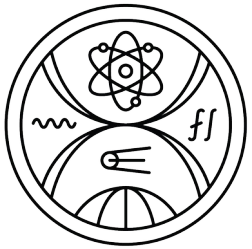
Introduction to Computational Intelligence

Supervised feedforward neural networks

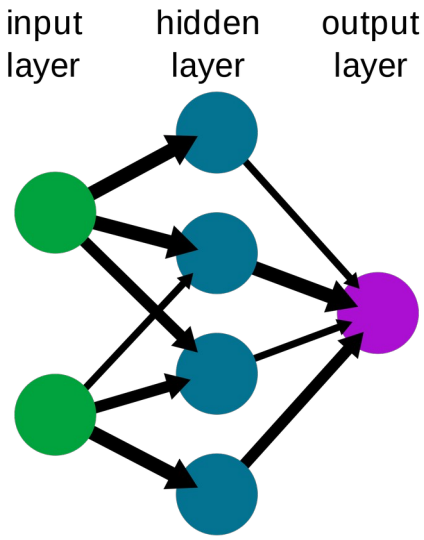


Igor Farkaš
Centre for Cognitive Science
Comenius University Bratislava

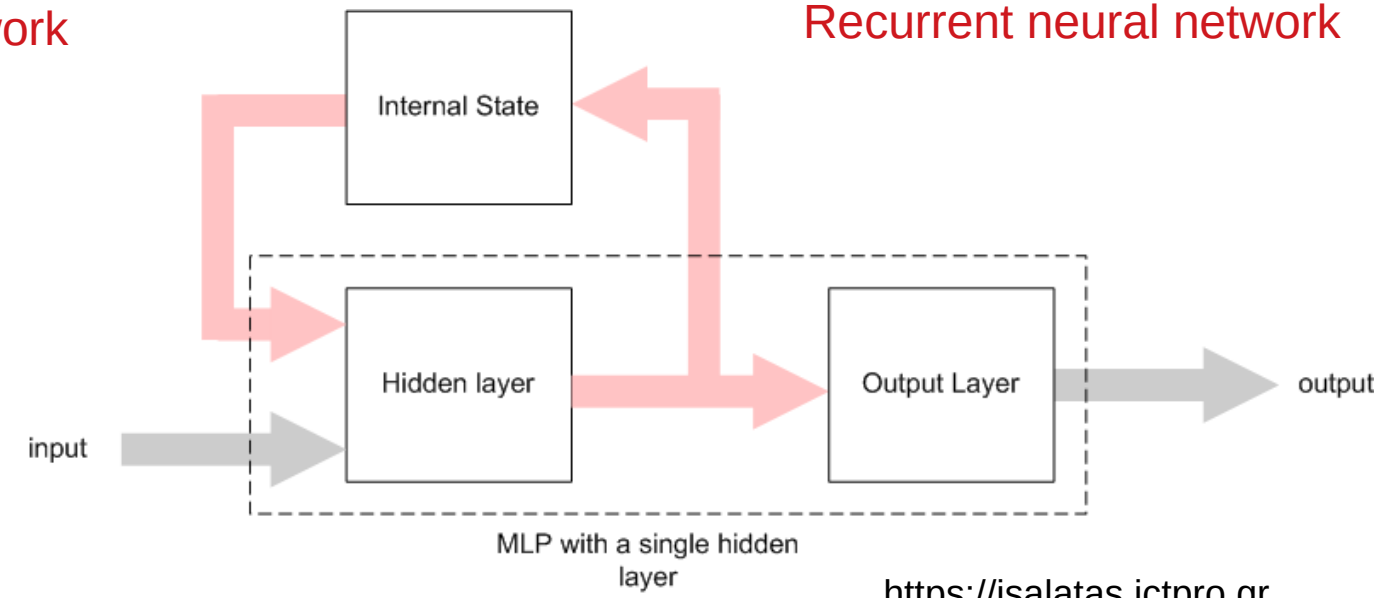
Artificial neural network as an agent

- ANNs can learn in various ways, here we focus on supervised learning (good for classification or regression tasks)
- ANN as a (passive) black box
- Assumption: environment provides all input-target pairs, the ANN “wants” to learn from the scratc

