Introduction to Neural Networks

Marcello Pelillo

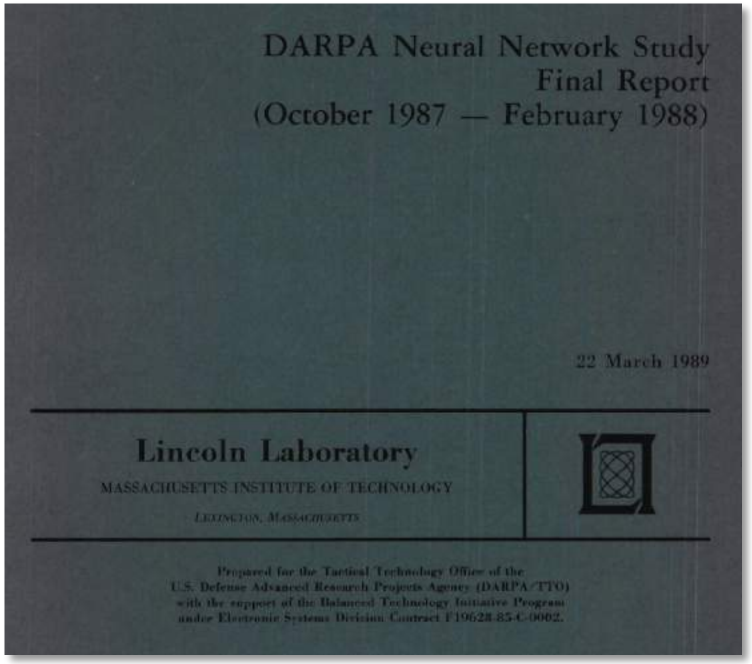
University of Venice, Italy

Artificial Intelligence

a.y. 2018/19

DARPA Neural Network Study (1989)

“Over the history of computing science, two advances have matured: High speed numerical processing and knowledge processing (Artificial Intelligence). Neural networks seem to offer the next necessary ingredient for intelligent machines – namely, knowledge formation and organization.”



DARPA Neural Network Study
Final Report
(October 1987 — February 1988)

22 March 1989

Lincoln Laboratory
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Lexington, Massachusetts

Prepared for the Tactical Technology Office of the
U.S. Defense Advanced Research Projects Agency (DARPA/TTO)
with the support of the Balanced Technology Initiative Program
under Electronic Systems Division Contract F19628-85-C-0002.

DARPA Neural Network Study (1989)

Two key features which, it is widely believed, distinguish neural networks from any other sort of computing developed thus far:

Neural networks are adaptive, or trainable. Neural networks are not so much programmed as they are trained with data – thus many believe that the use of neural networks can relieve today's computer programmers of a significant portion of their present programming load. Moreover, neural networks are said to improve with experience – the more data they are fed, the more accurate or complete their response.

Neural networks are naturally massively parallel. This suggests they should be able to make decisions at high-speed and be fault tolerant.

History

Early work (1940-1960)

• McCulloch & Pitts	(Boolean logic)
• Rosenblatt	(Learning)
• Hebb	(Learning)

Transition (1960-1980)

• Widrow – Hoff	(LMS rule)
• Anderson	(Associative memories)
• Amari	

Resurgence (1980-1990's)

• Hopfield	(Ass. mem. / Optimization)
• Rumelhart et al.	(Back-prop)
• Kohonen	(Self-organizing maps)
• Hinton , Sejnowski	(Boltzmann machine)

New resurgence (2012 -)

- CNNs, Deep learning, GAN's ….

A Few Figures

The human cerebral cortex is composed of about

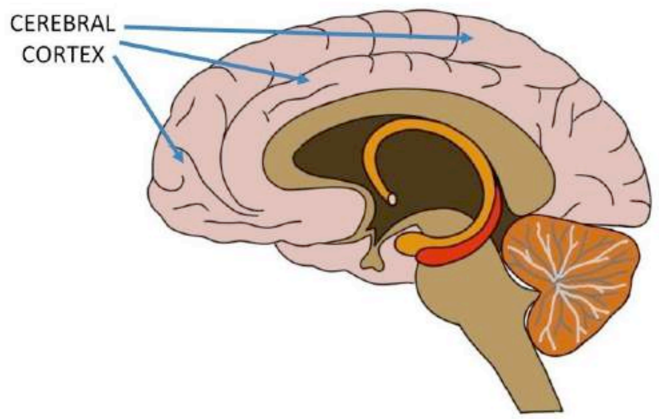
100 billion (10^{11}) neurons

of many different types.

Each neuron is connected to other 1000 / 10000 neurons, wich yields

$10^{14}/10^{15}$ connections

The cortex covers about 0.15 m²
and is 2-5 mm thick

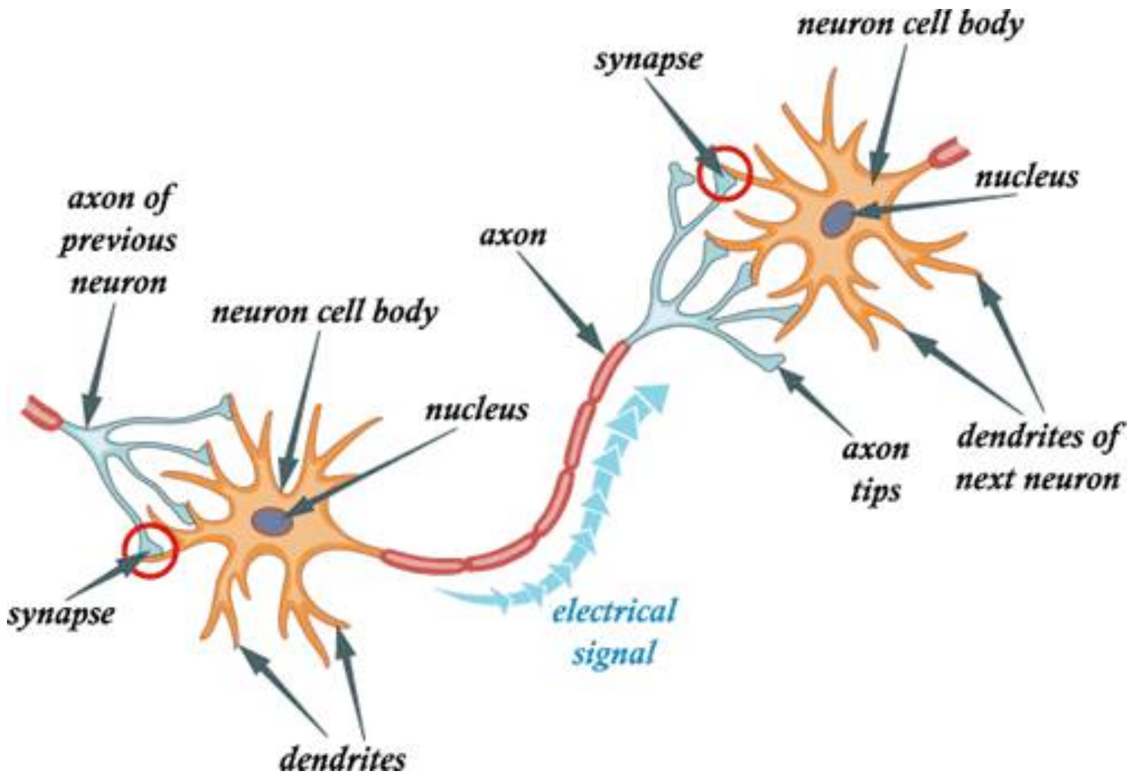


The Neuron

Cell Body (Soma): 5-10 microns in diameter

Axon: Output mechanism for a neuron; one axon/cell, but thousands of branches and cells possible for a single axon

Dendrites: Receive incoming signals from other nerve axons via synapse



Neural Dynamics

The transmission of signal in the cerebral cortex is a complex process:

electrical $\rightarrow$ chemical $\rightarrow$ electrical

Simplifying :

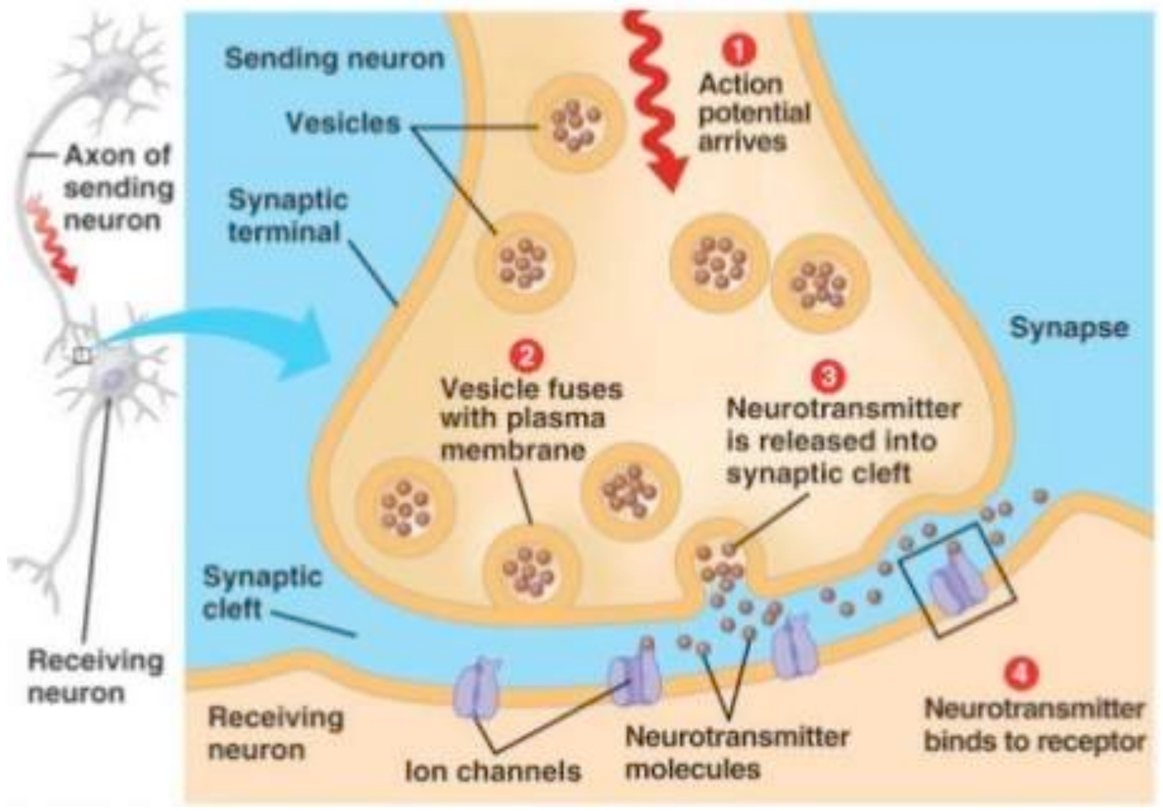
1) The cellular body performs a “weighted sum” of the incoming signals

2) If the result exceeds a certain threshold value, then it produces an “action potential” which is sent down the axon (cell has “fired”), otherwise it remains in a rest state

3) When the electrical signal reaches the synapse, it allows the “neuro-transmitter” to be released and these combine with the “receptors” in the post-synaptic membrane

4) The post-synaptic receptors provoke the diffusion of an electrical signal in the post-synaptic neuron

Synapses


Axon of sending neuron
Receiving neuron
Sending neuron
Vesicles
Synaptic terminal
1 Action potential arrives
2 Vesicle fuses with plasma membrane
3 Neurotransmitter is released into synaptic cleft
Synapse
Synaptic cleft
Receiving neuron
Ion channels
Neurotransmitter molecules
4 Neurotransmitter binds to receptor

SYNAPSE is the relay point where information is conveyed by chemical transmitters from neuron to neuron. A synapse consists of two parts: the knowblike tip of an axon terminal and the receptor region on the surface of another neuron. The membranes are separated by a synaptic cleft some 200 nanometers across. Molecules of chemical transmitter, stored in vesicles in the axon terminal, are released into the cleft by arriving nerve impulses. Transmitter changes electrical state of the receiving neuron, making it either more likely or less likely to fire an impulse.

Synaptic Efficacy

It’s the amount of current that enters into the post-synaptic neuron, compared to the action potential of the pre-synaptic neuron.

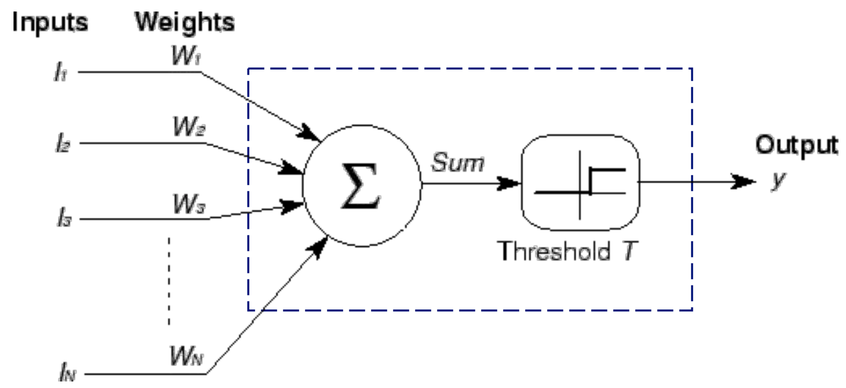
Learning takes place by modifying the synaptic efficacy.

