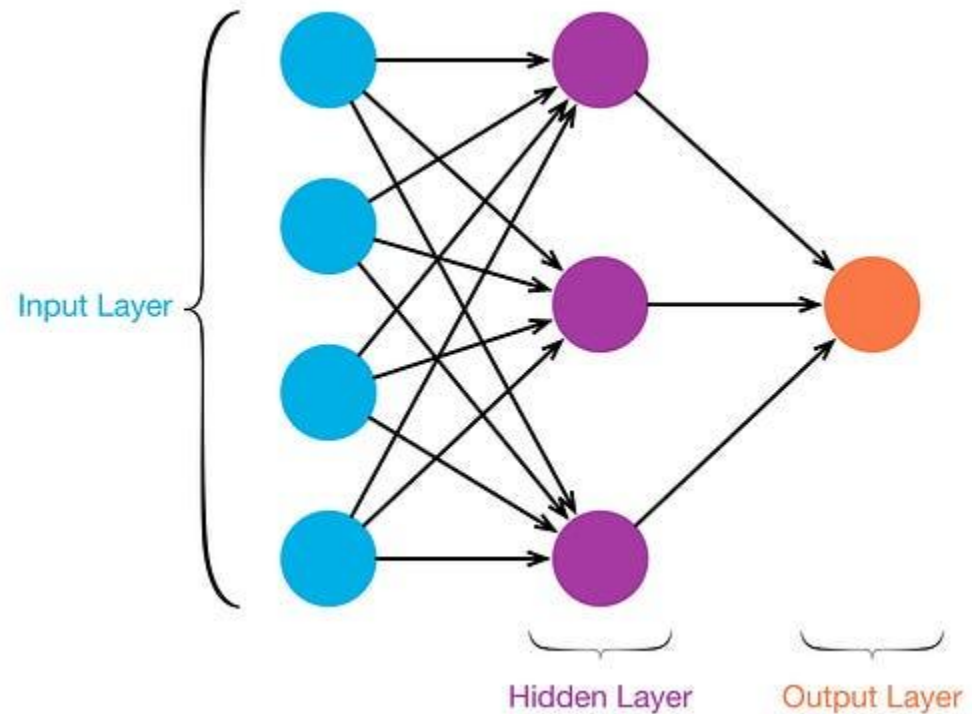
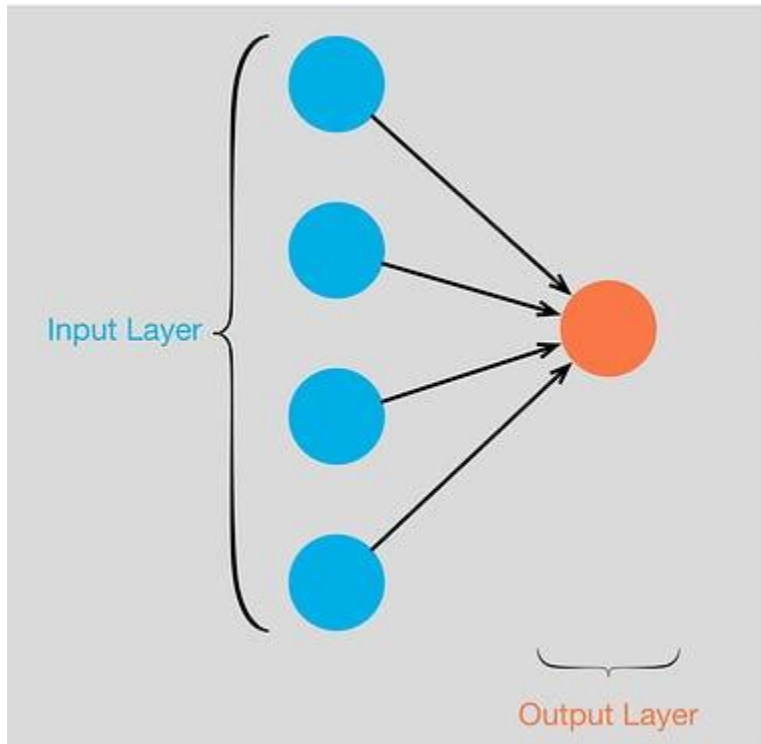
The perceptron is an algorithm for learning a **binary classifier** called a threshold function: a function that maps its input  $\mathbf{x}$  (a real-valued vector) to an output value



# Perceptron



# Single layer to multi-layer



Step function has no useful derivative (its derivative is 0 everywhere or undefined at the 0 point on x-axis). It doesn't work for backpropagation.

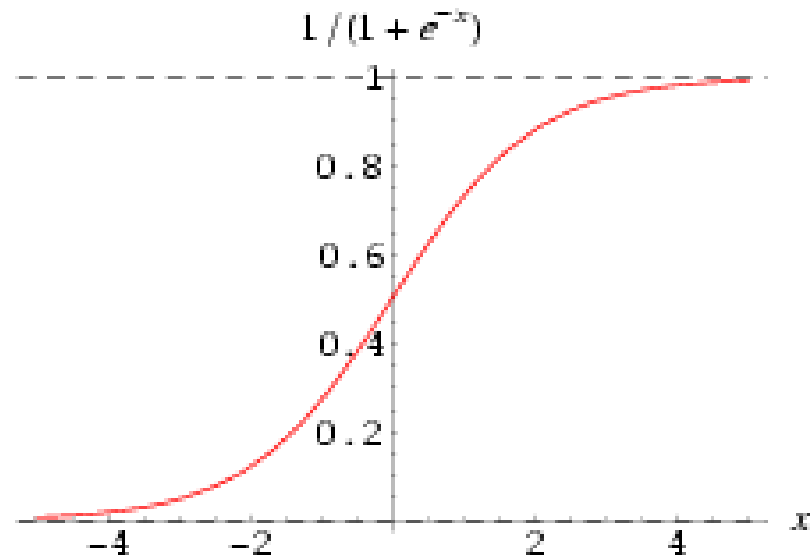
**Hence, we need a better activation function!**