Feedforward neural network



Recurrent neural network

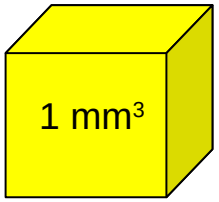


https://jsalatas.ictpro.gr

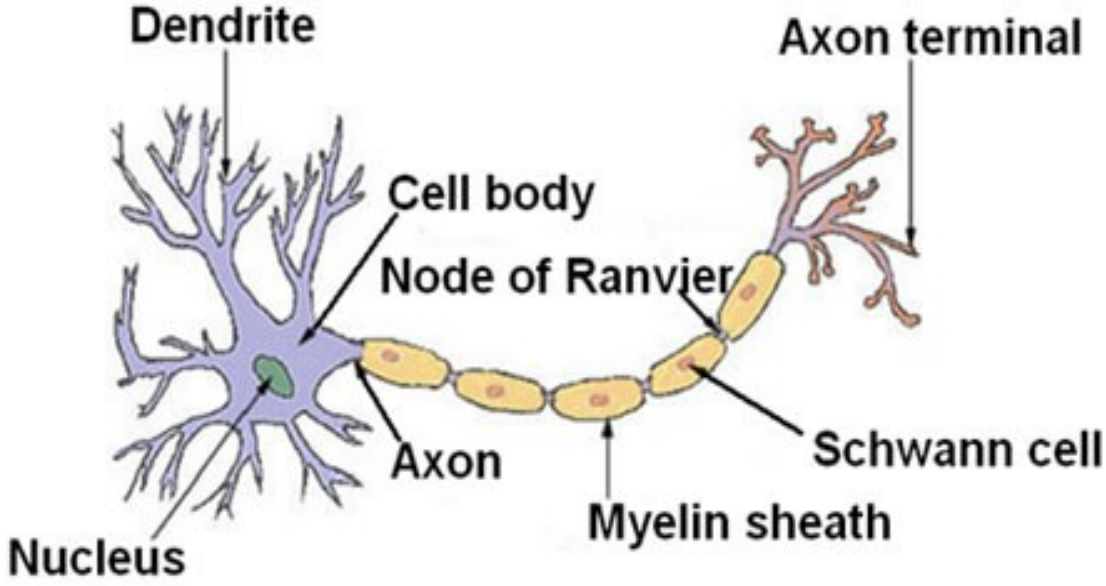
Core properties of ANNs

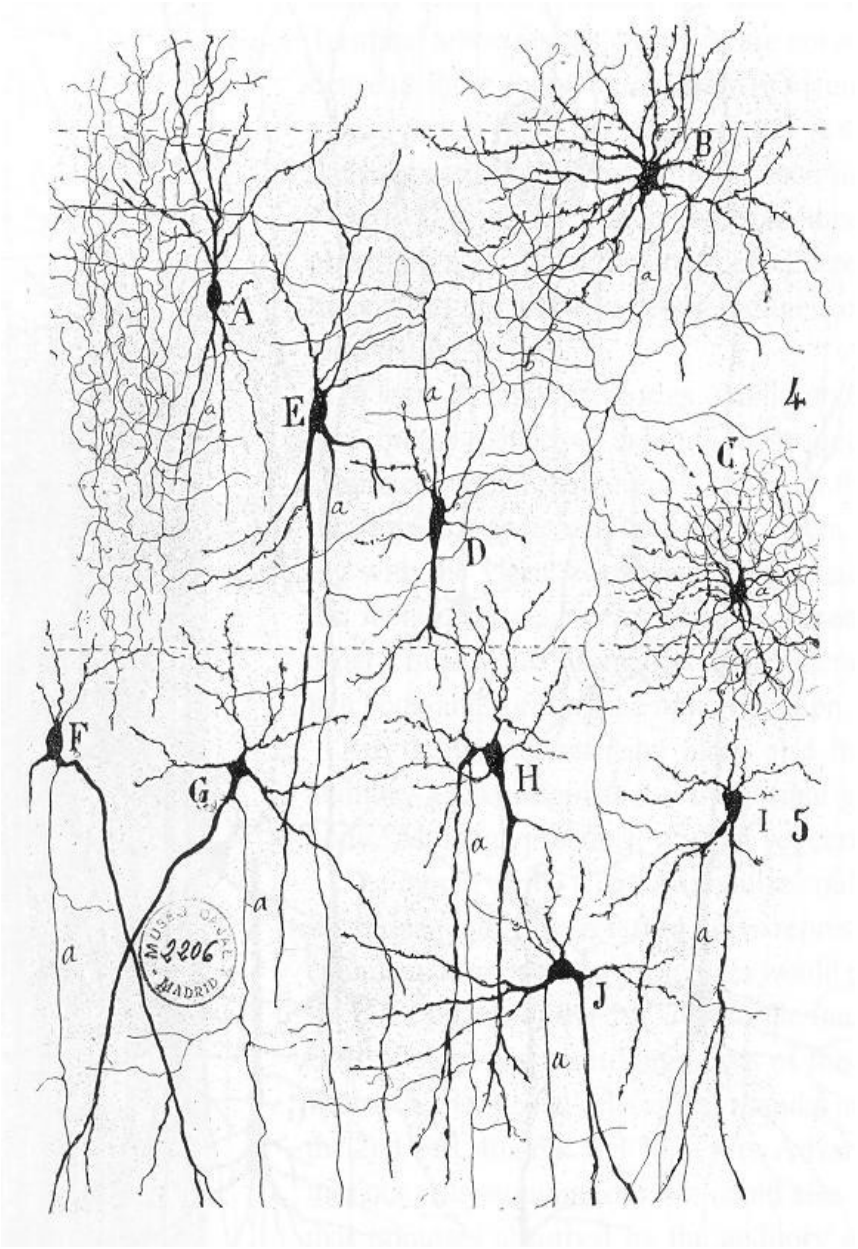
- Neurobiological inspiration with abstraction
- Simple computing elements (neurons)
- Nonlinear interactions between them (excitation, inhibition)
- Ability to learn (elementary changes to knowledge representation)
- … based on experience (training examples)
- Distributedness and robustness
- Realisation of (continuous) input-output mappings
- … in Euclidean space
- … in case of recurrent models, with context dependency

Biological neuron



10^5 neurons
3 km axons





From R. y Cajal: Texture of the Nervous System of Man and the Vertebrates (illustrates the diversity of neuronal morphologies in the auditory cortex).

Dentrites = inputs, axon = output

Type of artificial neuron – discrete perceptron

- Input ***x***, weights ***w***, output *y*
- Output calculation:

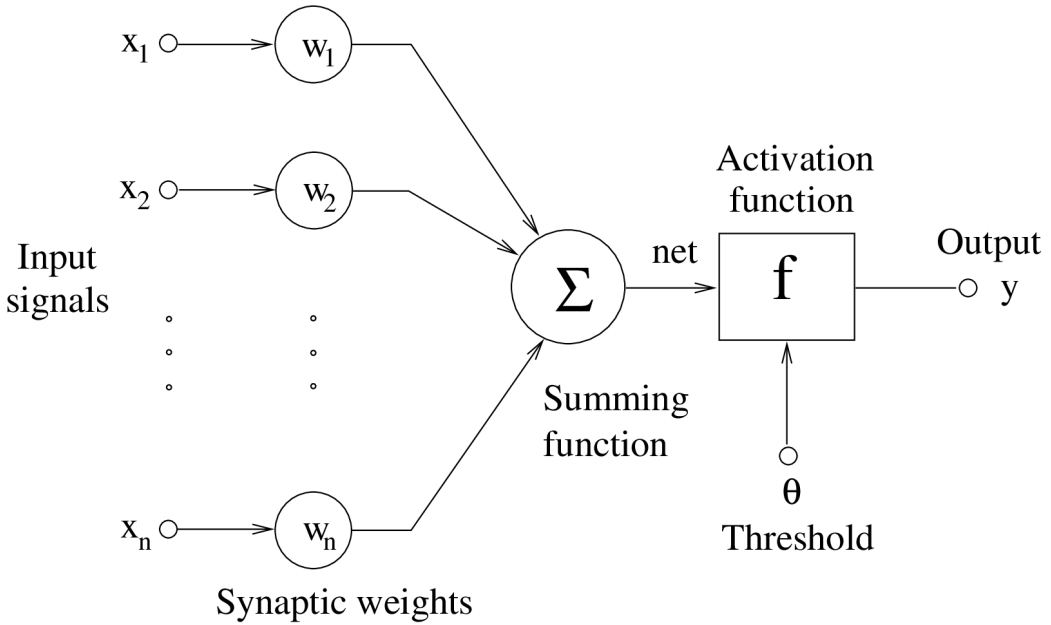
$$y = f(\sum_{j=1}^{n} w_j x_j - \theta)$$