Two types of synapses:

- **Excitatory** : favor the generation of action potential in the post-synaptic neuron
- **Inhibitory** : hinder the generation of action potential

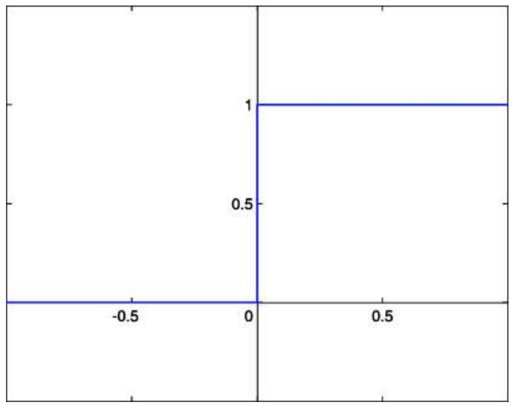
The McCulloch and Pitts Model (1943)

The McCulloch-Pitts (MP) Neuron is modeled as a binary threshold unit



The unit “fires” if the **net input** $\sum_j w_j I_j$ reaches (or exceeds) the unit’s threshold *T*:

$$y = g\left(\sum_j w_j I_j - T\right)$$



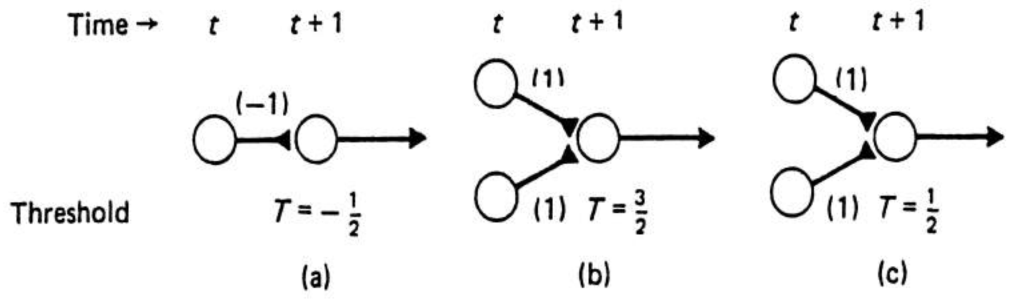
If neuron is firing, then its output *y* is 1, otherwise it is 0.

g is the unit step function: $g(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$

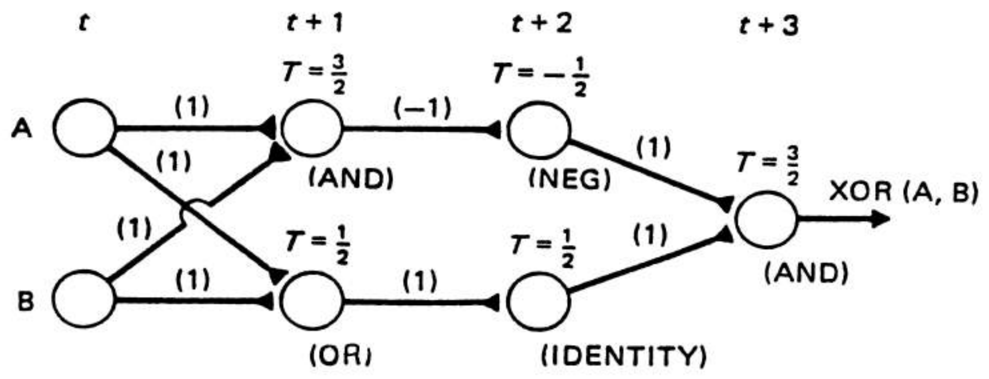
Weights w_{ij} represent the strength of the synapse between neuron *j* and neuron *i*

Properties of McCulloch-Pitts Networks

By properly combining MP neurons one can simulate the behavior of any Boolean circuit.



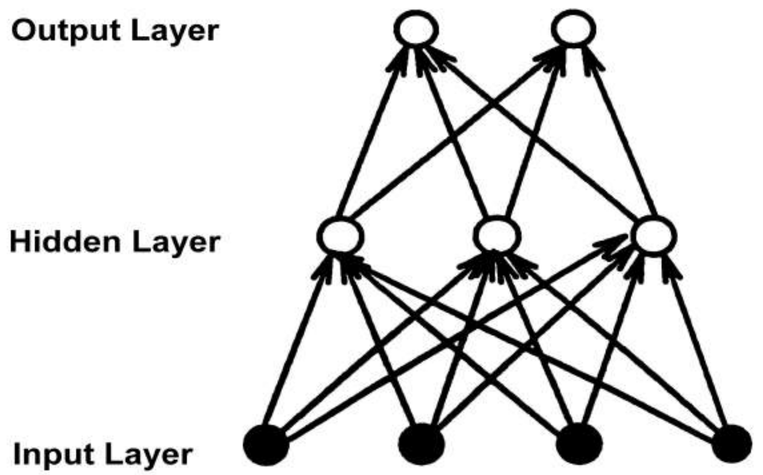
Three elementary logical operations (a) **negation**, (b) **and**, (c) **or**. In each diagram the states of the neurons on the left are at time *t* and those on the right at time *t* +1.



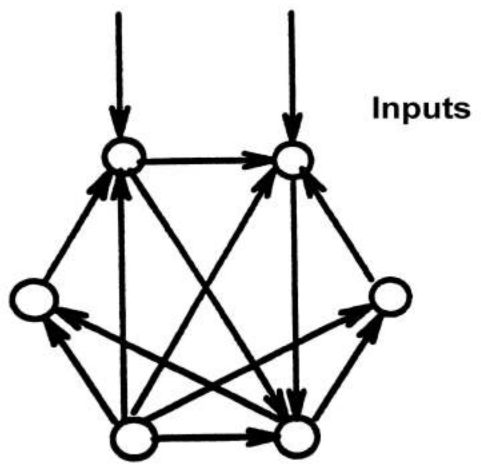
Network Topologies and Architectures

- Feedforward only *vs.* Feedback loop (Recurrent networks)
- Fully connected *vs.* sparsely connected
- Single layer *vs.* multilayer

Multilayer perceptrons, Hopfield networks,
Boltzman machines, Kohonen networks, …



(a)



(b)

Classification Problems

Given :

1) some "features": $f_1, f_2, \ldots, f_n$
2) some "classes": $c_1, \ldots, c_m$

Problem :

To classify an "object" according to its features

Example #1

To classify an “object” as :

c_1 = “ watermelon ”

c_2 = “ apple ”

c_3 = “ orange ”

According to the following features :

f_1 = “ weight ”

f_2 = “ color ”

f_3 = “ size ”

Example :

$$\left.\begin{array}{l} \text{weight} = 80\text{ g} \\ \text{color} = \text{green} \\ \text{size} = 10\text{ cm}^3 \end{array}\right\} \Longrightarrow \textbf{“apple”}$$

Example #2

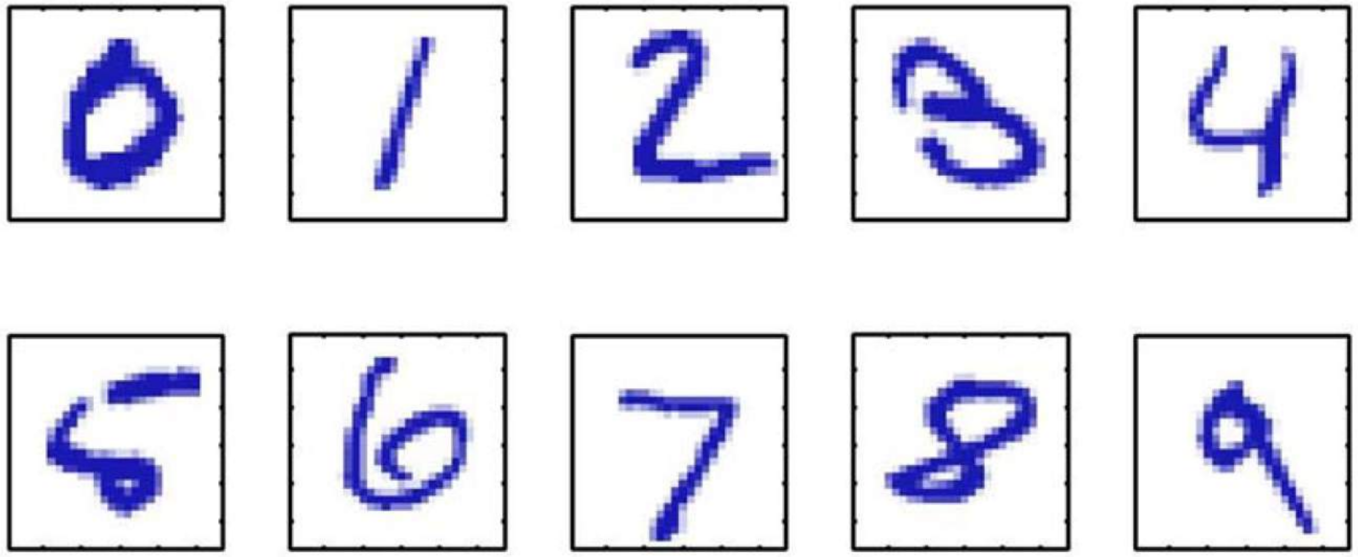
Problem: Establish whether a patient got the flu

- Classes : { " flu " , " non-flu " }

- (Potential) Features :

f_1	:	Body temperature	
f_2	:	Headache ?	(yes / no)
f_3	:	Throat is red ?	(yes / no / medium)
f_4	:		