# Multilayer Perceptron (MLP) network

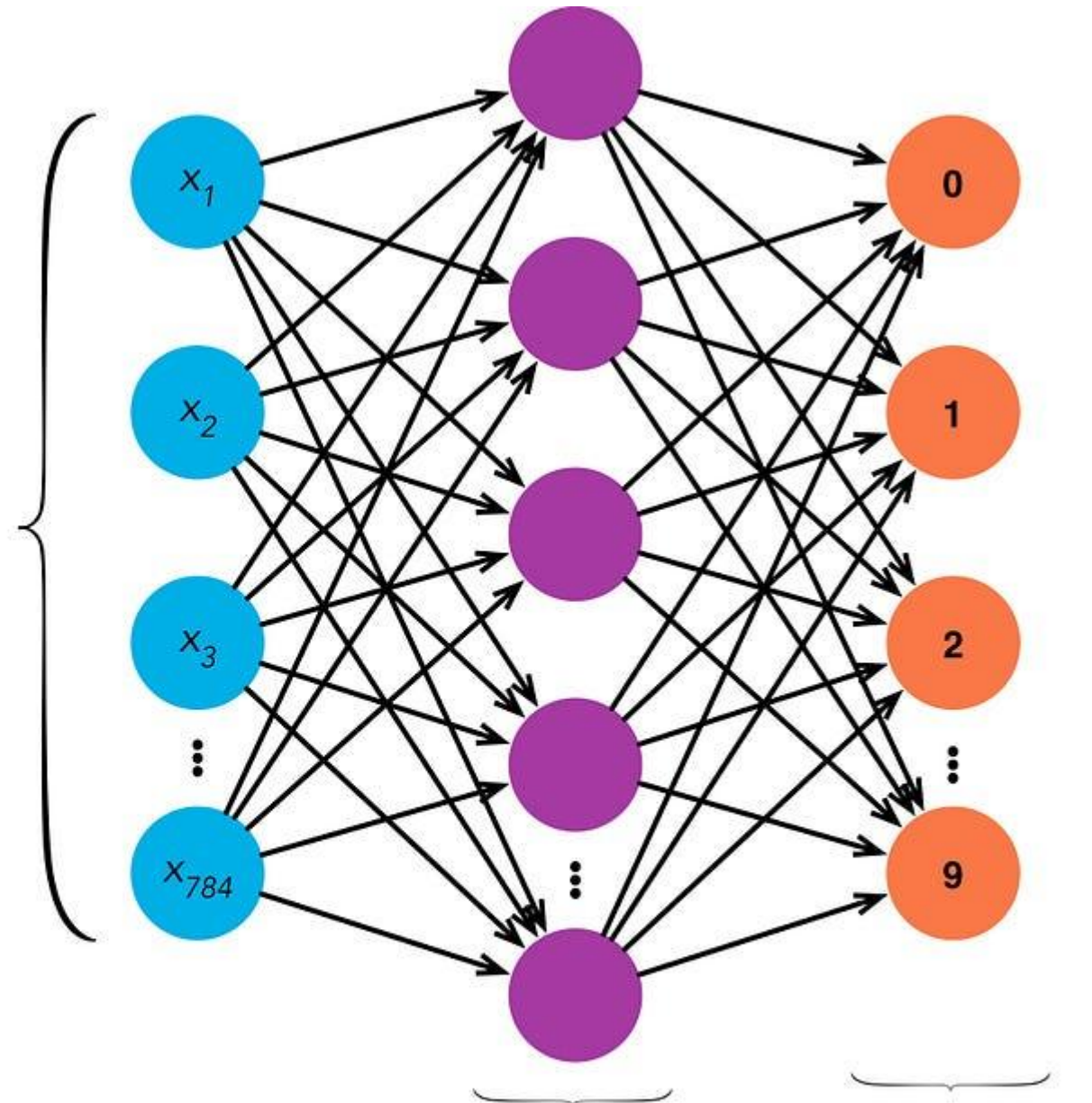
- **Basis functions of the form**  $g_i(t) = g((\mathbf{x}\mathbf{v}_i + b_i))$   
i.e. **sigmoid** aka logistic function

$$s(t) = \frac{1}{1 + \exp(-t)}$$



- commonly used in artificial neural networks
- combination of sigmoids ~ **universal approximator**

input layer: **784**  
(28x28) neurons, each  
with values between 0  
and 255



Hidden Layer:  
**16** hidden neurons

Output Layer:  
**10** classifiers  
for 10 digits

# Radial basis function

In mathematics a radial basis function (RBF) is a **real-valued function** whose value depends only on the **distance** between the input and some fixed point, either the origin or some other fixed point  $c$  called a center,

$$g_i(t) = g\left(\|\mathbf{x} - \mathbf{v}_i\|\right)$$

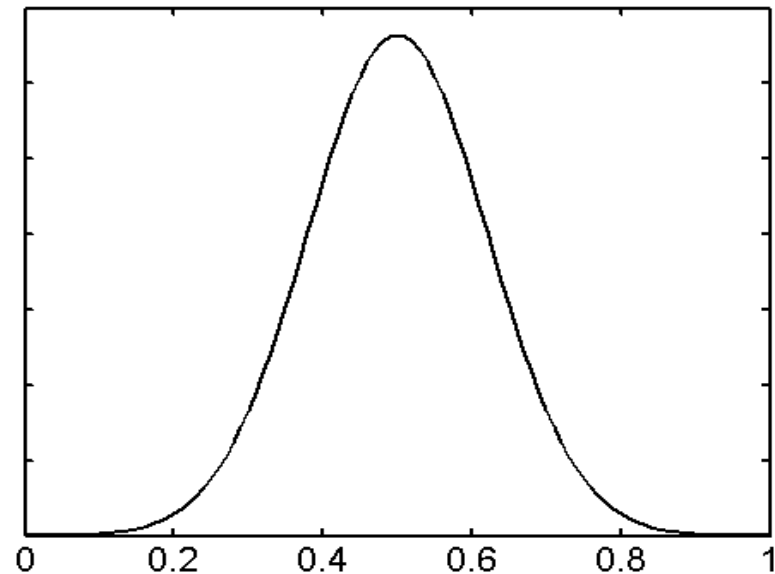
# RBF Networks

- Basis functions of the form  $g_i(t) = g(\|\mathbf{x} - \mathbf{v}_i\|)$   
i.e. Radial Basis Function(RBF)

$$g(t) = \exp\left(-\frac{t^2}{2\alpha^2}\right)$$

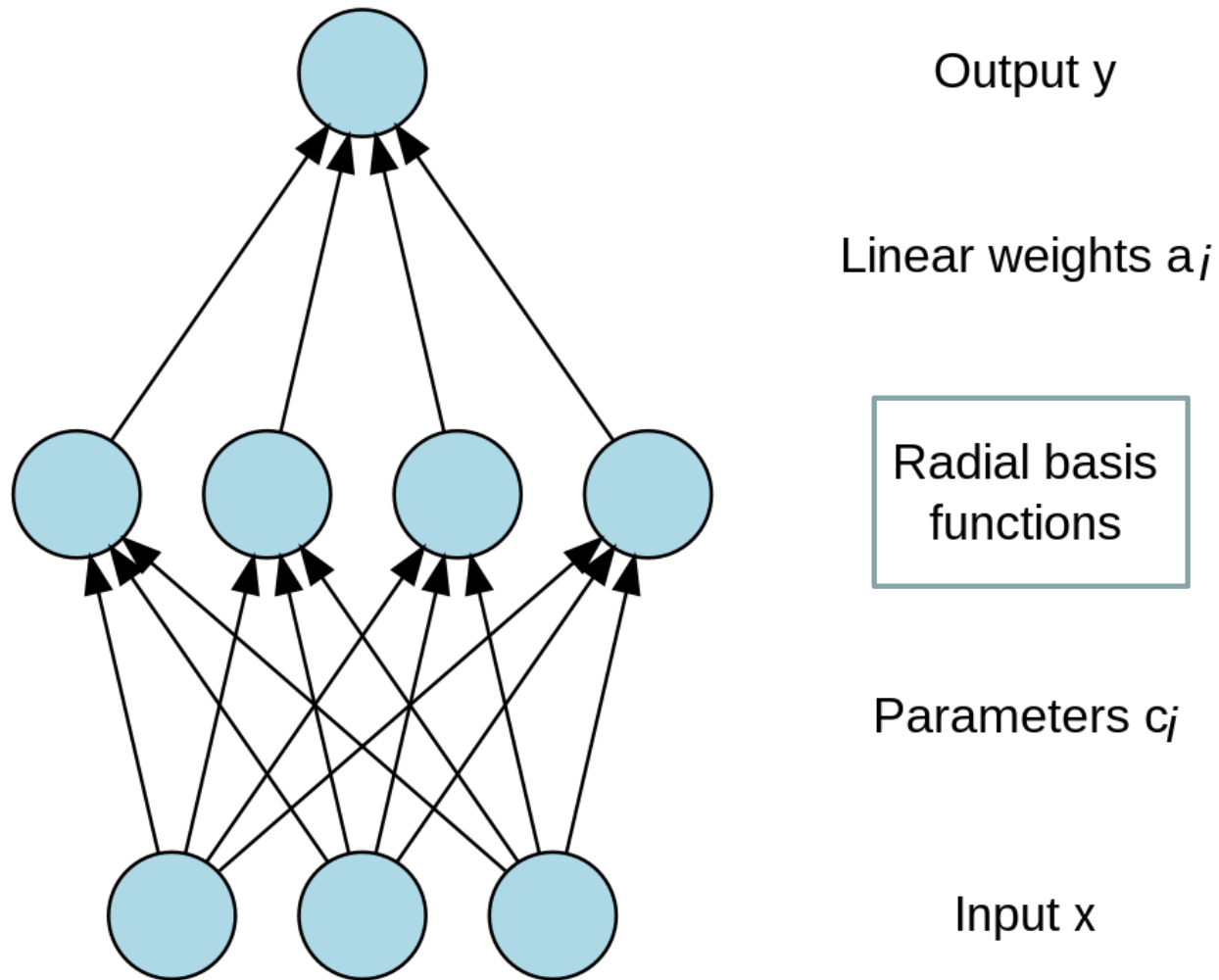
$$g(t) = (t^2 + b^2)^{-2}$$

$$g(t) = t$$



- RBF adaptive parameters: center, width
- commonly used in artificial neural networks
- combination of RBF's ~ universal approximator