- *f* = threshold function: unipolar {0,1} or bipolar {-1,+1}
- Supervised learning – uses teacher signal *d* (desired value)
- Learning rule:

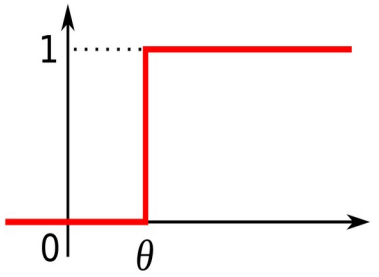
$$w_j(t+1) = w_j(t) + \alpha\ (d - y)\ x_j$$

learning rate

Basic computing element



Threshold act. func.



(Rosenblatt, 1962)

Summary of perceptron algorithm

Given: training data: input-target $\{\boldsymbol{x}, d\}$ pairs, unipolar perceptron
Initialization: randomize weights, set learning rate, $E = 0$.

Training:

1. choose input $\boldsymbol{x}$, compute output y
2. evaluate error function $e(t) = ½ (d - y)^2$, $E = E + e(t)$
3. adjust weights using delta rule (if $e(t) > 0$)
4. if all patterns used, then goto 5, else go to 1
5. if $E == 0$ (all patterns in the set classified correctly), then end

   else shuffle inputs, $E = 0$, go to 1

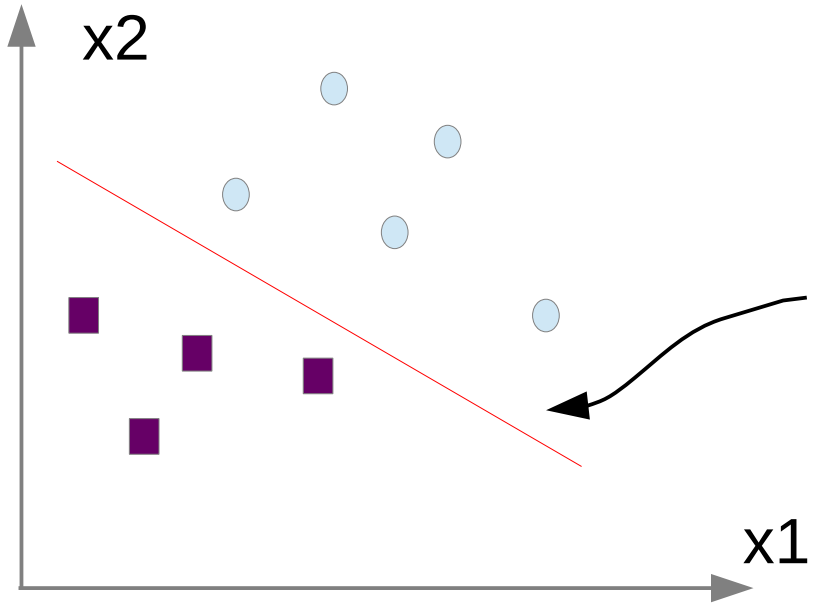
Perceptron classification capacity

$w_1x_1 + w_2x_2 + ... + w_nx_n = \theta$

n-dim. case.

linear separability of two classes

2D example



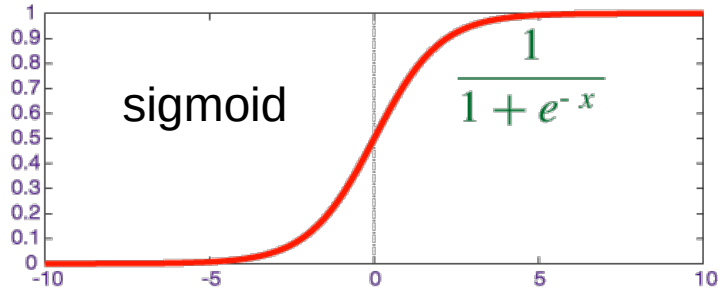
Fixed-increment convergence theorem (Rosenblatt, 1962)*: "Let the classes A and B are finite and linearly separable, then perceptron learning algorithm converges (updates its weight vector) in a finite number of steps."*

Type of artificial neuron – continuous perceptron

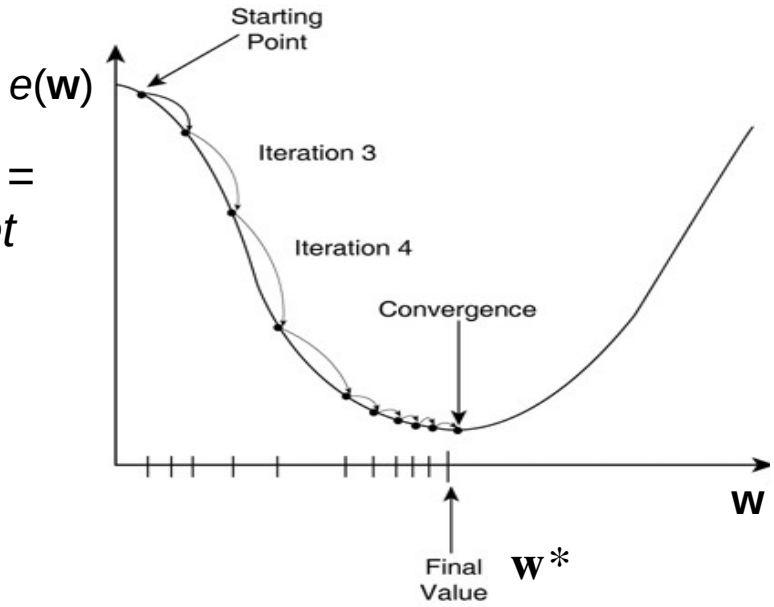
- Uses a nonlinear unit with sigmoid act. function: $y = 1 / (1+e^{-net})$