Example #3
Hand-written digit recognition



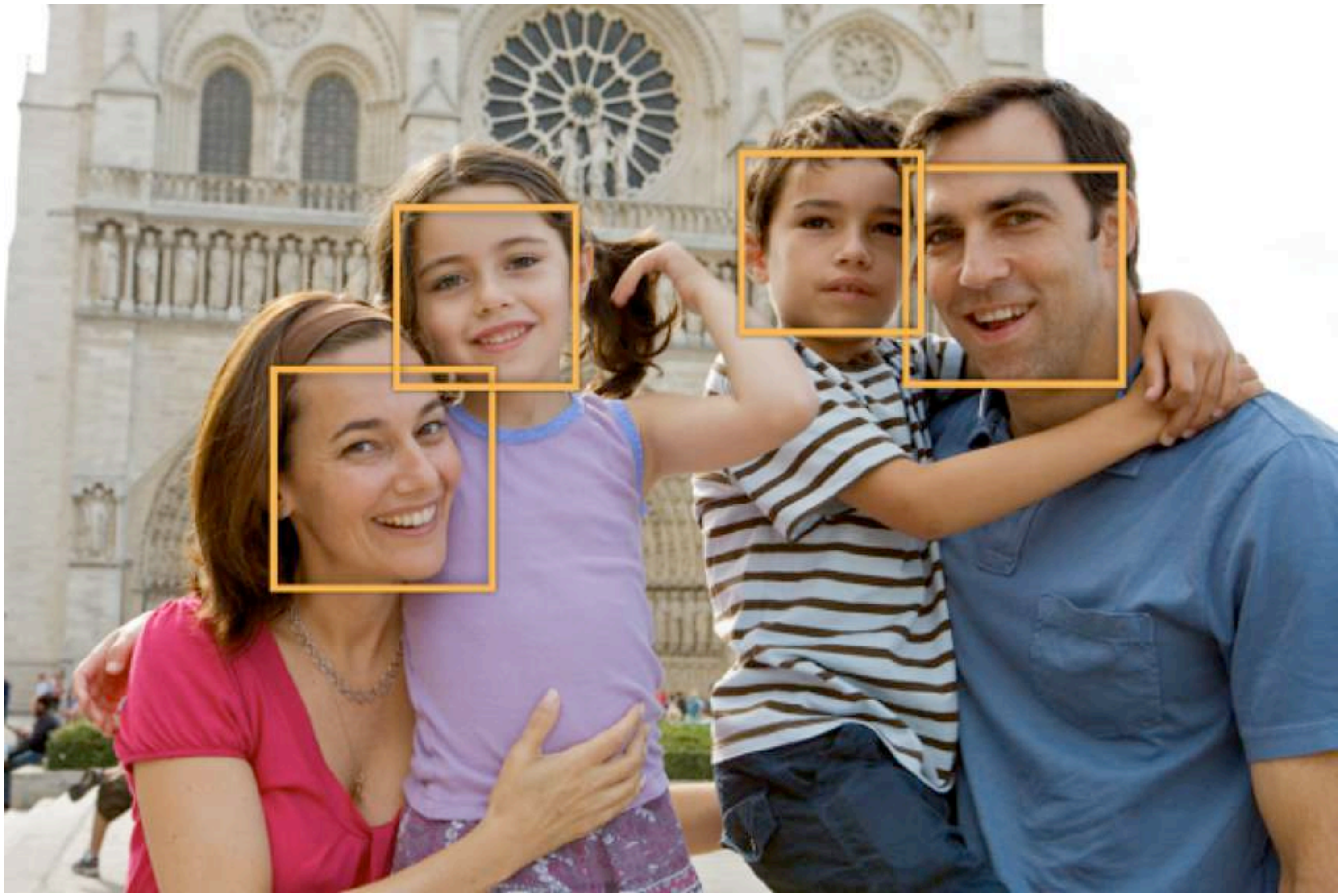
Images are 28 x 28 pixels

Represent input image as a vector $\mathbf{x} \in \mathbb{R}^{784}$

Learn a classifier $f(\mathbf{x})$ such that,

$$f : \mathbf{x} \to \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

Example #4:
Face Detection



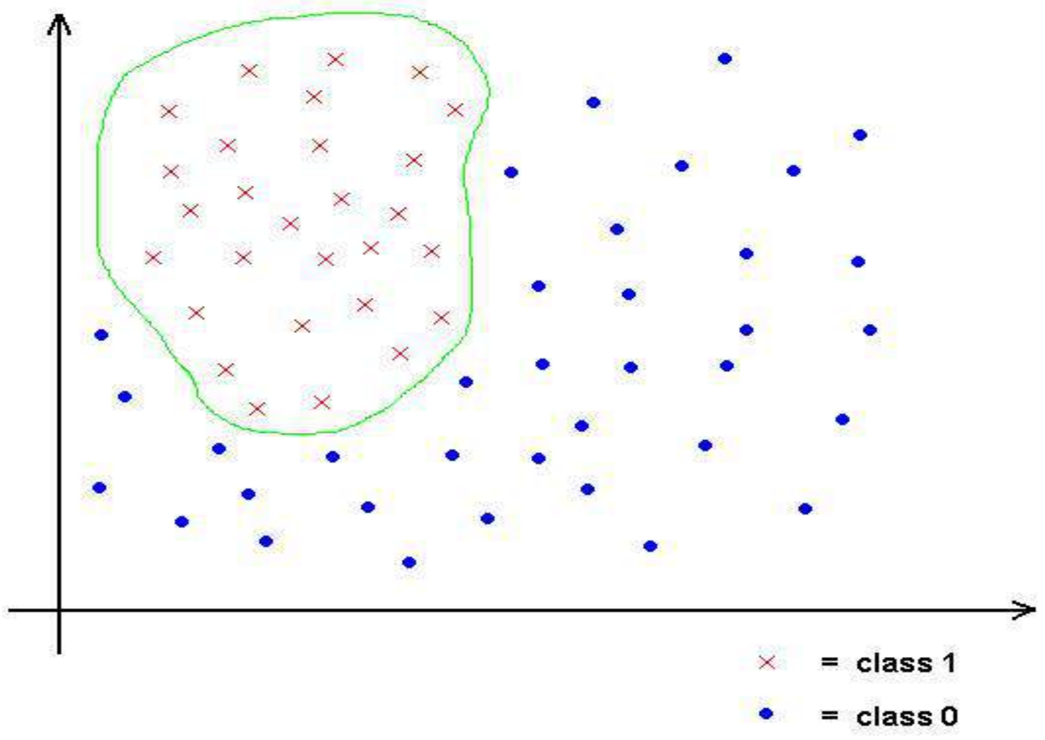
Geometric Interpretation

Example:

Classes = { 0 , 1 }

Features = x , y : both taking value in [0 , +∞ [

Idea: Objects are represented as "point" in a geometric space

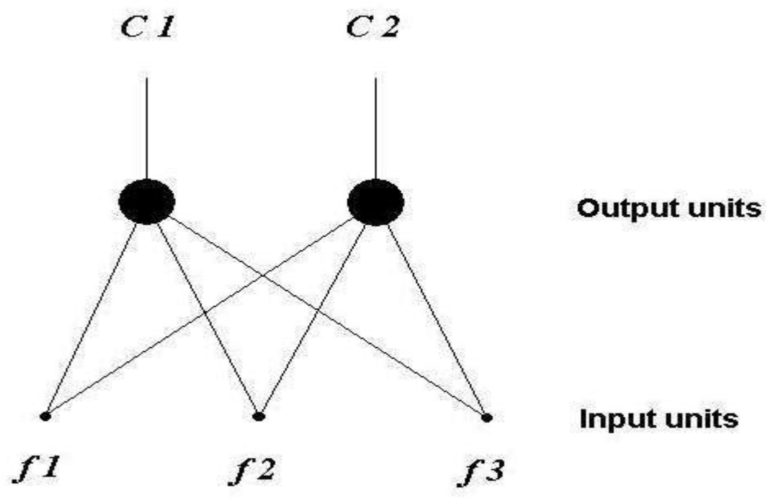


Neural Networks for Classification

A neural network can be used as a classification device .

Input	≡	features values
Output	≡	class labels

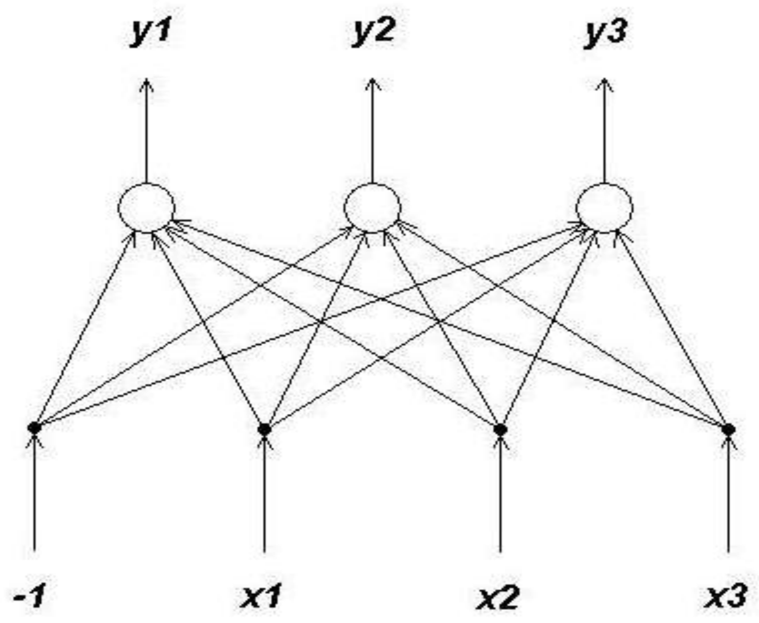
Example : 3 features , 2 classes



Thresholds

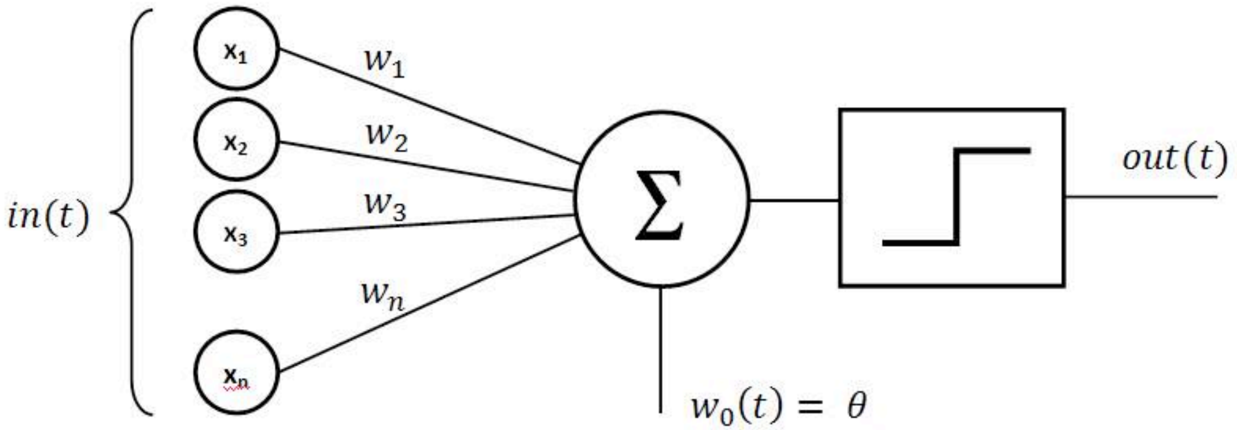
We can get rid of the thresholds associated to neurons by adding an extra unit **permanently clamped at -1** (or +1).

In so doing, thresholds become weights and can be adaptively adjusted during learning.



The Perceptron

A network consisting of one layer of M&P neurons connected in a feedforward way (i.e. no lateral or feedback connections).



- Discrete output (+1 / -1)
- Capable of “learning” from examples (Rosenblatt)
- They suffer from serious computational limitations

The Geometry of the *Perceptron*

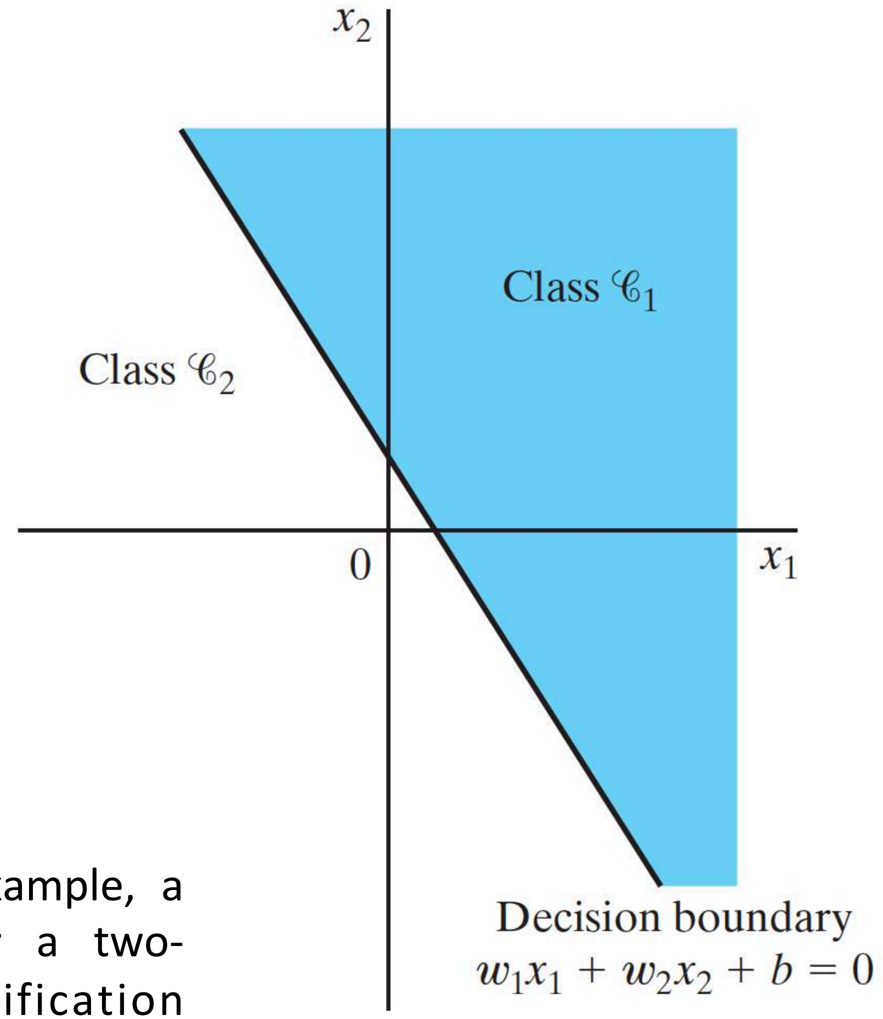


Illustration of the hyperplane (in this example, a straight line) as decision boundary for a two-dimensional, two-class pattern-classification problem.