# RBF Networks



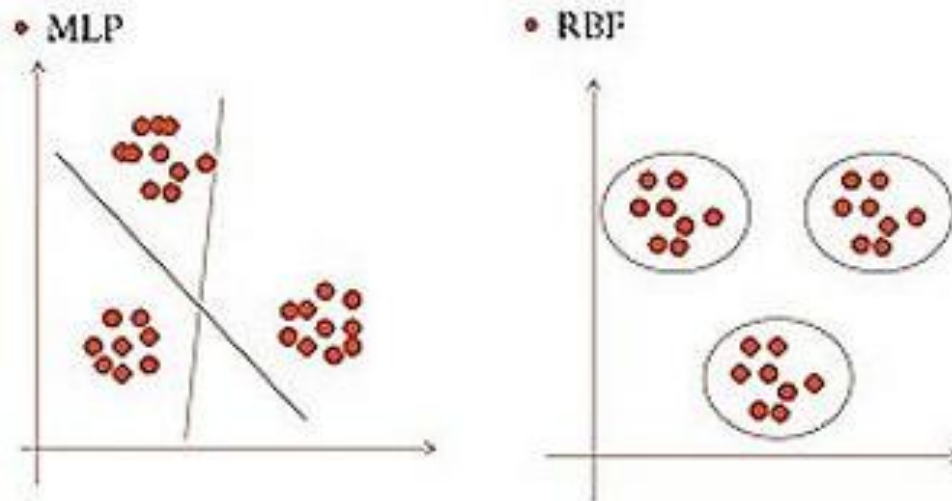
# MLP vs. RBF

- MLPs > RBFs

When the underlying characteristics feature of data is embedded deep inside very high dimensional sets for example, in image recognition etc.

- RBFs > MLPs

When low dimensional data where deep feature extraction is not required and results are directly correlated with the components of input vector.



# OUTLINE

- Nonlinear optimization in predictive learning
- Overview of ANN
- Stochastic approximation (gradient descent)
- Iterative methods
- Greedy optimization
- Summary and discussion

# Sequential estimation of model parameters

- Batch vs on-line (iterative) learning
  - Algorithmic (statistical) approaches ~ batch
  - Neural-network inspired methods ~ on-line

*BUT* the only difference is on the implementation level (so both types of methods should yield similar generalization)

- Recall ERM inductive principle (for regression):

$$R_{emp}(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n L(\mathbf{x}_i, y_i, \mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (y_i - f(\mathbf{x}_i, \mathbf{w}))^2$$

- Assume dictionary parameterization with fixed basis fcts

$$\hat{y} = f(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^m w_j g_j(\mathbf{x})$$



# Sequential (on-line) least squares minimization

- Training pairs  $\mathbf{x}(k), y(k)$  presented sequentially
- On-line update equations for minimizing empirical risk (MSE) wrt parameters  $\mathbf{w}$  are:

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \gamma_k \frac{\partial}{\partial \mathbf{w}} L(\mathbf{x}(k), y(k), \mathbf{w})$$

(gradient descent learning)

where the gradient is computed via the chain rule:

$$\frac{\partial}{\partial w_j} L(\mathbf{x}, y, \mathbf{w}) = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_j} = 2(\hat{y} - y)g_j(\mathbf{x})$$

the learning rate  $\gamma_k$  is a small positive value (decreasing with  $k$ )

# On-line least-squares minimization algorithm

- Known as **delta-rule** (Widrow and Hoff, 1960):

**Given** initial parameter estimates  $\mathbf{w}(0)$ , update parameters during each presentation of  $k$ -th training sample  $\mathbf{x}(k), y(k)$

- Step 1: **forward pass** computation

$$z_j(k) = g_j(\mathbf{x}(k)) \quad j = 1, \dots, m$$

$$\hat{y}(k) = \sum_{j=1}^m w_j(k) z_j(k) \quad - \text{estimated output}$$

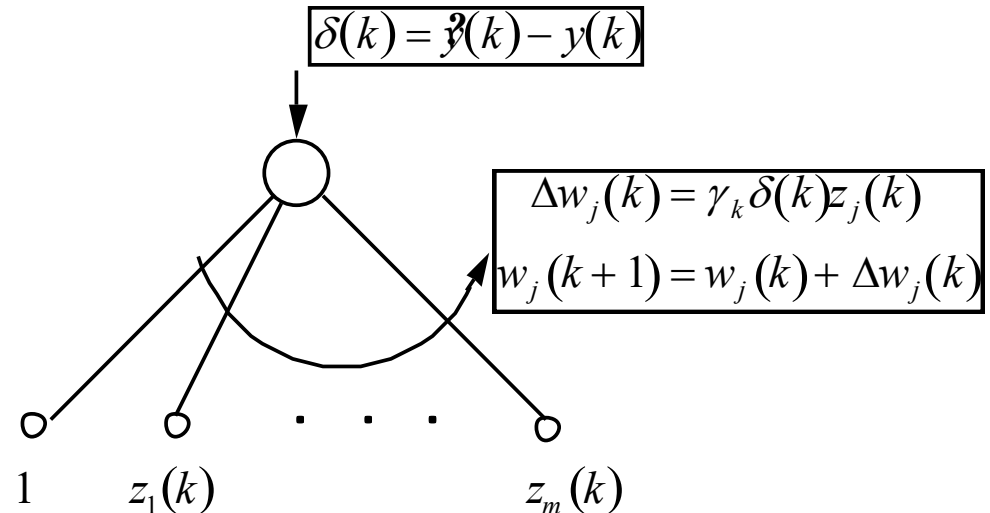
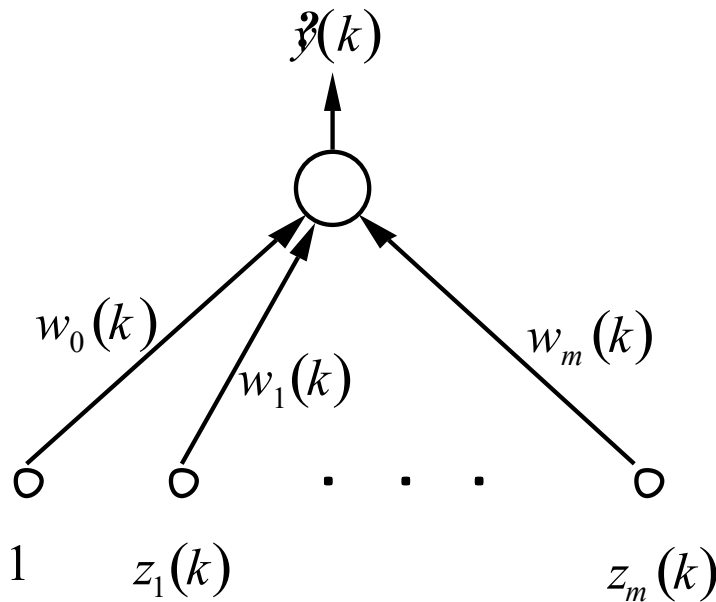
- Step 2: **backward pass** computation

$$\delta(k) = \hat{y}(k) - y(k) \quad - \text{error term (delta)}$$

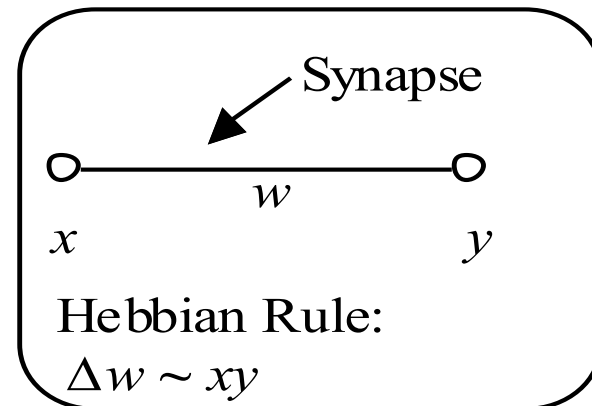
$$\mathbf{w}_j(k+1) = \mathbf{w}_j(k) - \gamma_k \delta(k) z_j(k), \quad j = 1, \dots, m$$