  - has nice properties (bounded, monotonous, differentiable)
- Let us consider quadratic error function: $e(\boldsymbol{w}) = \frac{1}{2} (d - y)^2$
- Error minimization: necessary condition $e(\mathbf{w}^*) \leq e(\mathbf{w})$
- (stochastic, online) gradient descent learning:

$$w_j(t+1) = w_j(t) + \alpha \ (d^{(p)} - y^{(p)}) \, f'(net) \, x_j$$



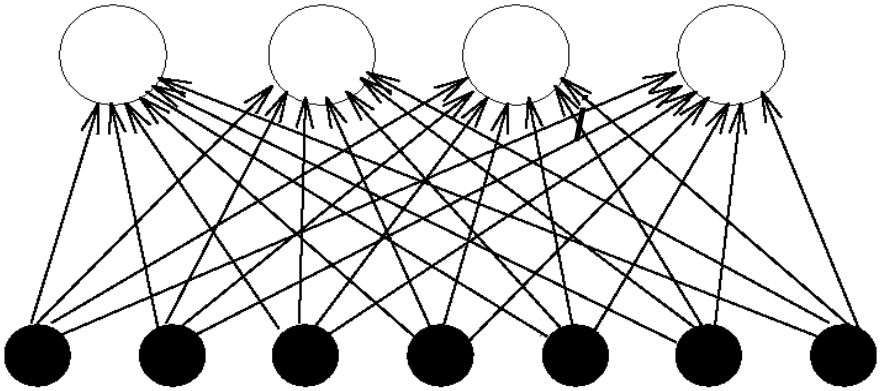
f ’(*net*) = d*f* / d*net* =
deriv. *of f w.r.t. net*



Perceptron as a binary classifier

Single-layer perceptrons

- We use *N* perceptrons working independently
- can classify linearly separable classes

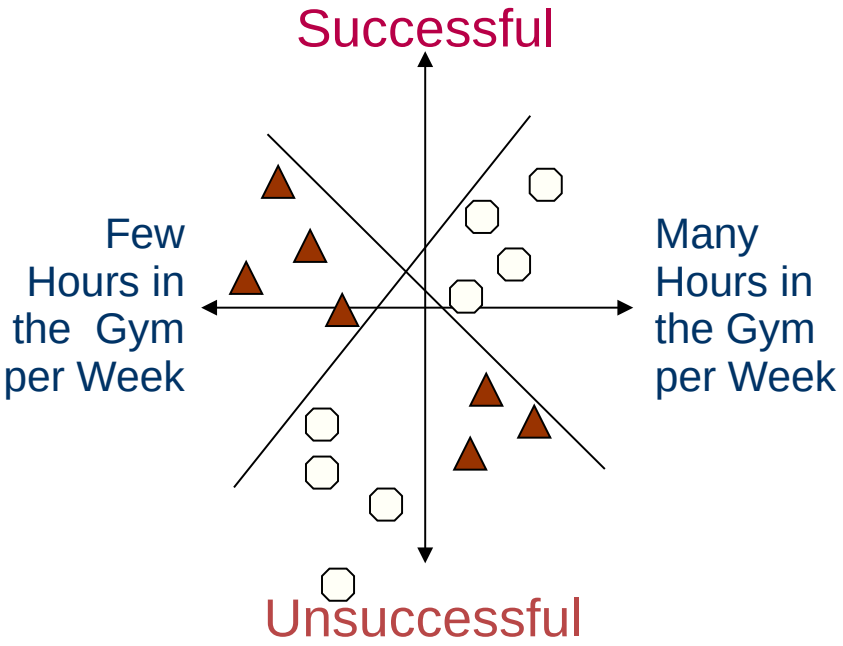


- *N* classes => *N* neurons, each representing one class
- Learning rule is the same: (except that weights have 2 indices)

$$w_{ij}(t+1) = w_{ij}(t) + \alpha \ (d_i - y_i) f_i{}' x_j$$

Perceptron limits

- Simple perceptron cannot separate linearly non-separable classes



Footballers

Academics

…despite the simplicity of their relationship:

Academics = Successful XOR Gym

This failure caused the loss of interest in connectionism by many (in 1970s) resuling in the 1st AI winter. Many categories in real problems turned out to be linearly non-separable.

M. Minsky & S. Papert (1969). *Perceptrons*, MIT Press, Cambridge, MA.

Multi-layer perceptrons

- Generalization of simple perceptrons
- MLPs can learn more complex mappings
- Features:
  - contain hidden-layer(s)
  - neurons have non-linear activation function (e.g. logistic)
  - full connectivity b/w layers
- (supervised) error "back-propagation" training algorithm introduced
- Best known from 1986: Rumelhart & McClelland: *Parallel distributed processing*
  - algorithm described earlier by Werbos (1974)
- response to earlier critique of perceptrons (Minsky & Papert, 1969)