The Perceptron Learning Algorithm

Variables and Parameters:

$$
\begin{aligned}
\mathbf{x}(n) &= (m+1)\text{-by-1 input vector} \\
&= [+1, x_1(n), x_2(n), \ldots, x_m(n)]^T \\
\mathbf{w}(n) &= (m+1)\text{-by-1 weight vector} \\
&= [b, w_1(n), w_2(n), \ldots, w_m(n)]^T \\
b &= \text{bias} \\
y(n) &= \text{actual response (quantized)} \\
d(n) &= \text{desired response} \\
\eta &= \text{learning-rate parameter, a positive constant less than unity}
\end{aligned}
$$

1. *Initialization.* Set $\mathbf{w}(0) = \mathbf{0}$. Then perform the following computations for time-step $n = 1, 2, \ldots$.
2. *Activation.* At time-step n, activate the perceptron by applying continuous-valued input vector $\mathbf{x}(n)$ and desired response $d(n)$.
3. *Computation of Actual Response.* Compute the actual response of the perceptron as

$$y(n) = \operatorname{sgn}[\mathbf{w}^T(n)\mathbf{x}(n)]$$

   where $\operatorname{sgn}(\cdot)$ is the signum function.
4. *Adaptation of Weight Vector.* Update the weight vector of the perceptron to obtain

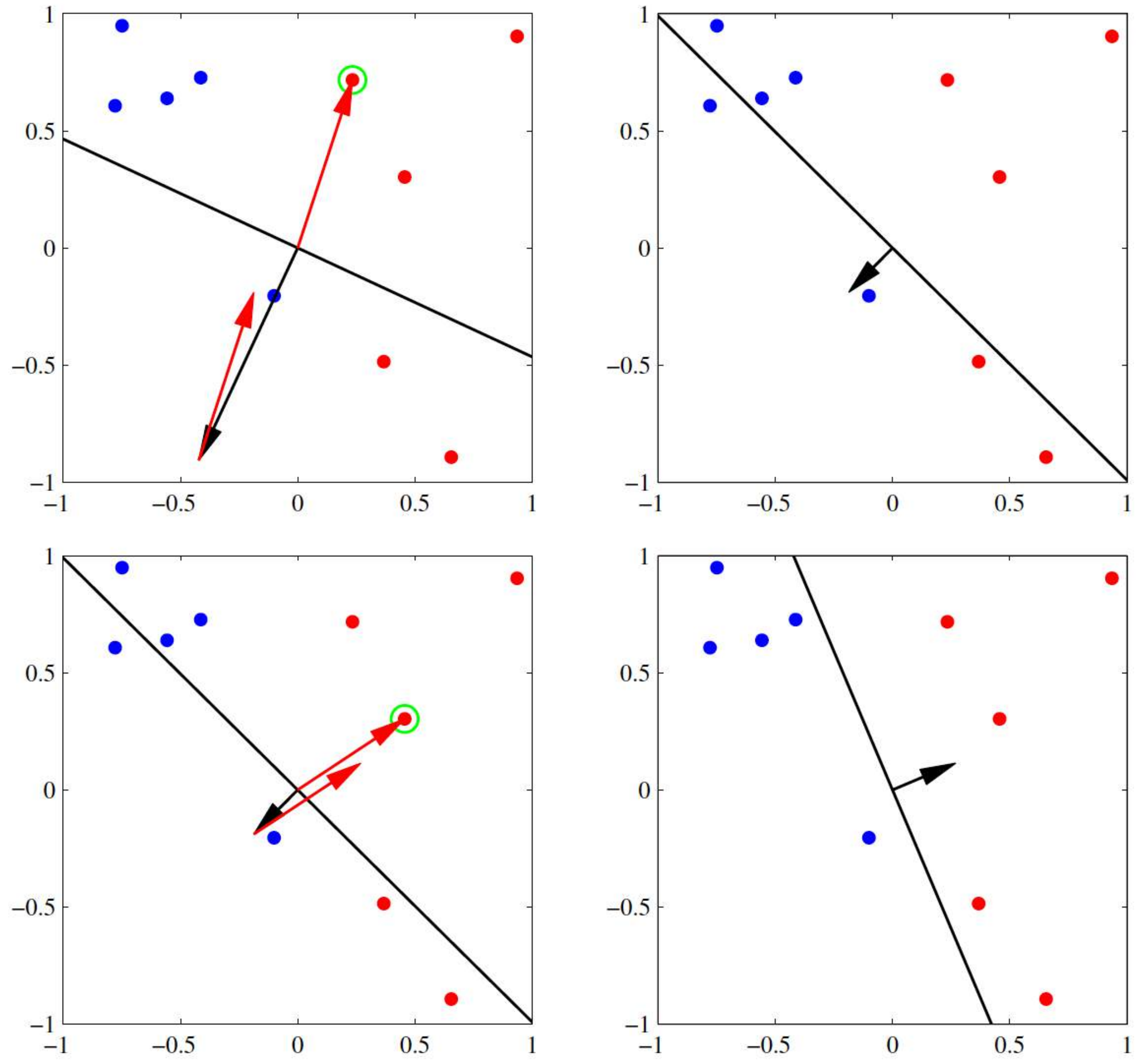
$$\mathbf{w}(n+1) = \mathbf{w}(n) + \eta[d(n) - y(n)]\mathbf{x}(n)$$

   where

$$d(n) = \begin{cases} +1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_1 \\ -1 & \text{if } \mathbf{x}(n) \text{ belongs to class } \mathscr{C}_2 \end{cases}$$

5. *Continuation.* Increment time step n by one and go back to step 2.

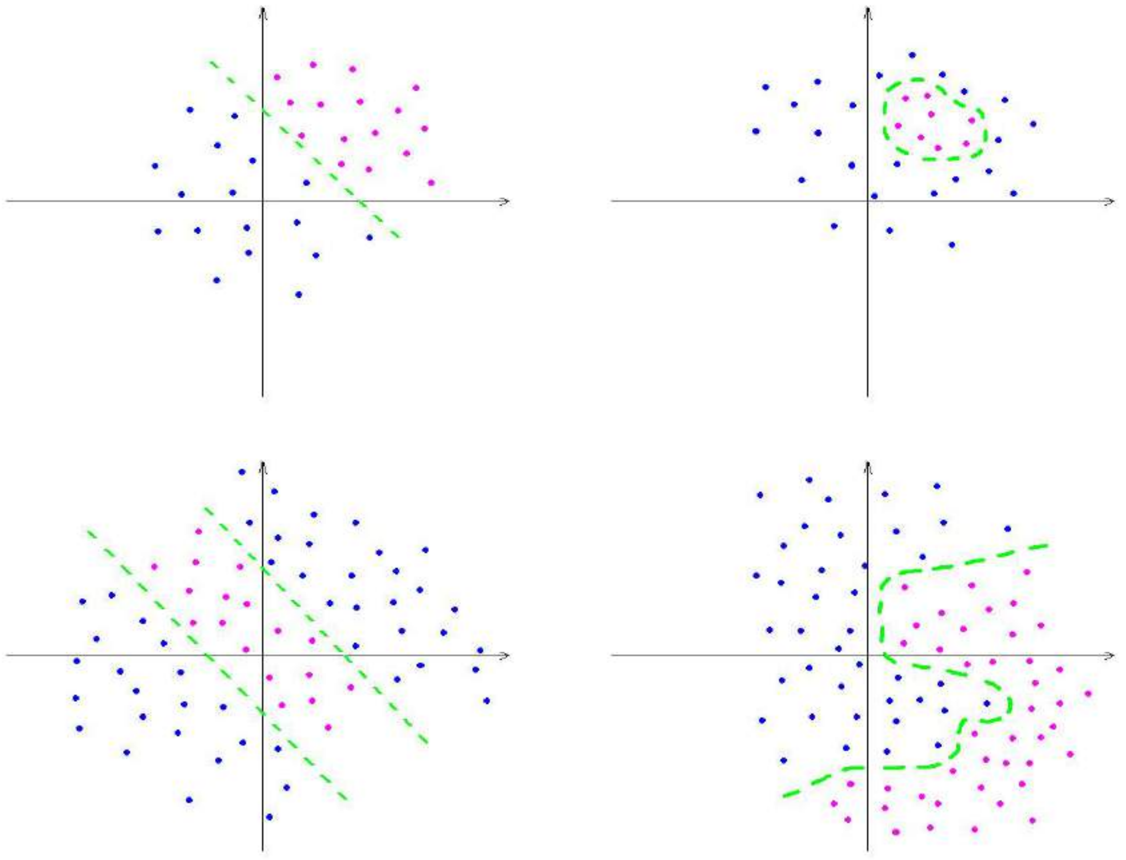
The Perceptron Learning Algorithm



Decision Regions

It’s an area wherein all examples of one class fall.

Examples:

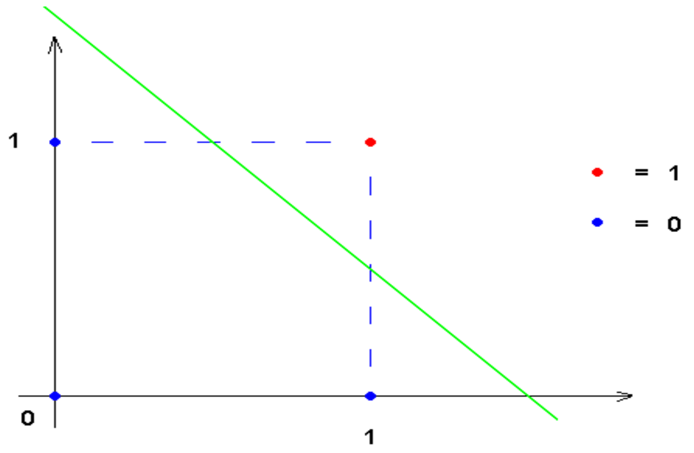


Linear Separability

A classification problem is said to be **linearly separable** if the decision regions can be separated by a hyperplane.

Example: AND

X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

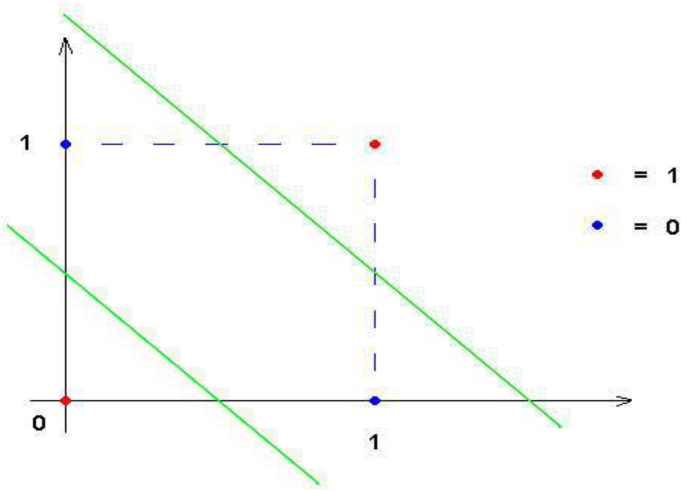


Limitations of Perceptrons

It has been shown that perceptrons can only solve linearly separable problems.

Example: XOR (exclusive OR)

X	Y	X XOR Y
0	0	0
0	1	1
1	0	1
1	1	0



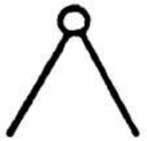
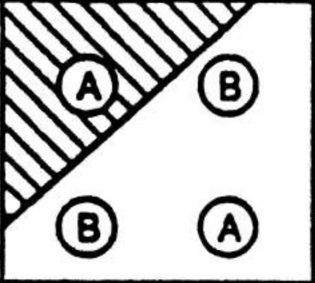
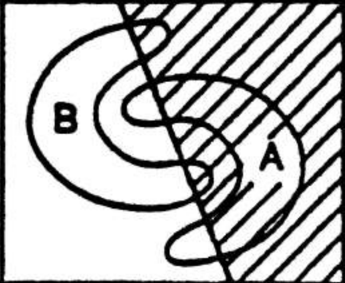
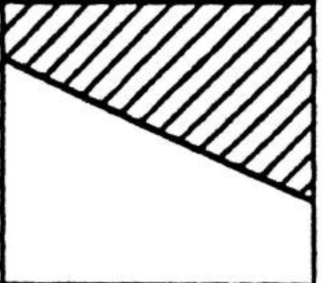
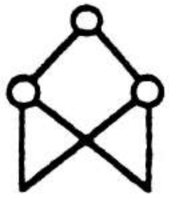
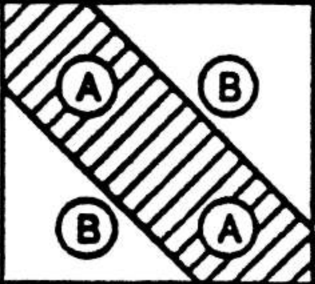
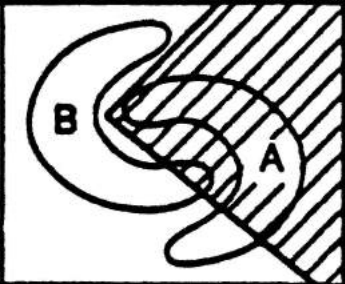
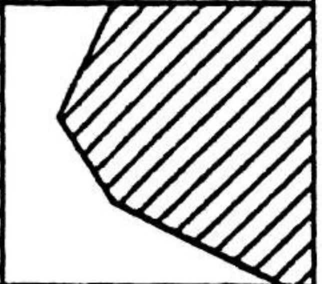
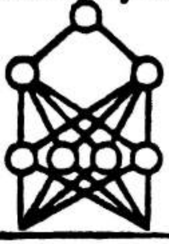
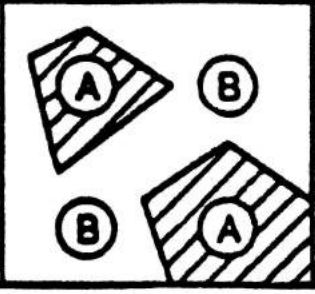
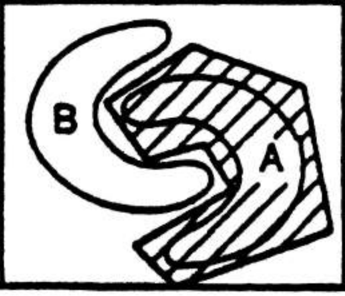
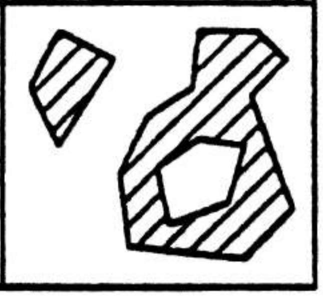
The Perceptron Convergence Theorem

Theorem (Rosenblatt, 1960)

If the training set is **linearly separable**, the perceptron learning algorithm **always converges** to a consistent hypothesis after a **finite** number of epochs, for any $\eta > 0$.