# Neural network interpretation of delta rule

- Forward pass**      **Backward pass**



- “Biological learning”**
  - parallel+ distributed comp.
  - can be extended to multiple-layer networks



# Backpropagation Training of MLP Networks

Consider a set of approximating functions

$$f(\mathbf{x}, \mathbf{w}, \mathbf{V}) = w_0 + \sum_{j=1}^m w_j g \left( v_{0j} + \sum_{i=1}^d x_i v_{ij} \right)$$

The risk functional is

$$R_{\text{emp}} = \sum_{i=1}^n (f(\mathbf{x}_i, \mathbf{w}, \mathbf{V}) - y_i)^2$$

The stochastic approximation procedure for minimizing this risk with respect to the parameters  $\mathbf{V}$  and  $\mathbf{w}$  is

$$\mathbf{V}(k+1) = \mathbf{V}(k) - \gamma_k \text{grad}_{\mathbf{V}} L(\mathbf{x}(k), y(k), \mathbf{V}(k), \mathbf{w}(k)),$$

$$\mathbf{w}(k+1) = \mathbf{w}(k) - \gamma_k \text{grad}_{\mathbf{w}} L(\mathbf{x}(k), y(k), \mathbf{V}(k), \mathbf{w}(k)), k = 1, \dots, n,$$

# Backpropagation Training of MLP Networks

The loss  $L$  is

$$L(\mathbf{x}(k), y(k), \mathbf{V}(k), \mathbf{w}(k)) = \frac{1}{2}(f(\mathbf{x}, \mathbf{w}, \mathbf{V}) - y)^2$$

The gradient of the loss  $L$  can be computed via the chain rule of derivatives if the approximating function is decomposed as

$$f(\mathbf{x}, \mathbf{w}, \mathbf{V}) = w_0 + \sum_{j=1}^m w_j g \left( v_{0j} + \sum_{i=1}^d x_i v_{ij} \right)$$

$$a_j = \sum_{i=0}^d x_i v_{ij}, \quad j = 1, \dots, m,$$

$$z_j = g(a_j), \quad j = 1, \dots, m,$$

$$z_0 = 1,$$

$$\hat{y} = \sum_{j=0}^m w_j z_j.$$

# Backpropagation Training of MLP Networks

Based on the chain rule, the relevant gradients are

$$\frac{\partial R}{\partial v_{ij}} = \frac{\partial R}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial a_j} \frac{\partial a_j}{\partial v_{ij}}$$

$$\frac{\partial R}{\partial w_j} = \frac{\partial R}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_j}.$$

We can calculate these partial derivatives using functions in the previous slide.

$$\begin{array}{ll} \frac{\partial R}{\partial \hat{y}} = \hat{y} - y & \frac{\partial \hat{y}}{\partial a_j} = g'(a_j)w_j \\ \frac{\partial a_j}{\partial v_{ij}} = x_i & \frac{\partial \hat{y}}{\partial w_j} = z_j \end{array} \quad \Rightarrow \quad \begin{array}{l} \frac{\partial R}{\partial v_{ij}} = (\hat{y} - y)g'(a_j)w_jx_i \\ \frac{\partial R}{\partial w_j} = (\hat{y} - y)z_j. \end{array}$$

# Backpropagation Training of MLP Networks

- With these **gradients** and the **stochastic approximation updating equations**, it is now possible to construct a computational procedure to **find the local minimum of the empirical risk**.

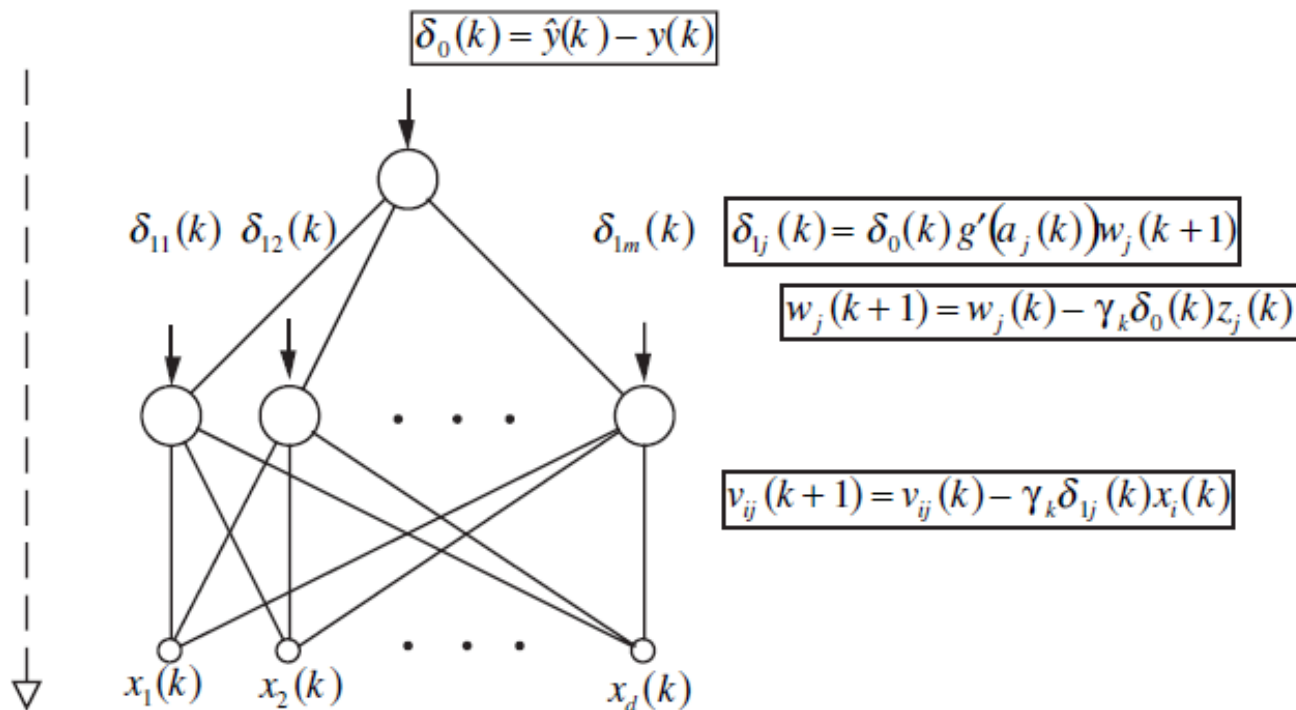
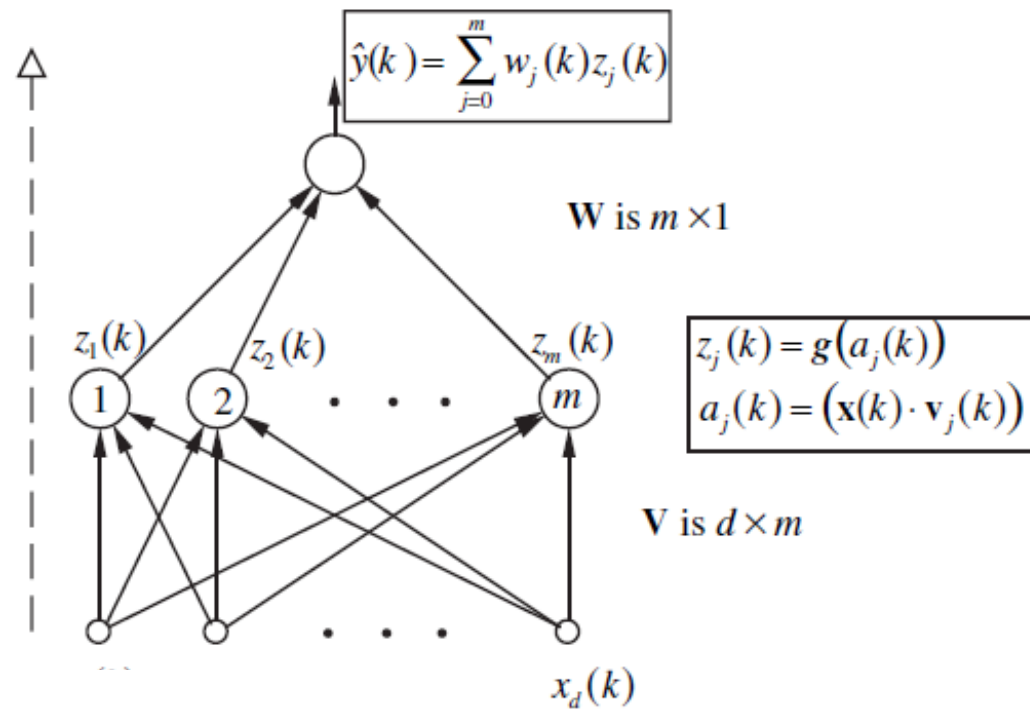
$$\frac{\partial R}{\partial v_{ij}} = (\hat{y} - y)g'(a_j)w_jx_i$$