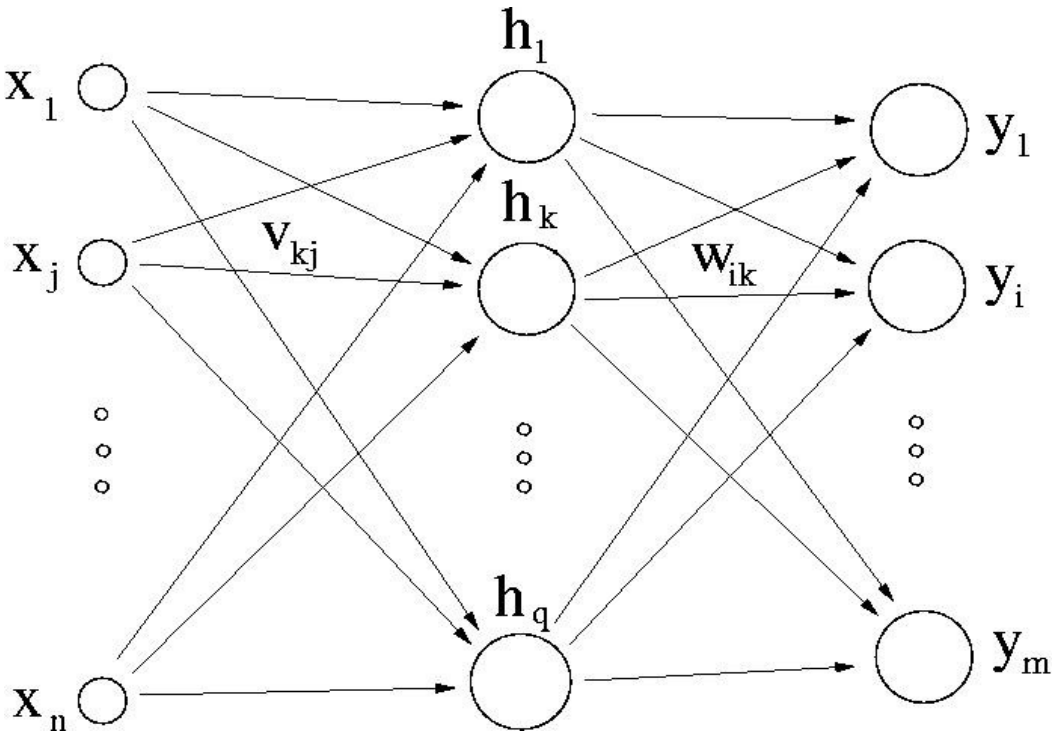
Two-layer perceptron

- Inputs $\boldsymbol{x}$, weights $\boldsymbol{w}$, $\boldsymbol{v}$, outputs $\boldsymbol{y}$
- Nonlinear activation function f
- Unit activation:

$$h_k = f\left(\sum_{j=1}^{n+1} v_{kj} x_j\right)$$

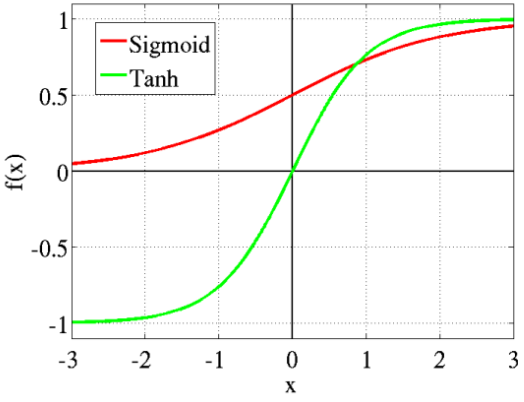
$$y_i = f\left(\sum_{k=1}^{q+1} w_{ik} h_k\right)$$

- Bias input: $x_{n+1} = h_{q+1} = -1$
- Activation function examples:



$$f(net) = \sigma(net) = \frac{1}{1+e^{-net}}$$

$$f(net) = \tanh(net) = \frac{e^{net} - e^{-net}}{e^{net} + e^{-net}} = \frac{2}{1+e^{-2net}} - 1$$



How to apply error?

- Output layer – application of delta rule
  - Because targets are known
- How to compute error at hidden layer(s)?
- Instantaneous output error (for pattern *p*): $e^{(p)} = \frac{1}{2} \sum_i (d_i^{(p)} - y_i^{(p)})^2$
- We will show that error can be back-propagated across layers backwards
- At each layer the (local) weight correction has this form:

  (weight change) = (learning rate)*(unit error)*(unit input)

Learning equations for original BP

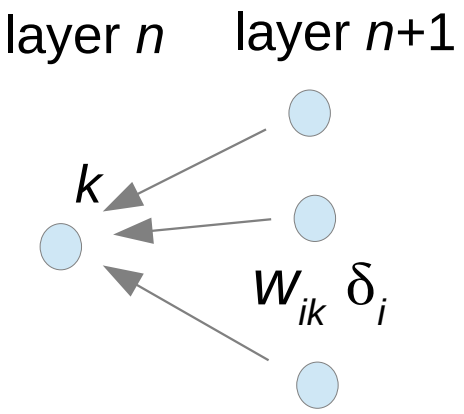
Hidden-output weights:

$$w_{ik}(t+1) = w_{ik}(t) + \alpha\delta_i h_k \quad \text{where} \quad \delta_i = (d_i - y_i) f_i'$$

Input-hidden weights:

$$v_{kj}(t+1) = v_{kj}(t) + \alpha\ \delta_k x_j \quad \text{where} \quad \delta_k = (\Sigma_i w_{ik} \delta_i) f_k'$$

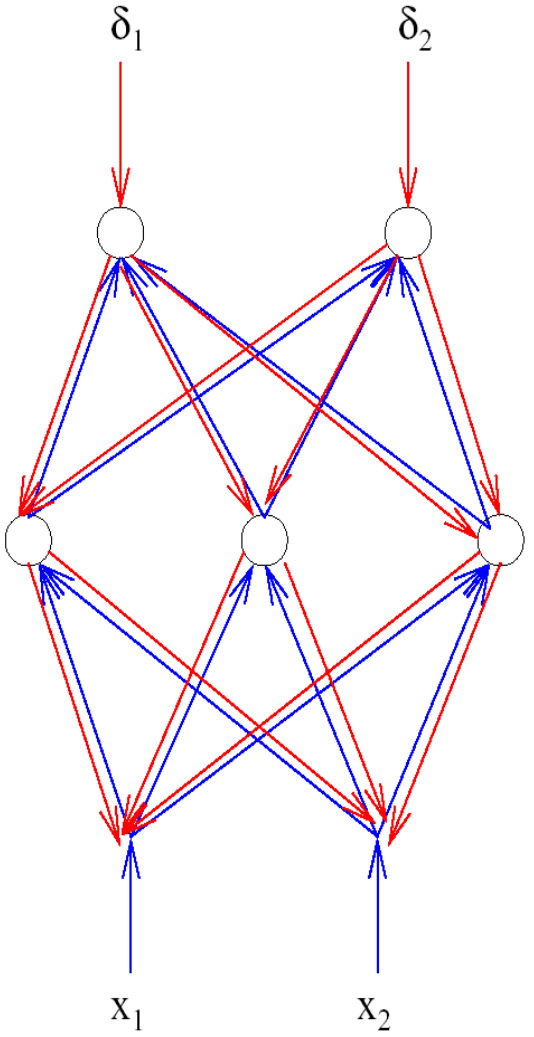
- BP is the first-order algorithm, based on the method of the steepest descent in the weight space
- smoothness of the trajectory depends on the learning rate