If the training set is **not** linearly separable, after a certain number of epochs the weights start oscillating.

A View of the Role of Units

Structure	Type of Decision Regions	Exclusive-OR Problem	Classes with Meshed Regions	Most General Region Shapes
Single-layer 	Half plane bounded by hyperplane			
Two-layers 	Convex open or closed regions			
Three-layers 	Arbitrary (Complexity limited by number of nodes)			

Multi–Layer Feedforward Networks

- Limitation of simple perceptron: can implement only linearly separable functions

- Add “ hidden” layers between the input and output layer. A network with just one hidden layer can represent any Boolean functions including XOR

- Power of multilayer networks was known long ago, but algorithms for training or learning, e.g. back-propagation method, became available only recently (invented several times, popularized in 1986)

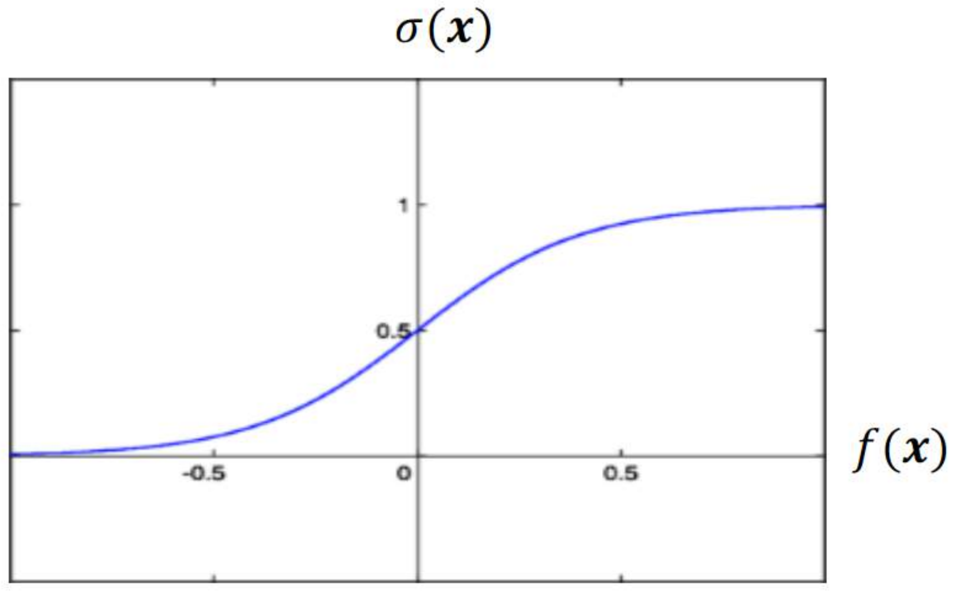
- **Universal approximation power:** Two-layer network can approximate any smooth function (Cybenko, 1989; Funahashi, 1989; Hornik, et al.., 1989)

- Static (no feedback)

Continuous-Valued Units

Sigmoid (or logistic)

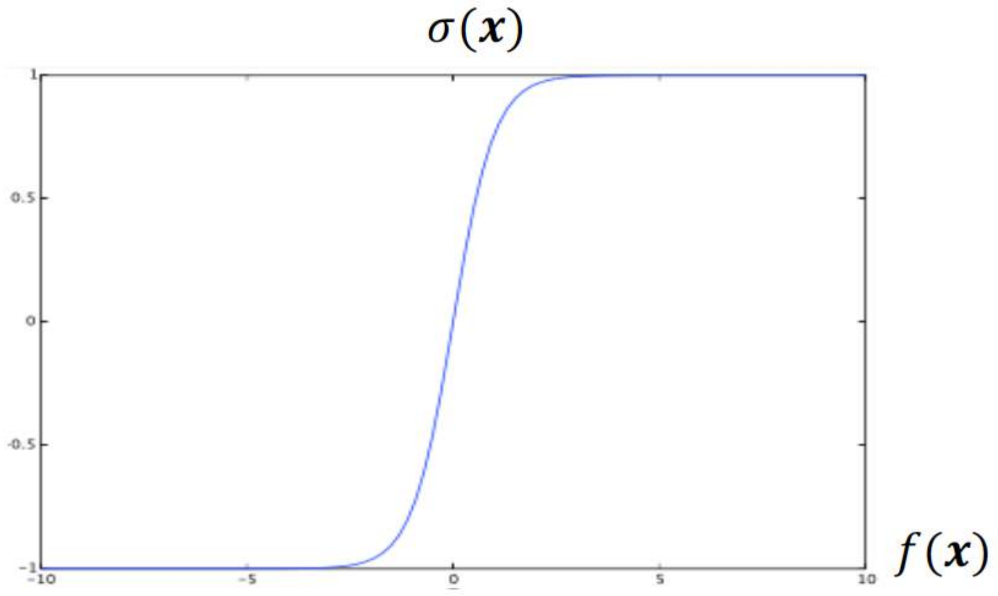
$$\sigma(\boldsymbol{x}) = \frac{1}{1 + e^{-f(\boldsymbol{x})}} \in (0,1)$$



Continuous-Valued Units

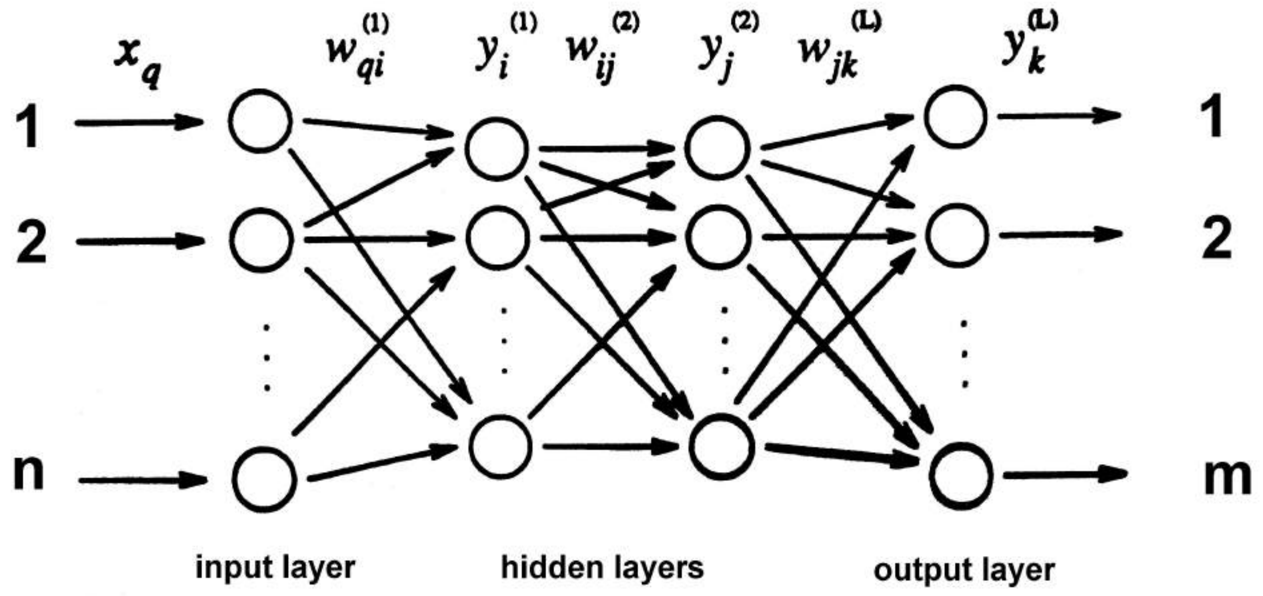
Hyperbolic tangent

$$\sigma(\boldsymbol{x}) = \frac{e^{f(\boldsymbol{x})} - e^{-f(\boldsymbol{x})}}{e^{f(\boldsymbol{x})} + e^{-f(\boldsymbol{x})}} \in (-1,1)$$



Back-propagation Learning Algorithm

- An algorithm for learning the weights in a feed-forward network, given a training set of input-output pairs
- The algorithm is based on gradient descent method.



Supervised Learning

Supervised learning algorithms require the presence of a “teacher” who provides the right answers to the input questions.

Technically, this means that we need a **training set** of the form

$$L = \left\{ \left(\mathrm{x}^1, \mathrm{y}^1\right), \quad ..... \quad \left(\mathrm{x}^p, \mathrm{y}^p\right) \right\}$$

where :

$\mathrm{x}^\mu \quad \left(\mu = 1 \ldots p \right)$ is the network input vector

$\mathrm{y}^\mu \quad \left(\mu = 1 \ldots p \right)$ is the **desired** network output vector

Supervised Learning

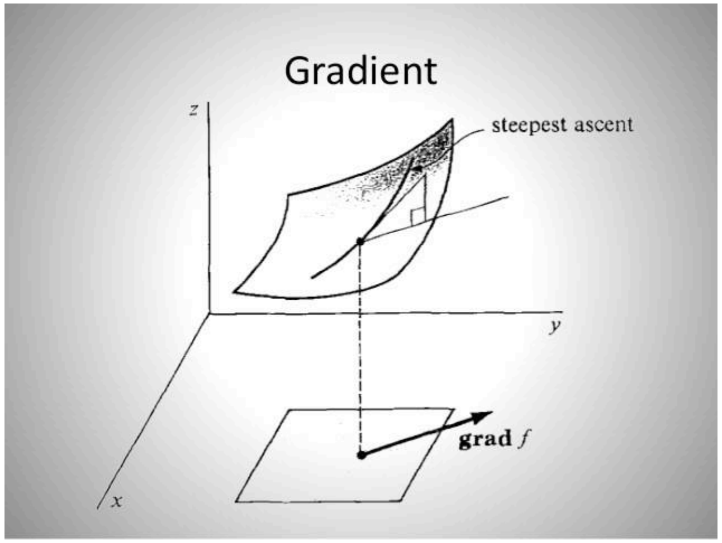
The learning (or training) phase consists of determining a configuration of weights in such a way that the network output be as close as possible to the desired output, for all the examples in the training set.

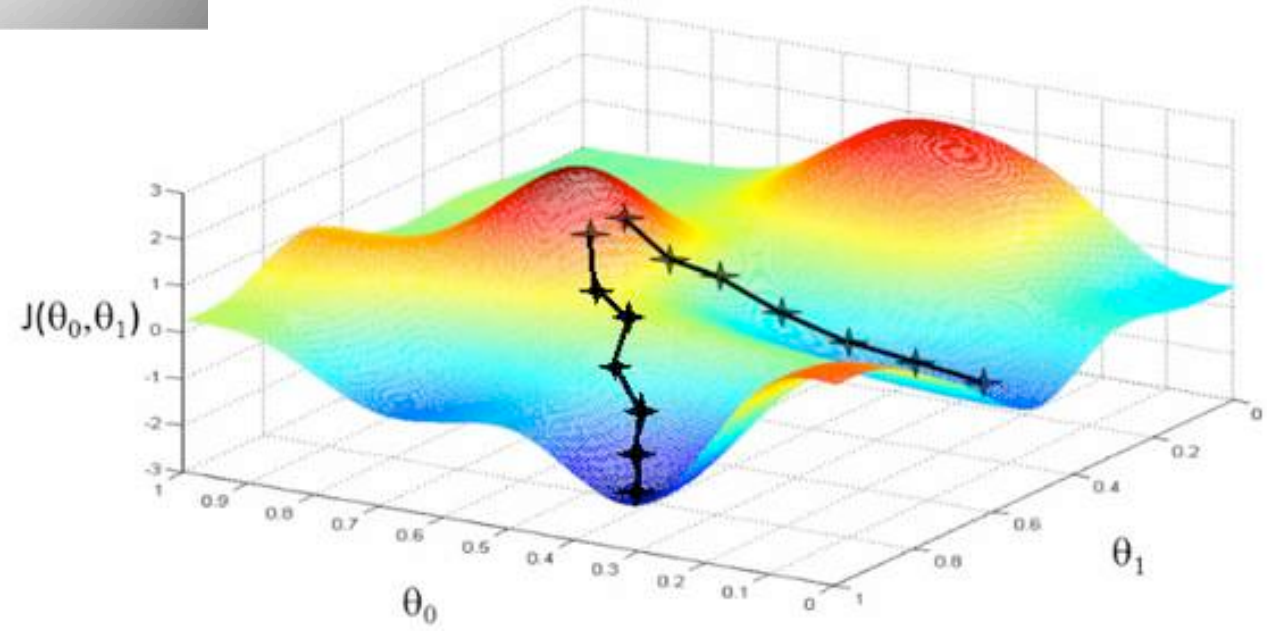
Formally, this amounts to minimizing an **error function** such as (not only possible one):

$$E = \frac{1}{2} \sum_{\mu} \sum_{k} \left(y_k^{\mu} - O_k^{\mu} \right)^2$$

where O_k^{μ} is the output provided by the output unit *k* when the network is given example μ as input.