$$\frac{\partial R}{\partial w_j} = (\hat{y} - y)z_j.$$

- Starting with an **initial guess for values  $w(0)$  and  $V(0)$** , the stochastic approximation procedure for **parameter (weight) updating** upon presentation of a sample  $(x(k), y(k))$  at **iteration** step  $k$  with learning rate  $\gamma_k$ .

# Backpropagation training:

(a) forward pass;  
(b) backward pass.





# Theoretical basis for on-line learning

- Standard inductive learning: given training data  $\mathbf{z}_1, \dots, \mathbf{z}_n$  find the model providing *min* of prediction risk  $R(\omega) = \int L(\mathbf{z}, \omega) p(\mathbf{z}) d\mathbf{z}$

- **Stochastic Approximation** guarantees minimization of risk (asymptotically):  
$$\omega(k+1) = \omega(k) - \gamma_k \mathbf{grad}_{\omega} L(\mathbf{z}_k, \omega(k))$$

under general conditions

$$\lim_{k \rightarrow \infty} \gamma_k = 0$$

on the learning rate:

$$\sum_{k=1}^{\infty} \gamma_k = \infty$$

$$\sum_{k=1}^{\infty} \gamma_k^2 < \infty$$

# Practical issues for on-line learning

- **Given finite training set** ( $n$  samples):  $\mathbf{Z}_1, \dots, \mathbf{Z}_n$   
this set is presented to a sequential learning algorithm many times.
  1. **Epoch** is the complete passing through of all the datasets exactly at once.
  2. **Batch** is the dataset that has been divided into smaller parts to be fed into the algorithm.
- **Learning rate schedule**: initially set large, then slowly decreasing with  $k$  (*iteration number*). Typically 'good' learning rate schedules are data-dependent.
- **Stopping conditions**:
  - (1) monitor the gradient (i.e., stop when the gradient falls below some small threshold)
  - (2) early stopping can be used for complexity control

# OUTLINE

- Nonlinear optimization in predictive learning
- Overview of ANN
- Stochastic approximation (gradient descent)
- **Iterative methods**
- Greedy optimization
- Summary and discussion

- **Dictionary representation** with fixed  $m$  :

$$f(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{j=0}^m w_j g_j(\mathbf{x}, \mathbf{v}_j)$$

***Iterative strategy*** for ERM (nonlinear optimization)

- **Step 1**: for current estimates of  $\hat{\mathbf{v}}_i$  update  $w_i$
- **Step 2**: for current estimates of  $\hat{w}_i$  update  $\mathbf{v}_i$

**Iterate** Steps (1) and (2) above

**Note**: - this idea can be implemented for different problems:  
e.g., unsupervised learning, supervised learning

***Specific update rules*** depend on the type of problem & loss fct.

# Iterative Methods

- Implement **iterative parameter estimation**.
- This leads to a generic parameter estimation scheme, where the **two steps (expectation and maximization)** are iterated until some **convergence criterion** is met.
- **Expectation-Maximization (EM)** developed/used in statistics.

# EM Methods for Density Estimation

Motivating Example:

- Have two coins: Coin1 and Coin2

$$p_1 = P(\text{H on Coin 1})$$

$$p_2 = P(\text{H on Coin 2})$$

1. Select a coin at random and flip that one coin  $m$  times.
2. Repeat this process  $n$  times.

- We have

$$\begin{array}{ccccccc} X_{11} & X_{12} & \cdots & X_{1m} & & Y_1 \\ X_{21} & X_{22} & \cdots & X_{2m} & & Y_2 \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ X_{n1} & X_{n2} & \cdots & X_{nm} & & Y_n \end{array}$$

$$Y_i \text{ live in } \{1, 2\} \quad X_{ij} = \begin{cases} 1 & , \text{ if the } j\text{th flip for the } i\text{th coin chosen is H} \\ 0 & , \text{ if the } j\text{th flip for the } i\text{th coin chosen is T} \end{cases}$$

- Note that all the  $X$ 's are independent and, in particular

$$X_{i1}, X_{i2}, \dots, X_{im} | Y_i = j \stackrel{iid}{\sim} \text{Bernoulli}(p_j)$$

# EM Methods for Density Estimation

- We can write out the joint pdf of all  $nm+n$  random variables and formally come up with MLEs for  $p_1$  and  $p_2$ .
- Call these MLEs  $\hat{p}_1$  and  $\hat{p}_2$ . They will turn out as expected:

$$\hat{p}_1 = \frac{\text{total \# of times Coin 1 came up H}}{\text{total \# times Coin 1 was flipped}}$$

$$\hat{p}_2 = \frac{\text{total \# of times Coin 2 came up H}}{\text{total \# times Coin 2 was flipped}}$$

- Now suppose that the  $Y_i$  are not observed but we still want MLEs for  $p_1$  and  $p_2$ . The data set now consists of only the  $X$ 's and is “incomplete”.
- The goal of the EM Algorithm is to find MLEs for  $p_1$  and  $p_2$  in this case.

# EM Methods for Density Estimation

- Let  $X$  be observed data, generated by some distribution depending on some parameters.
- Let  $Y$  be some “hidden” or “unobserved data” depending on some parameters.
- Let  $Z=(X,Y)$  represent the “complete” dataset.
- Assume we can write the pdf for  $Z$  as (depends on some parameter  $\theta$ )

$$f(z|\theta) = f(x, y|\theta) = f(y|x, \theta)f(x|\theta)$$

- We have the complete likelihood function:

$$L(\theta|Z) = L(\theta|X, Y) = f(X, Y|\theta)$$

- We have the incomplete likelihood function:

$$L(\theta|X) = f(X|\theta)$$



# EM Methods for Density Estimation

- The EM Algorithm is a numerical iterative for **finding an MLE of  $\theta$** . The rough idea is to start with an initial guess for  $\theta$  and to use this and the observed data  $X$  to “complete” the dataset by using  $X$  and the guessed  $\theta$  to postulate a value for  $Y$ , at which point we can then find an MLE for  $\theta$  in the usual way.
- We will use an initial guess for  $\theta$  and postulate an entire distribution for  $Y$ , ultimately averaging out the unknown  $Y$ .