Summary of back-propagation algorithm

Given: training data: input-target $\{\boldsymbol{x}^{(p)}, \boldsymbol{d}^{(p)}\}$ pairs
Initialization: randomize weights, set learning parameters
Training:

1. choose input $\boldsymbol{x}^{(p)}$, compute outputs $\boldsymbol{y}^{(p)}$ (forward pass),
2. evaluate chosen error function $e(t)$, $E \leftarrow E + e(t)$
3. compute δ_i , δ_k (backward pass)
4. adjust weights Δw_{ik} and Δv_{kj}
5. if all patterns used, then goto 6, else go to 1
6. if stopping_criterion is met, then end
   else permute inputs and go to 1

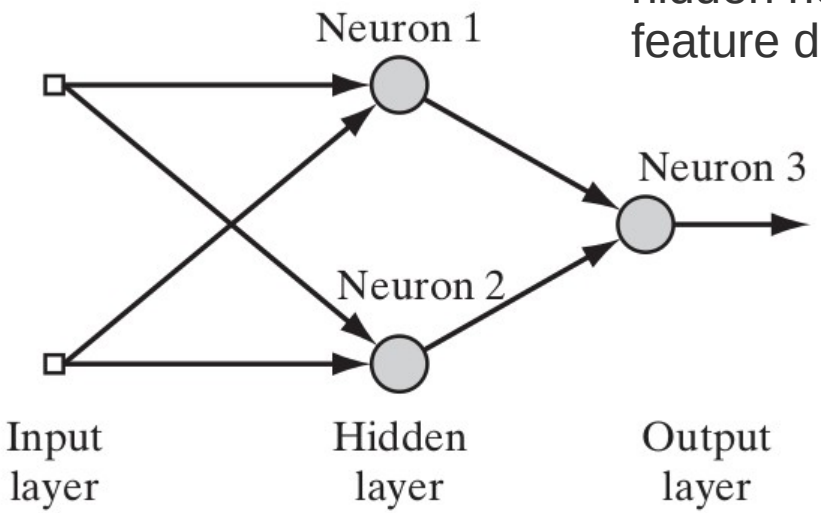


No well-defined stopping criteria exist. Suggestions:

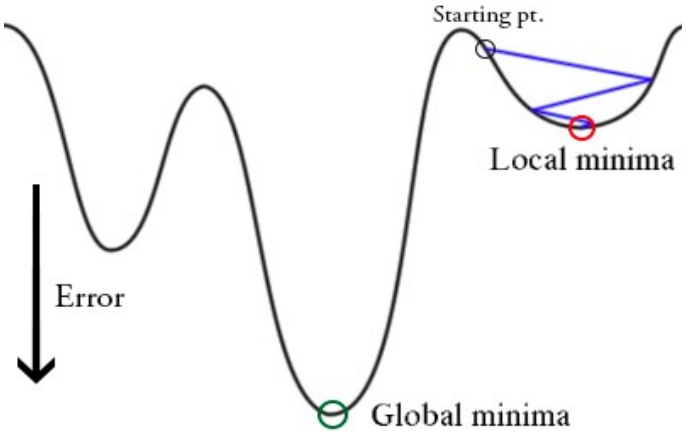
- when change in E_{epoch} is sufficiently small (<1%)
- when generalization performance is adequate