Optimization by Gradient Descent





Back-Propagation

To minimize the error function *E* we can use the classic **gradient-descent** algorithm:

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial E_k}{\partial w_{ji}}$$

η = "learning rate"

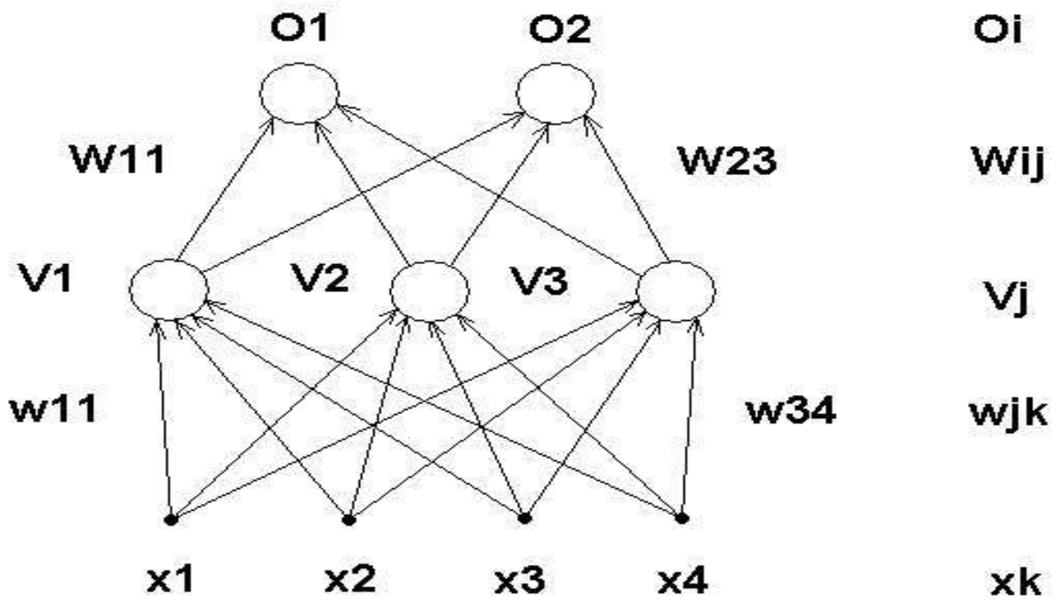
To compute the partial derivates we use the **error back propagation** algorithm.

It consists of two stages:

Forward pass : the input to the network is propagated layer after layer in forward direction

Backward pass : the "error" made by the network is propagated backward, and weights are updated properly

Notations



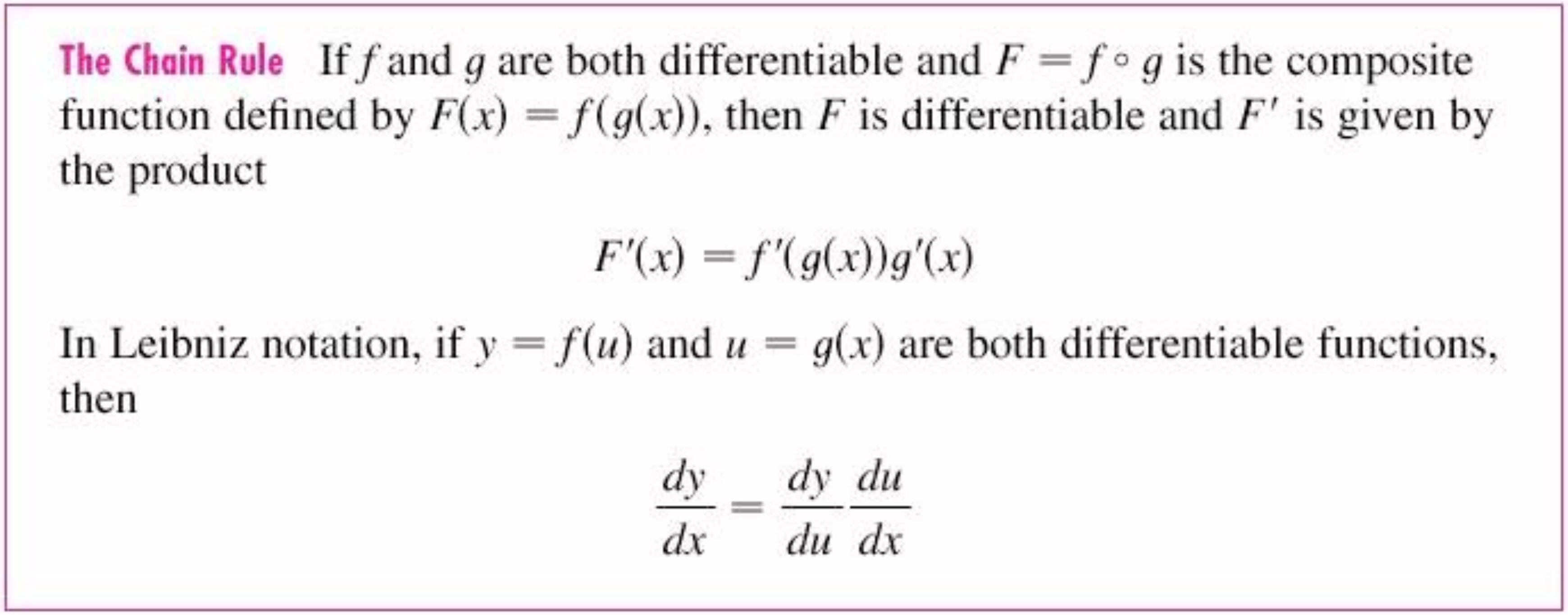
Given pattern *μ*, hidden unit *j* receives a net input

$$h_j^\mu = \sum_k w_{jk}\, x_k^\mu$$

and produces as output :

$$V_j^\mu = g\left(h_j^\mu\right) = g\left(\sum_k w_{jk}\, x_k^\mu\right)$$

Calculus Refresher



The Chain Rule If f and g are both differentiable and $F = f \circ g$ is the composite function defined by $F(x) = f(g(x))$, then F is differentiable and F' is given by the product

$$F'(x) = f'(g(x))g'(x)$$

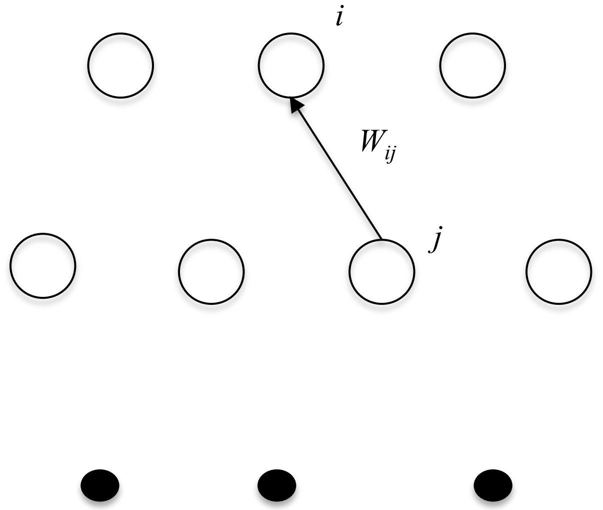
In Leibniz notation, if $y = f(u)$ and $u = g(x)$ are both differentiable functions, then

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

Back-Prop:
Updating Hidden-to-Output Weights

$$E = \frac{1}{2}\sum_{\mu}\sum_{k}\left(y_k^\mu - O_k^\mu\right)^2$$

$$\begin{aligned}
\Delta W_{ij} &= -\eta\,\frac{\partial E}{\partial W_{ij}} \\
&= -\eta\,\frac{\partial}{\partial W_{ij}}\left[\frac{1}{2}\sum_{\mu}\sum_{k}\left(y_k^\mu - O_k^\mu\right)^2\right] \\
&= \eta\sum_{\mu}\sum_{k}\left(y_k^\mu - O_k^\mu\right)\frac{\partial O_k^\mu}{\partial W_{ij}} \\
&= \eta\sum_{\mu}\left(y_i^\mu - O_i^\mu\right)\frac{\partial O_i^\mu}{\partial W_{ij}} \\
&= \eta\sum_{\mu}\left(y_i^\mu - O_i^\mu\right)g'\left(h_i^\mu\right)V_j^\mu \\
&= \eta\sum_{\mu}\delta_i^\mu V_j^\mu
\end{aligned}$$



where : $\delta_i^\mu = \left(y_i^\mu - O_i^\mu\right)g'\left(h_i^\mu\right)$

Back-Prop: Updating Input-to-Hidden Weights (1)

$$\Delta w_{jk} = -\eta \frac{\partial E}{\partial w_{jk}}$$

$$= \eta \sum_{\mu} \sum_{i} \left(y_i^{\mu} - O_i^{\mu}\right) \frac{\partial O_i^{\mu}}{\partial w_{jk}}$$

$$= \eta \sum_{\mu} \sum_{i} \left(y_i^{\mu} - O_i^{\mu}\right) g'\left(h_i^{\mu}\right) \frac{\partial h_i^{\mu}}{\partial w_{jk}}$$

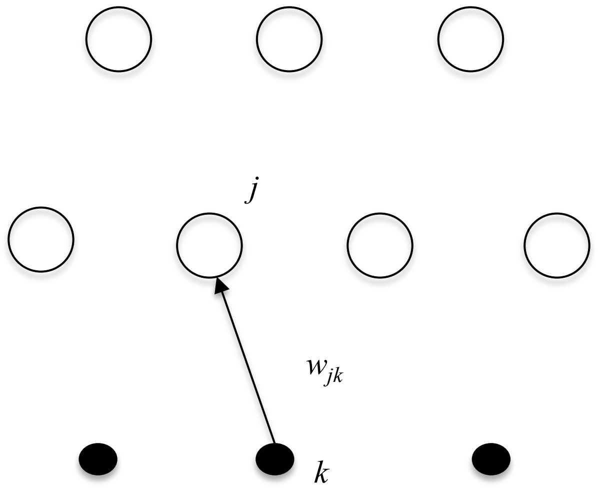
$$E = \frac{1}{2} \sum_{\mu} \sum_{i} \left(y_i^{\mu} - O_i^{\mu} \right)^2$$

$$\frac{\partial h_i^{\mu}}{\partial w_{jk}} = \sum_{l} W_{il} \frac{\partial V_l^{\mu}}{\partial w_{jk}}$$

$$= W_{ij} \frac{\partial V_j^{\mu}}{\partial w_{jk}}$$

$$= W_{ij} \frac{\partial g\left(h_j^{\mu}\right)}{\partial w_{jk}}$$

$$= W_{ij}\, g'\left(h_j^{\mu}\right) \frac{\partial h_j^{\mu}}{\partial w_{jk}}$$



Back-Prop: Updating Input-to-Hidden Weights (2)

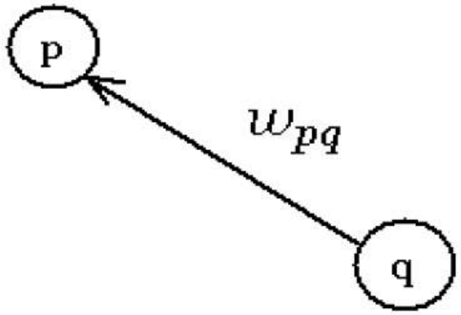
$$\frac{\partial h_j^\mu}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \sum_m w_{jm}\, x_m^\mu$$
$$= x_k^\mu$$

Hence, we get:

$$\Delta w_{jk} = \eta \sum_{\mu,i} \left(y_i^\mu - O_i^\mu\right) g'\left(h_i^\mu\right) W_{ij}\; g'\left(h_j^\mu\right) x_k^\mu$$
$$= \eta \sum_{\mu,i} \delta_i^\mu\, W_{ij}\; g'\left(h_j^\mu\right) x_k^\mu$$
$$= \eta \sum_{\mu} \hat{\delta}_j^\mu\; x_k^\mu$$

$$\text{where}: \quad \hat{\delta}_j^\mu = g'\left(h_j^\mu\right) \sum_i \delta_i^\mu\, W_{ij}$$