# EM Methods for Density Estimation

- 1 Let  $k = 0$ . Give an initial estimate for  $\theta$ . Call it  $\hat{\theta}^{(k)}$ .
- 2 Given observed data  $X$  and assuming that  $\hat{\theta}^{(k)}$  is correct for the parameter  $\theta$ , find the conditional density  $f(y|X, \hat{\theta}^{(k)})$  for the completion variables.
- 3 Calculate the conditional expected log-likelihood or “ $Q$ -function”:

$$Q(\theta|\hat{\theta}^{(k)}) = \mathbb{E}[\ln f(X, Y|\theta)|X, \hat{\theta}^{(k)}].$$

Here, the expectation is with respect to the conditional distribution of  $Y$  given  $X$  and  $\hat{\theta}^{(k)}$  and thus can be written as

$$Q(\theta|\hat{\theta}^{(k)}) = \int \ln(f(X, y|\theta)) \cdot f(y|X, \hat{\theta}^{(k)}) dy.$$

(The integral is high-dimensional and is taken over the space where  $Y$  lives.)

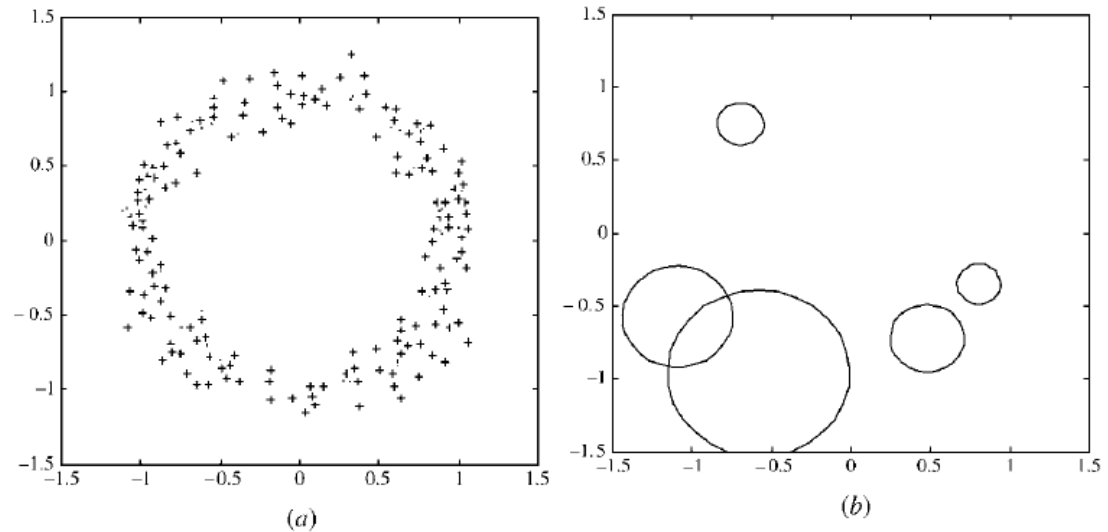
- 4 Find the  $\theta$  that maximizes  $Q(\theta|\hat{\theta}^{(k)})$ . Call this  $\hat{\theta}^{(k+1)}$ .

Let  $k = k + 1$  and return to Step 2.

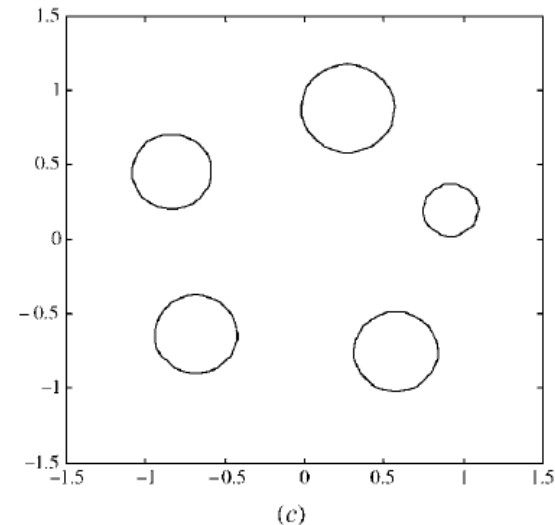
The EM Algorithm is iterated until the estimate for  $\theta$  stops changing.

$$||\hat{\theta}^{(k+1)} - \hat{\theta}^{(k)}|| < \varepsilon$$

# EM Methods for Density Estimation



- a) Two hundred data points drawn from a doughnut distribution.
- b) Initial configuration of five Gaussian mixtures.
- c) Configuration after 20 iterations of the EM algorithm



# OUTLINE

- Nonlinear optimization in predictive learning
- Overview of ANN
- Stochastic approximation (gradient descent)
- Iterative methods
- Greedy optimization
- Summary and discussion

# Greedy Optimization Methods

- Minimization of empirical risk (regression problems)

$$R_{emp}(\mathbf{V}, \mathbf{W}) = \frac{1}{n} \sum_{i=1}^n L(\mathbf{x}_i, y_i, \mathbf{V}, \mathbf{W}) = \frac{1}{n} \sum_{i=1}^n (y_i - f(\mathbf{x}_i, \mathbf{V}, \mathbf{W}))^2$$

where the model  $f(\mathbf{x}, \mathbf{V}, \mathbf{W}) = \sum_{j=1}^m w_j g_j(\mathbf{x}, \mathbf{v}_j)$

- Greedy Optimization Strategy

basis functions are **estimated sequentially**, one at a time,  
i.e., the training data is represented as  
**structure** (model fit) + **noise** (residual):

(1) DATA = (model) FIT 1 + RESIDUAL 1

(2) RESIDUAL 1 = FIT 2 + RESIDUAL 2

and so on. The final model for the data will be

**MODEL = FIT 1 + FIT 2 + ....**

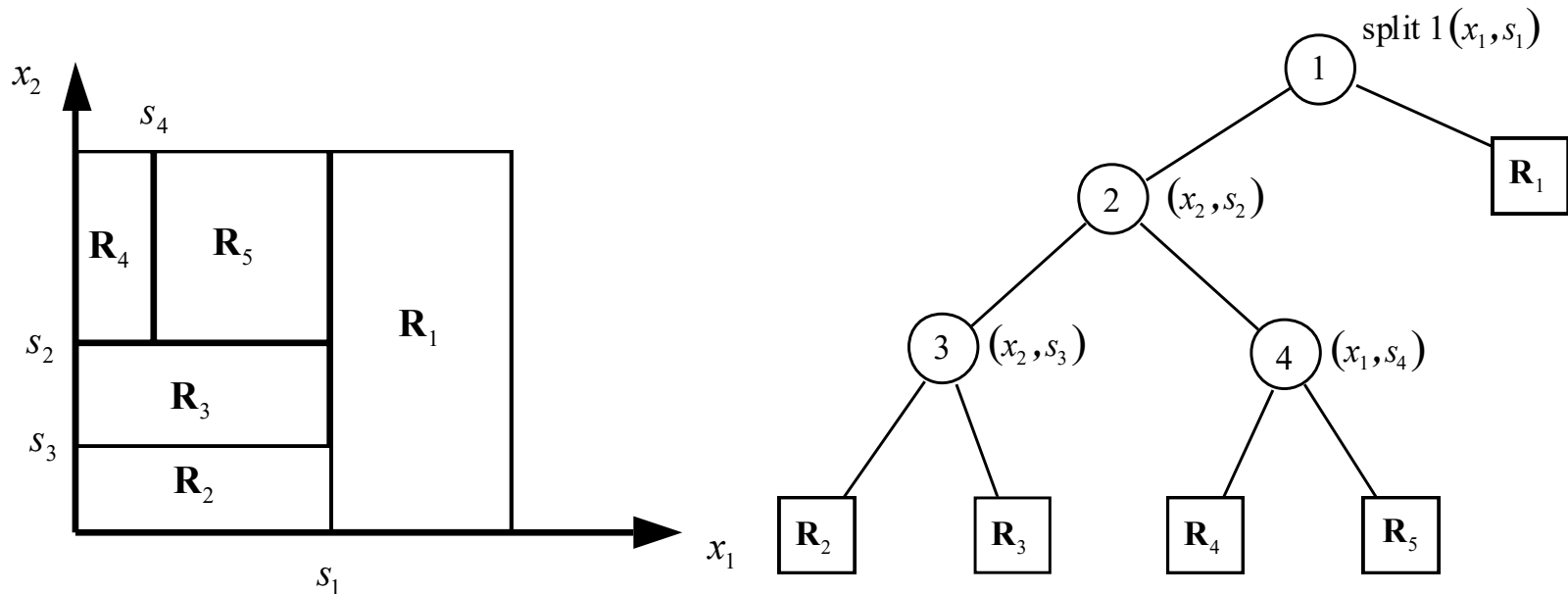
- **Advantages:** computational speed, interpretability