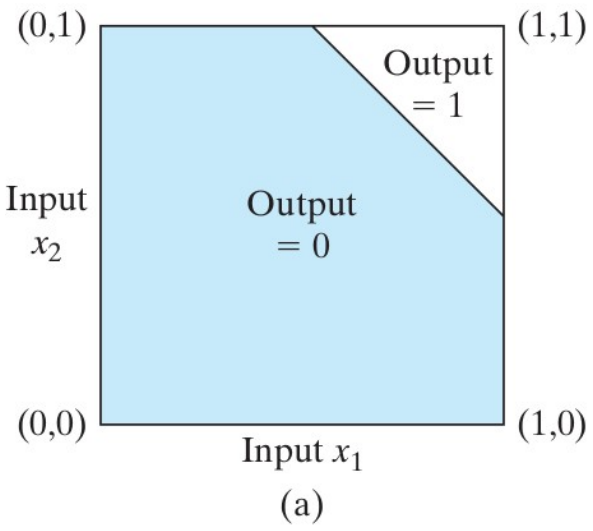
XOR problem solved with MLP

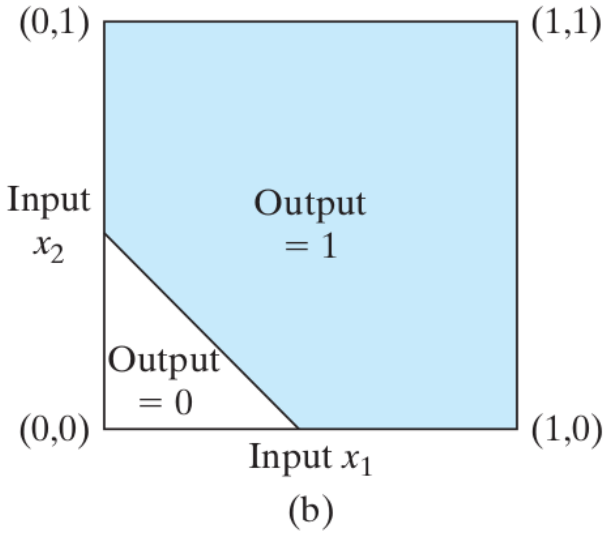
hidden neurons as feature detectors

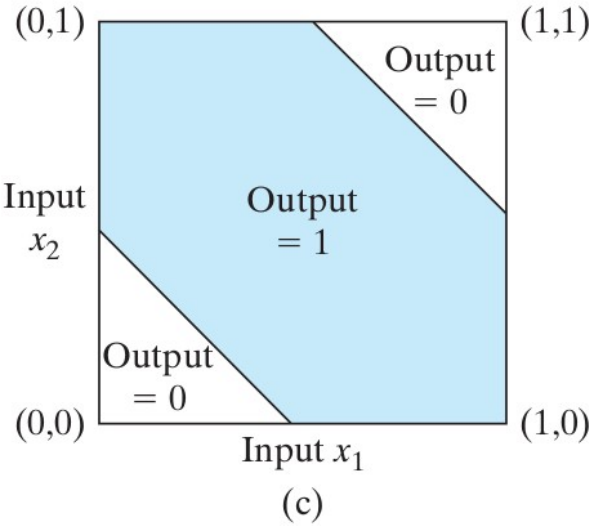


Error function in MLPs

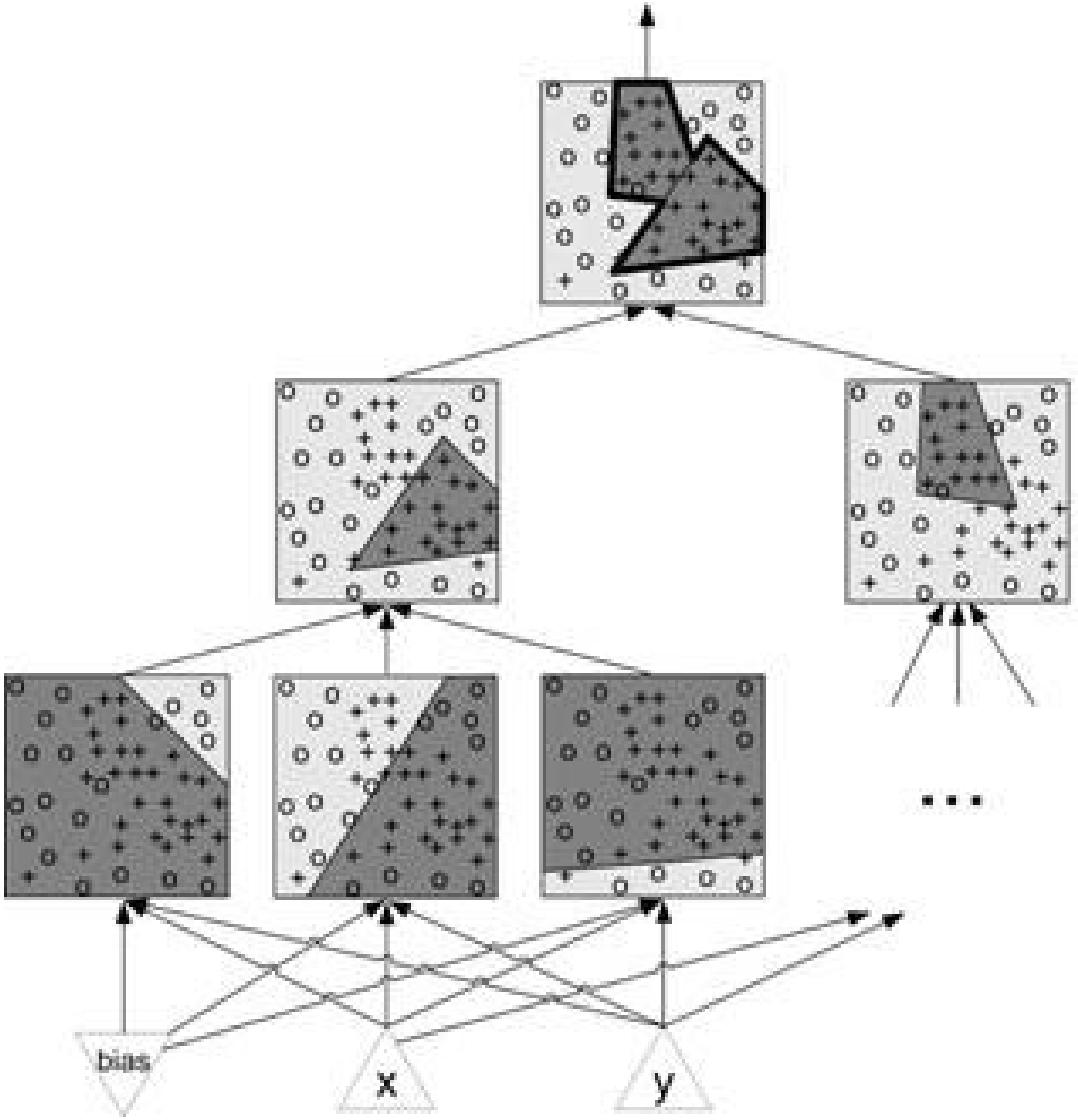


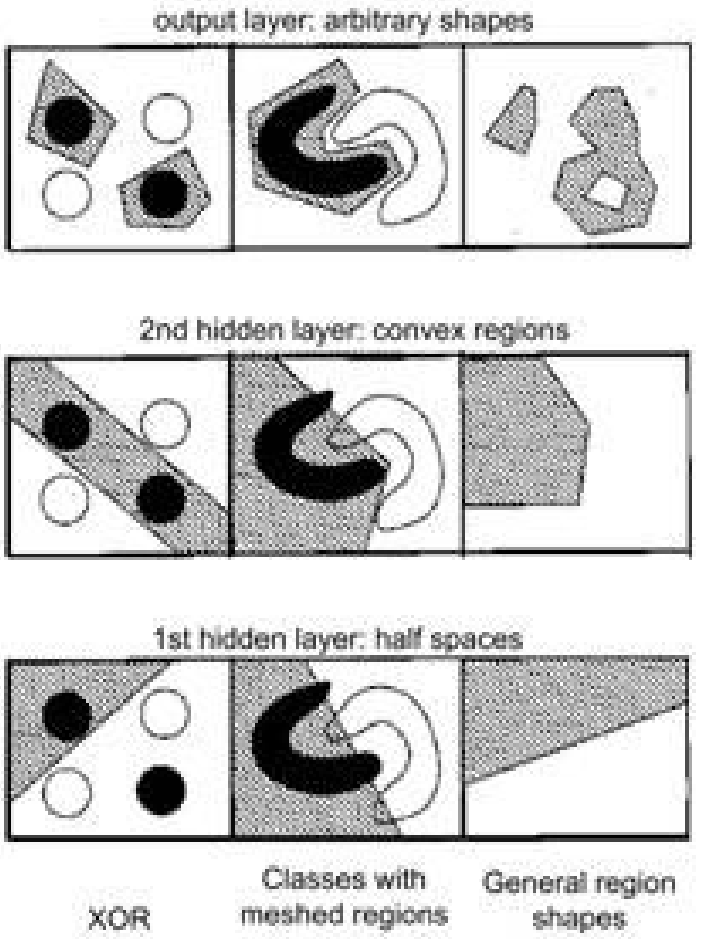






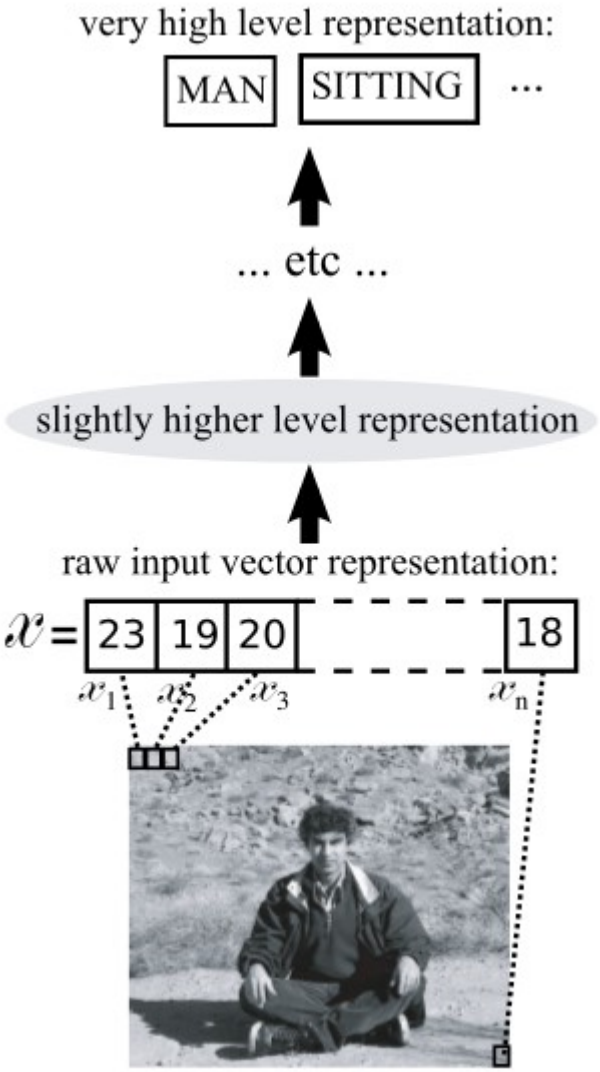
Decision regions in MLP





Deep neural networks (popular)

- multi-layer architectures (>3 layers)
- increasing abstractness
- with distributed representations emerging
- gradient-descent based learning
- currently top results in various large-data domains: vision (object recognition), language modeling, speech recognition
- different from biological models

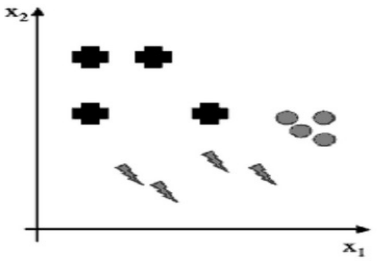


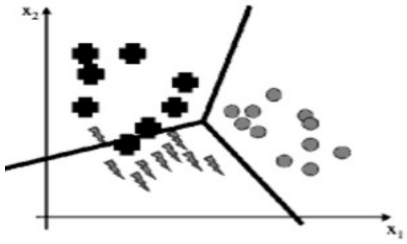
(Bengio, 2007)

Supervised learning tasks

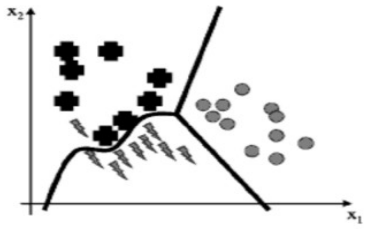