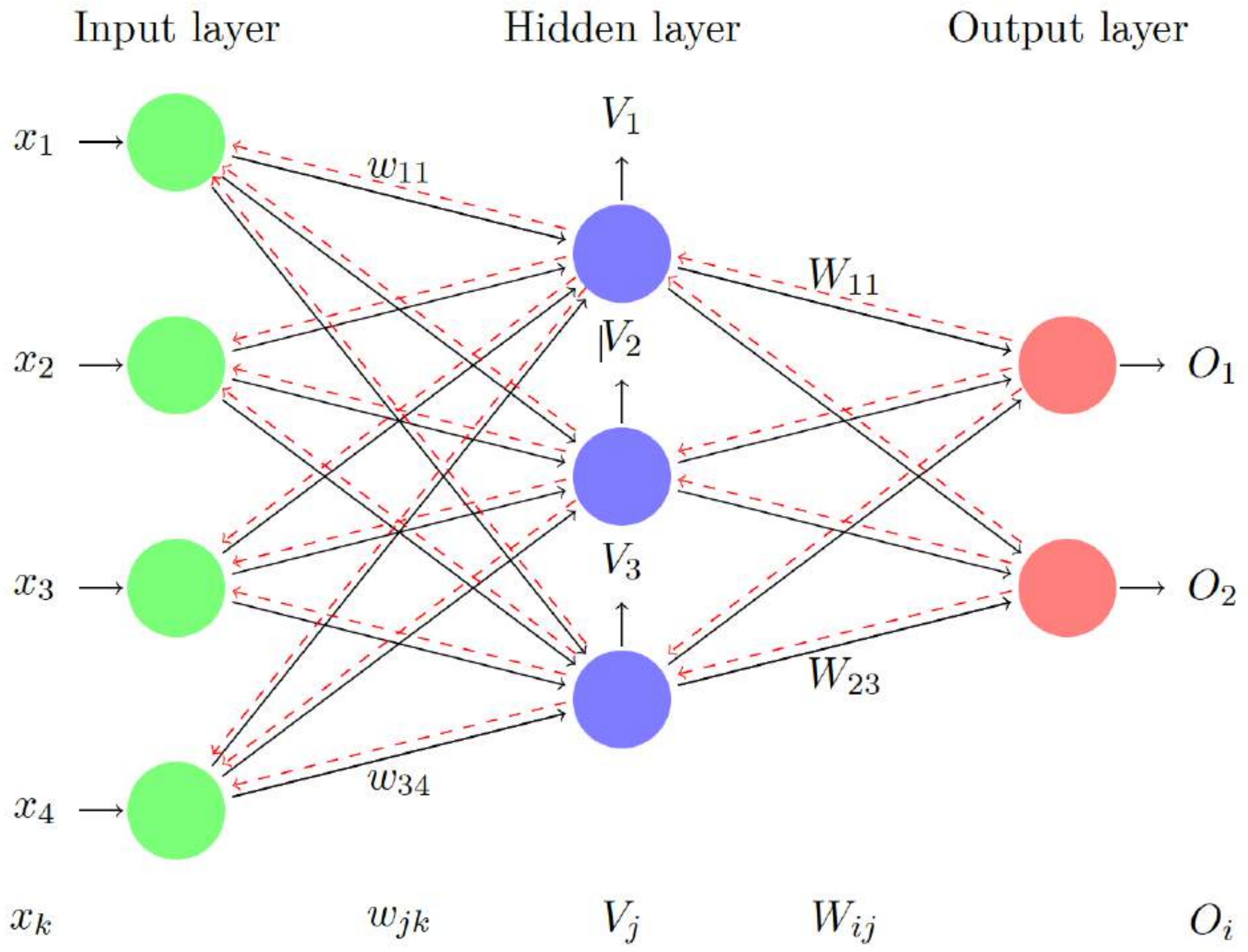
Locality of Back-Prop



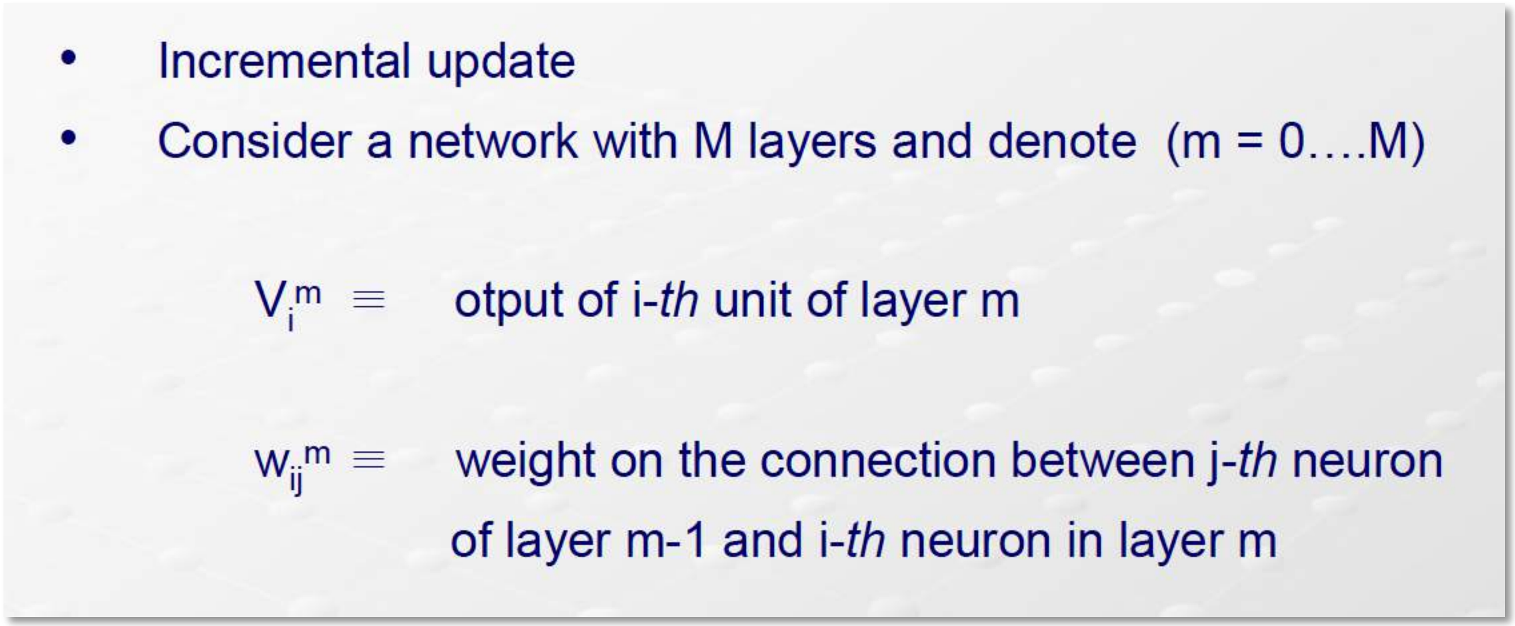
$$\Delta\omega_{pq} = \eta \sum_{\mu} \delta_p^{\mu} V_q^{\mu}$$ off - line

$$\Delta\omega_{pq} = \eta\, \delta_p^{\mu} V_q^{\mu}$$ on - line

Error Back-Propagation



The Back-Propagation Algorithm



- Incremental update
- Consider a network with M layers and denote (m = 0….M)

$V_i^m \equiv$ otput of i-*th* unit of layer m

$w_{ij}^m \equiv$ weight on the connection between j-*th* neuron of layer m-1 and i-*th* neuron in layer m

The Back-Propagation Algorithm

1. Initialize the weight to (small) random values
2. Choose a pattern $\overline{x}^{\mu}$ and apply it to the input layer (m=0)

$$V_k^0 = x_k^{\mu} \qquad \forall k$$

3. Propagate the signal forward:

$$V_i^m = g\left(h_i^m\right) = g\left(\sum_j w_{ij} V_j^{m-1}\right)$$

4. Compute the δ's for the output layer:

$$\delta_i^M = g'\left(h_i^M\right)\left(y_i^M - V_i^M\right)$$

5. Compute the δ's for all preceding layers:

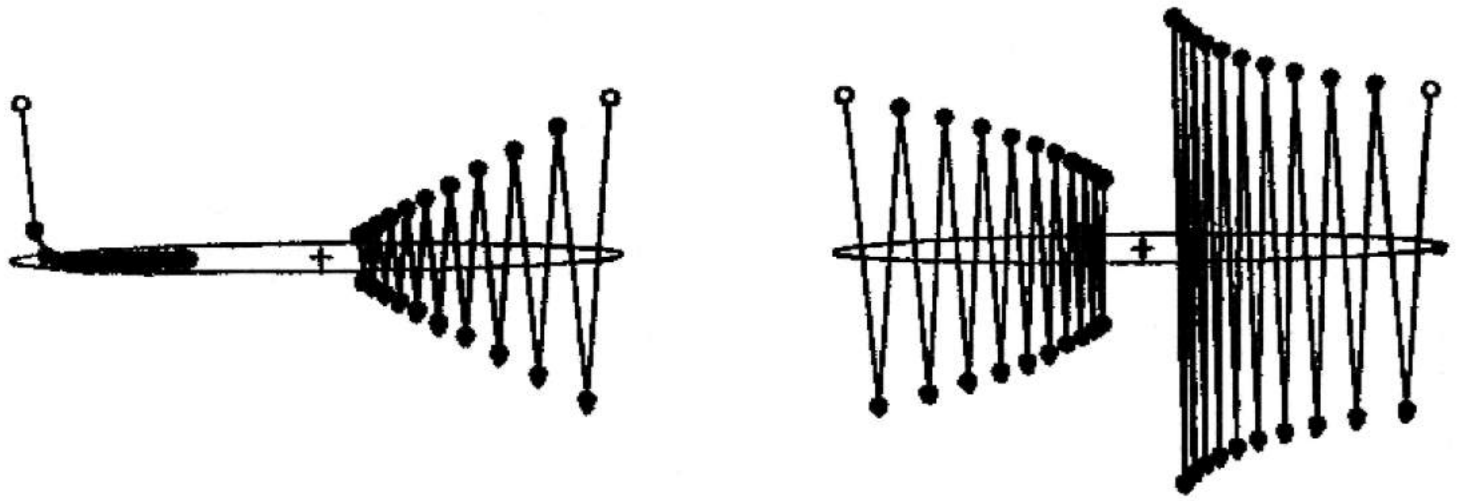
$$\delta_i^{m-1} = g'\left(h_i^{m-1}\right)\sum_j w_{ji}^m \delta_j^m$$

6. Update connection weights:

$$w_{ij}^{NEW} = w_{ij}^{OLD} + \Delta w_{ij} \qquad where \qquad \Delta w_{ij} = \eta\, \delta_i^m V_j^{m-1}$$

7. Go back to step 2 until convergence

The Role of the Learning Rate



Gradient descent on a simple quadratic surface (the left and right parts are copies of the same surface). Four trajectories are shown, each for 20 steps from the open circle. The minimum is at the + and the ellipse shows a constant error contour. The only significant difference between the trajectories is the value of η, which was 0.02, 0.0476, 0.049, and 0.0505 from left to right.

The Momentum Term

Gradient descent may:

- Converge too slowly if η is too small
- Oscillate if η is too large

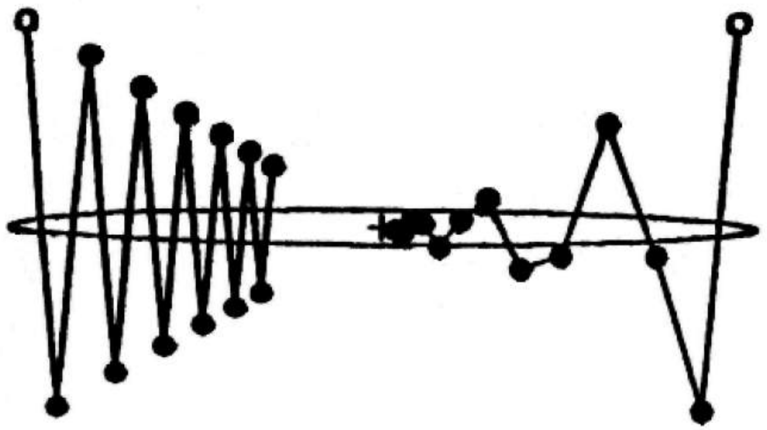
Simple remedy:

$$\Delta\omega_{pq}(t+1) = -\eta \frac{\partial E}{\partial w_{pq}} + \underbrace{\alpha\,\Delta w_{pq}(t)}_{momentum}$$

The momentum term allows us to use large values of η thereby avoiding oscillatory phenomena

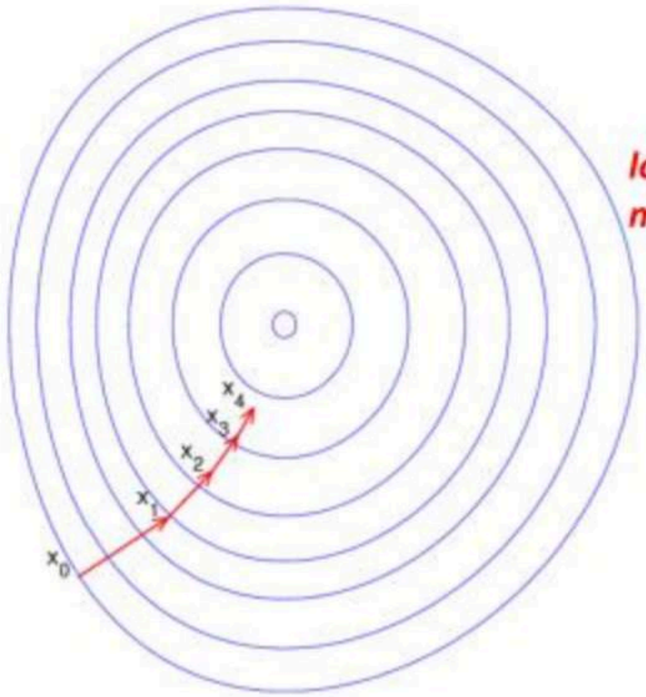
Typical choice: $\alpha = 0.9$, $\eta = 0.5$