# Classification and Regression Trees (CART)

- Minimization of empirical risk (squared error) via partitioning of the input space into regions

$$f(\mathbf{x}) = \sum_{j=1}^m w_j I(\mathbf{x} \in \mathbf{R}_j)$$

- Example of CART partitioning for a function of 2 inputs



# Growing CART

- **Recursive partitioning** for estimating regions (via binary splitting)
- **Initial Model** ~ Region  $\mathbf{R}_0$  (the whole input domain) is divided into two regions  $\mathbf{R}_1$  and  $\mathbf{R}_2$
- A split is defined by **one of the inputs( $k$ )** and **split point  $s$**
- **Optimal values** of  $(k, s)$  chosen so that splitting a region into two daughter regions **minimizes empirical risk**
- **Issues:**
  - efficient implementation (selection of opt. split)
  - optimal tree size ~ model selection (complexity control)

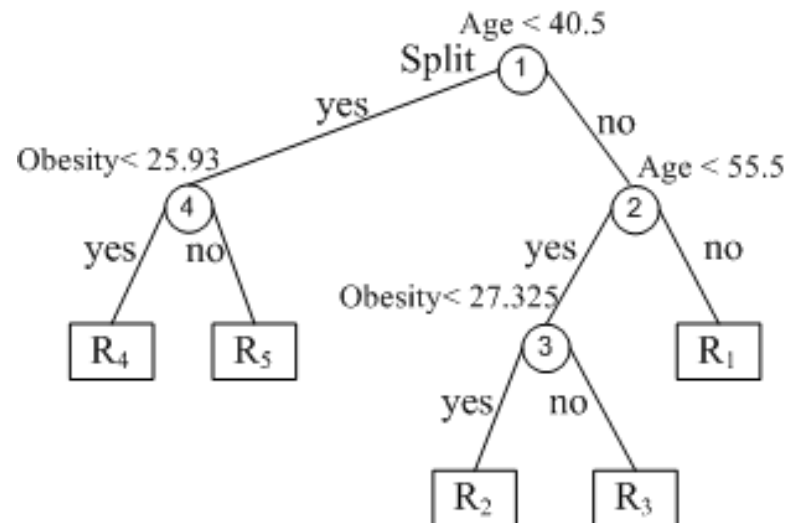
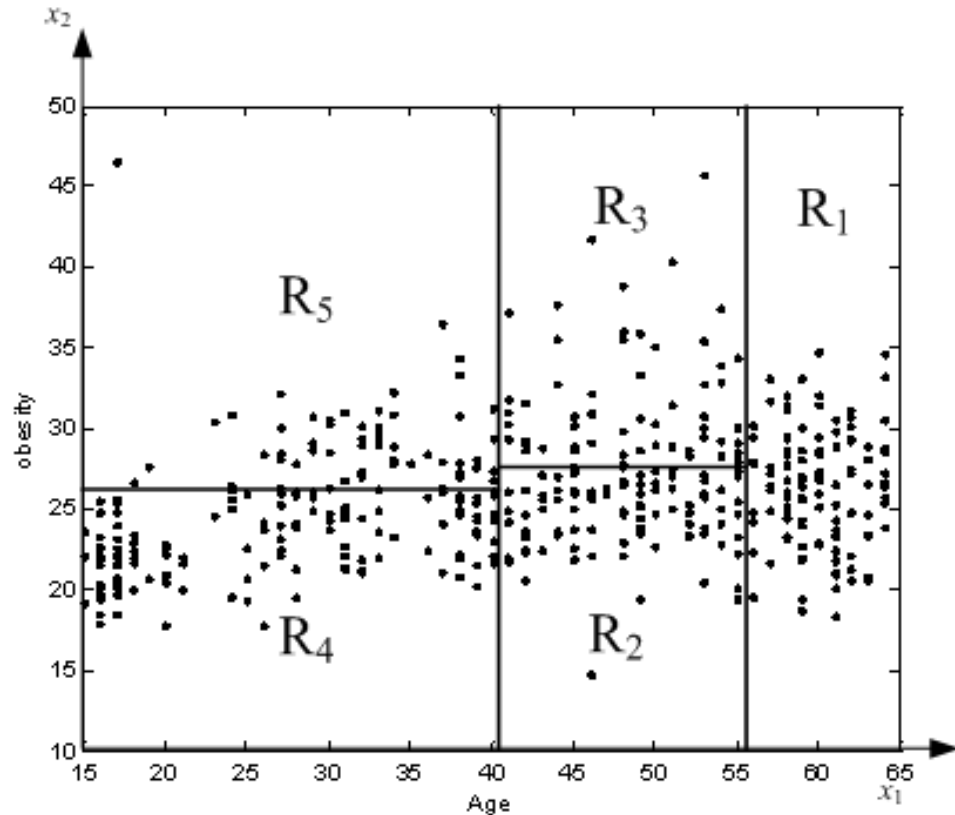
# Growing CART

- Advantages:
  1. Results are simplistic.
  2. Classification and regression trees are Nonparametric and Nonlinear.
  3. Classification and regression trees implicitly perform feature selection.
  4. Outliers have no meaningful effect on CART.
  5. It requires minimal supervision and produces easy-to-understand models.
- Limitations:
  1. Overfitting.
  2. High Variance.
  3. Low bias.
  4. The tree structure may be unstable.



# CART Example

- CART model** for estimating *Systolic Blood Pressure (SBP)* as a function of *Age* and *Obesity* in a population of males in S. Africa



# CART model selection

- Model selection strategy
  - (1) **Grow a large tree** (subject to min leaf node size)
  - (2) **Tree pruning** by selectively merging tree nodes
- **The final model** ~ minimizes **penalized risk**
$$R_{pen}(\omega, \lambda) = R_{emp}(\omega) + \lambda \|T\|$$

where empirical risk ~ MSE  
number of leaf nodes ~  $\|T\|$   
regularization parameter ~  $\lambda$
- *Note:* **larger**  $\lambda \rightarrow$  **smaller** trees
- In practice: often user-defined (*splitmin* in Matlab)

# Pitfalls of greedy optimization

- Greedy binary splitting strategy may yield:
  - sub-optimal solutions (regions  $\mathbf{R}_i$ )
  - solutions very sensitive to random samples (especially with correlated input variables)

- Counterexample for CART:

estimate function  $f(a,b,c)$ :

- which variable for the first split?