classification

x_1	x_2	Class
0.1	1	1
0.15	0.2	2
0.48	0.6	3
0.1	0.6	1
0.2	0.15	2
0.5	0.55	3
0.2	1	1
0.3	0.25	2
0.52	0.6	3
0.3	0.6	1
0.4	0.2	2
0.52	0.5	3





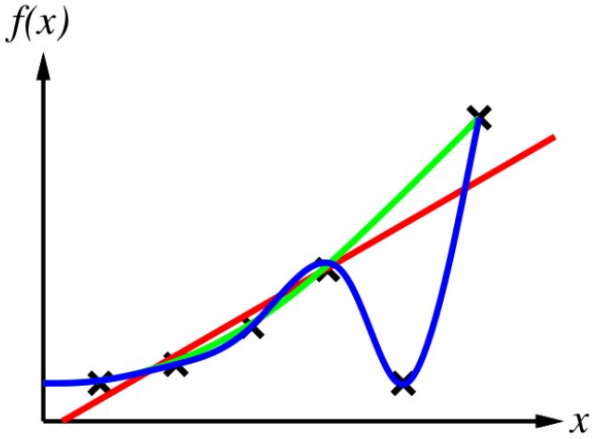
Model A



Model B

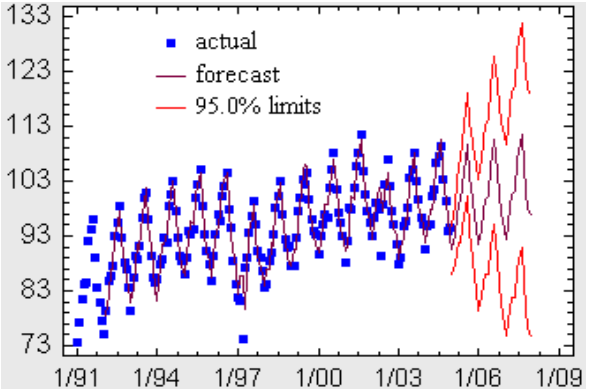
regression

3 different models shown



Predicting real-valued dependent variable, based on several (or many) independent variable(s).

time-series forecasting



Predicting future real-valued dependent variable based on its observations so far.

Summary

- Biological inspiration of artificial neural networks
- Input-output mapping = reflex agents with learning
- Data assumed to be available
- Learning with a teacher (supervised)
- Feature extraction at hidden layers
- Error minimization by gradient descent
- Generalization is crucial for using ANNs.
- Supervised tasks: classification and regression.