Choose a  $\rightarrow$  3 errors

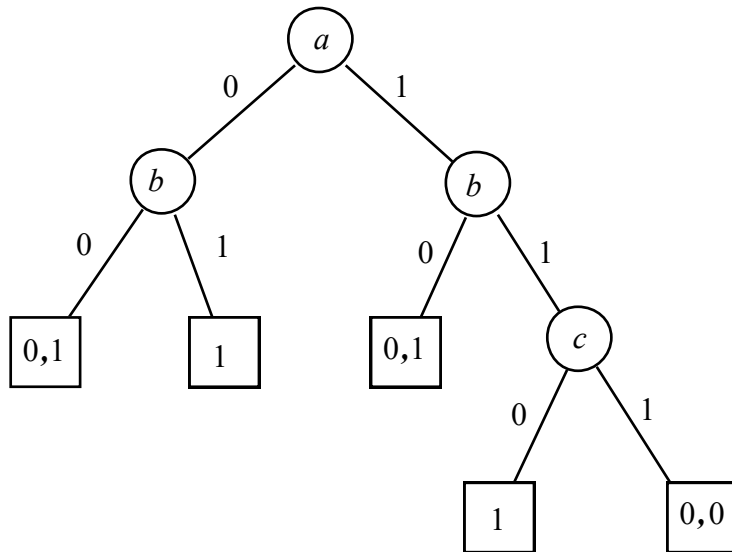
Choose b  $\rightarrow$  4 errors

Choose c  $\rightarrow$  4 errors

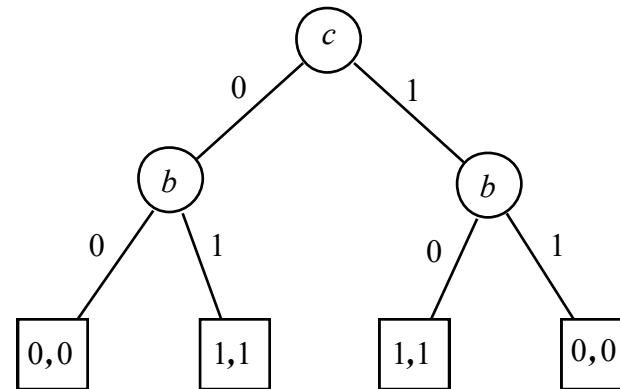
y	a	b	c
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
1	0	1	0
1	1	1	0
0	1	1	1
0	1	1	1

# Counter example (cont'd)

(a) Suboptimal tree by CART



(b) Optimal binary tree

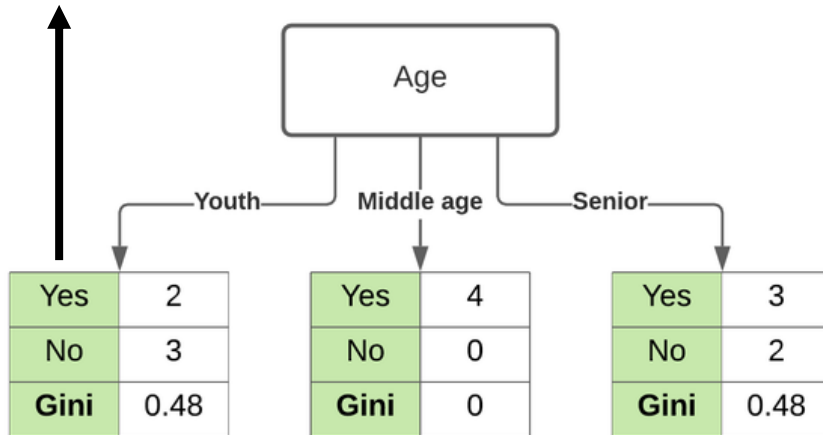


# Gini score

$$Gini = 1 - \sum_{i=1}^n (p_i)^2$$

$p_i$  is the probability of an object being classified to a particular class.

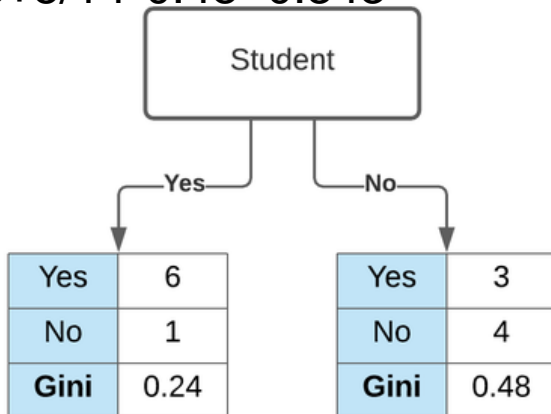
$$1 - (2/5)^2 - (3/5)^2 = 1 - 4/25 - 9/25 = 12/25 = 0.48$$



Gini Impurity for Age is 0.343

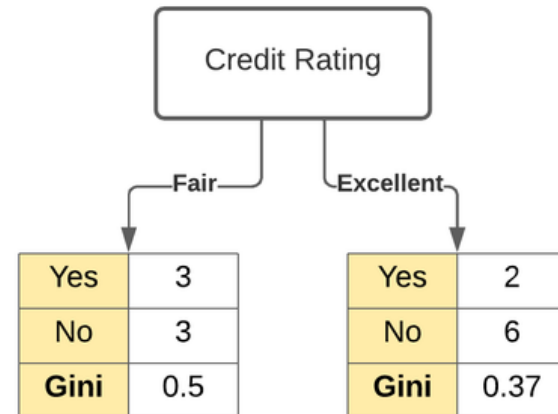
Gini Impurity for Income is 0.440

$$5/14 * 0.48 + 5/14 * 0.48 = 0.343$$



Gini Impurity for Student is 0.367

**Best**



Gini Impurity for Credit Rating is 0.429

# Backfitting Algorithm

- Consider regression estimation of a function of two variables of the form  $y = g_1(x_1) + g_2(x_2) + \text{noise}$  from training data  $(x_{1i}, x_{2i}, y_i) \quad i = 1, 2, \dots, n$

For example  $t(x_1, x_2) = x_1^2 + \sin(2\pi x_2) \quad \mathbf{x} \in (0, 1)^2$

**Backfitting method:**

- (1) estimate  $g_1(x_1)$  for fixed  $g_2$
- (2) estimate  $g_2(x_2)$  for fixed  $g_1$

*iterate above two steps*

- Estimation via minimization of empirical risk

$$\begin{aligned} R_{emp}(g_1(x_1), g_2(x_2)) &= \frac{1}{n} \sum_{i=1}^n (y_i - g_1(x_{1i}) - g_2(x_{2i}))^2 \\ (\text{first\_iteration}) &= \frac{1}{n} \sum_{i=1}^n ([y_i - g_2(x_{2i})] - g_1(x_{1i}))^2 \\ &= \frac{1}{n} \sum_{i=1}^n (r_i - g_1(x_{1i}))^2 \end{aligned}$$

# Backfitting Algorithm(cont'd)

- Estimation of  $g_1(x_1)$  via minimization of MSE

$$R_{emp}(g_1(x_1)) = \frac{1}{n} \sum_{i=1}^n (r_i - g_1(x_{1i}))^2 \rightarrow \min$$

- This is a univariate regression problem of estimating  $g_1(x_1)$  from  $n$  data points  $(x_{1i}, r_i)$  where  $r_i = y_i - g_2(x_{2i})$
- Can be estimated by smoothing (kNN regression)
- Estimation of  $g_2(x_2)$  (second iteration) proceeds in a similar manner, via minimization of
$$R_{emp}(g_2(x_2)) = \frac{1}{n} \sum_{i=1}^n (r_i - g_2(x_{2i}))^2 \quad \text{where } r_i = y_i - g_1(x_{1i})$$
- **Backfitting ~gradient descent in the function space**

# OUTLINE

- Nonlinear optimization in predictive learning
- Overview of ANN
- Stochastic approximation (gradient descent)
- Iterative methods
- Greedy optimization
- Summary and discussion



# Summary

- Different interpretation of optimization
- Consider dictionary parameterization

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=0}^m w_i g(\mathbf{x}, \mathbf{v}_i)$$

- VC-theory: implementation of SRM
- Gradient descent + iterative optimization
  - SRM structure is specified a priori
  - selection of  $m$  is separate from ERM
- Greedy optimization strategy
  - no a priori specification of a structure
  - model selection is a part of optimization

# Summary (cont'd)

- Interpretation of greedy optimization

$$f_m(\mathbf{x}, \mathbf{w}, \mathbf{V}) = \sum_{i=0}^m w_i g(\mathbf{x}, \mathbf{v}_i)$$

- Statistical strategy for iterative data fitting

(1) Data = (model) Fit1 + Residual\_1

(2) Residual\_1 = (model) Fit2 + Residual\_2

.....

→ Model = Fit1 + Fit2 + ...

- This has superficial similarity to SRM