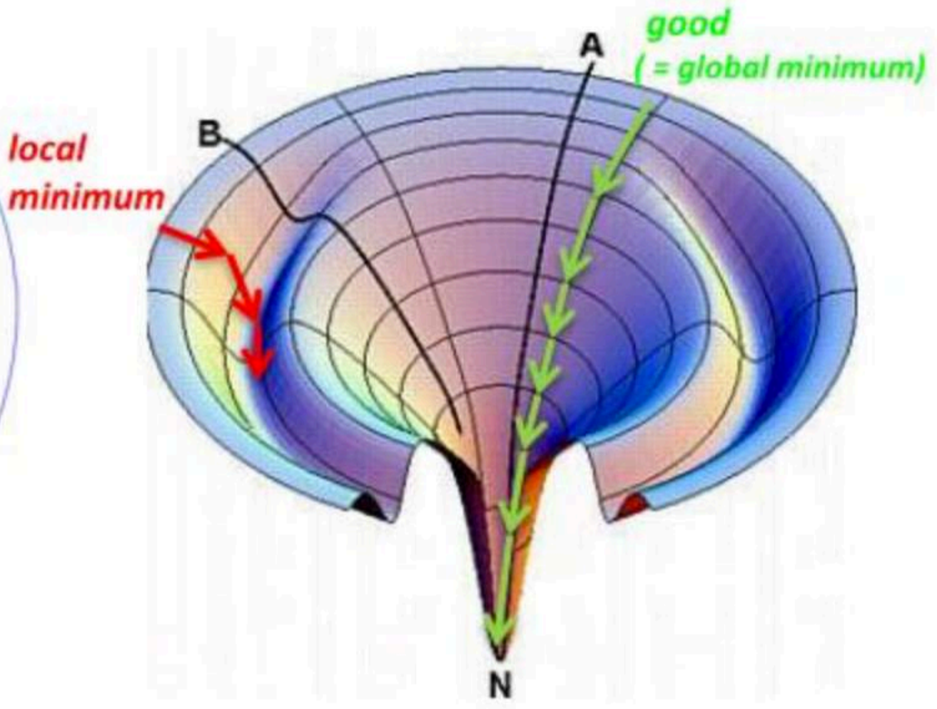
The Momentum Term



Gradient descent on the simple quadratic surface. Both trajectories are for 12 steps with $\eta = 0.0476$, the best value in the absence of momentum. On the left there is no momentum ($\alpha = 0$), while $\alpha = 0.5$ on the right.

The Problem of Local Minima



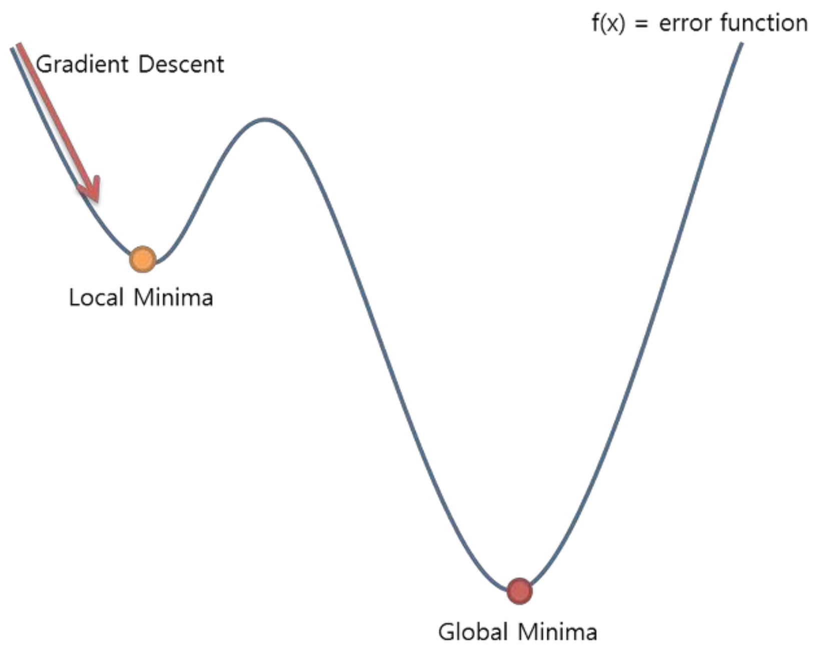


The Problem of Local Minima

Back-prop **cannot avoid local minima**.

Choice of initial weights is important.

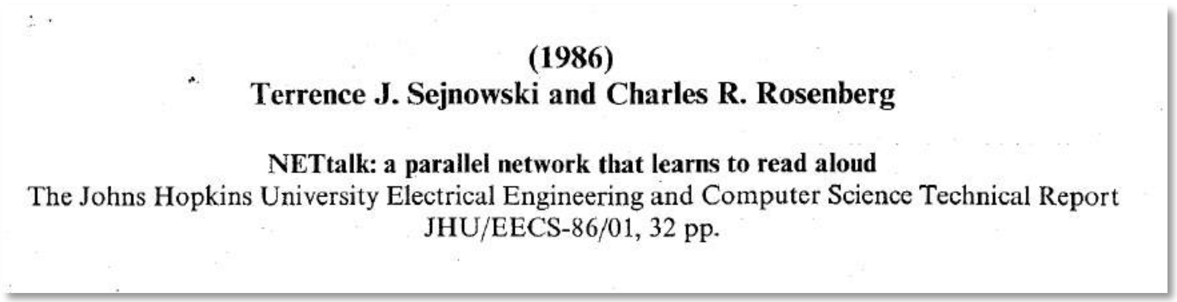
If they are too large the nonlinearities tend to saturate since the beginning of the learning process.



Heuristic $\Longrightarrow$ Choose initial weights as $w_{ij} \cong 1 / \sqrt{k_i}$

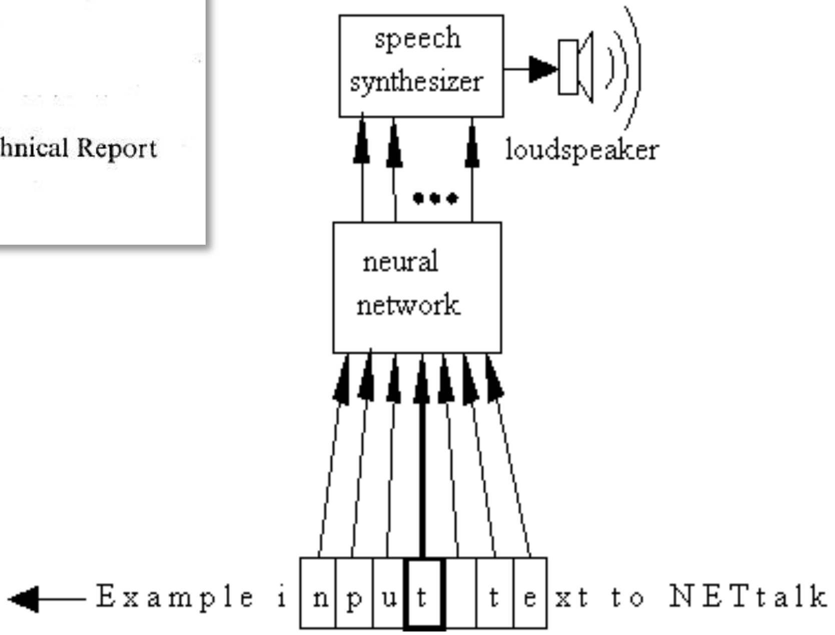
where k_i is the number of units that feed unit *i* (the "fan-in" of *i*)

NETtalk



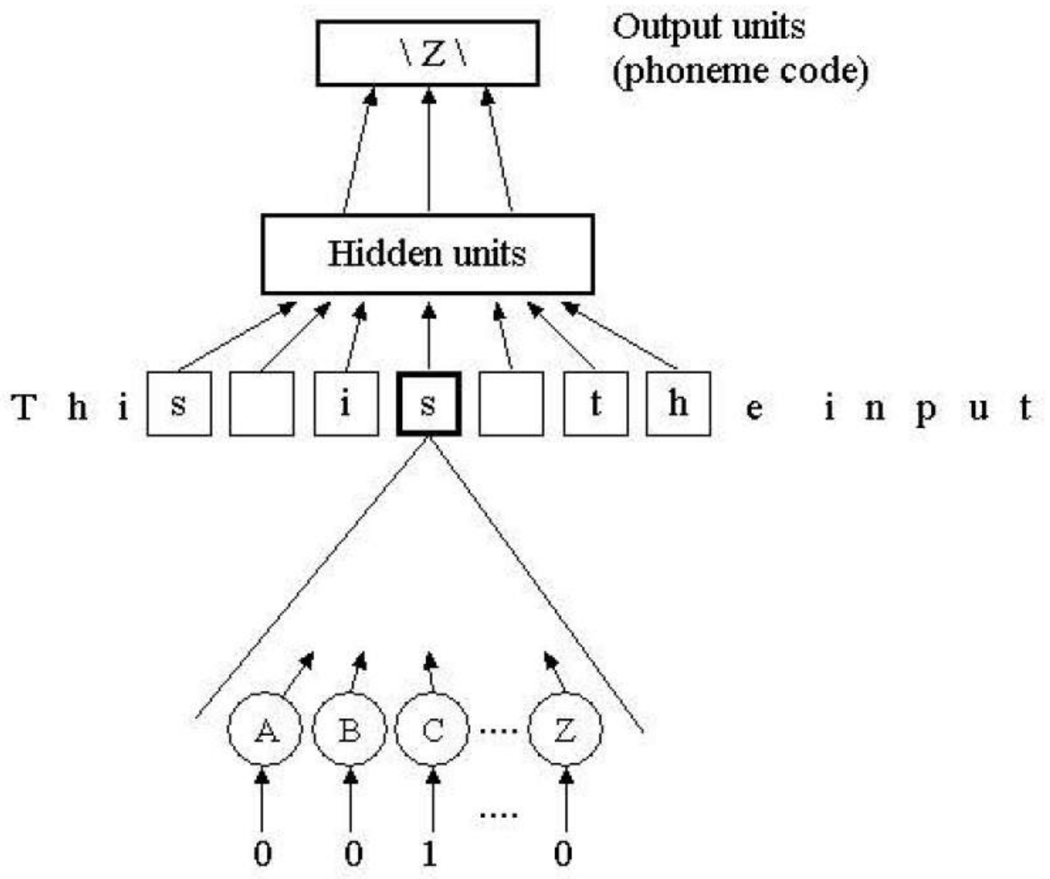
(1986)
Terrence J. Sejnowski and Charles R. Rosenberg

NETtalk: a parallel network that learns to read aloud
The Johns Hopkins University Electrical Engineering and Computer Science Technical Report
JHU/EECS-86/01, 32 pp.



NETtalk neural network speech synthesizer. The NETtalk backpropagation network is trained by a rule-based expert system element of the DECtalk commercial speech synthesis system. NETtalk is then used to replace that element. The result is a new speech synthesis system that has approximately the same overall performance as the original. In other words, the NETtalk neural network becomes functionally equivalent to an expert system with hundreds of rules. The question then becomes: how are these rules represented within the NETtalk neural network? The answer is: nobody really knows.

NETtalk



\Z\
Output units
(phoneme code)
Hidden units
T h i s i s t h e i n p u t
A
B
C
Z
....
....
0
0
1
0

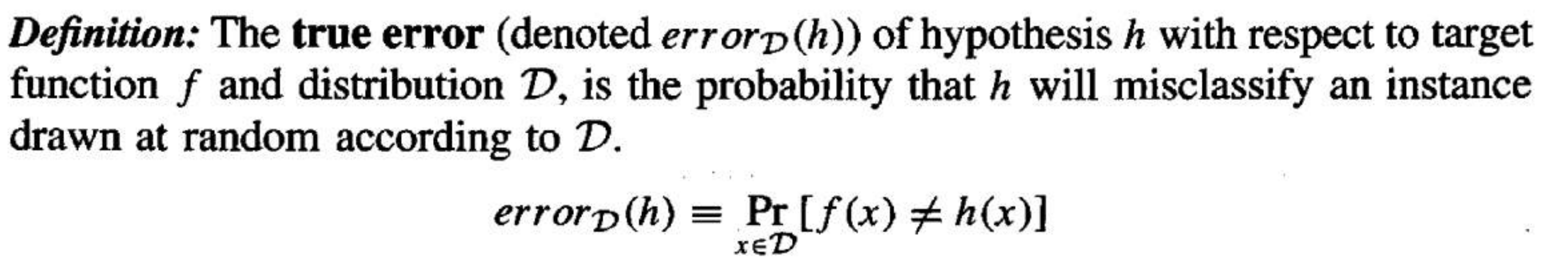
NETtalk

- A network to pronounce English text
- 7 x 29 (=203) input units
- 1 hidden layer with 80 units
- 26 output units encoding phonemes
- Trained by 1024 words in context
- Produce intelligible speech after 10 training epochs
- Functionally equivalent to DEC-talk
- Rule-based DEC-talk was the result of a decade effort by many linguists
- NETtalk learns from examples and, require no linguistic knowledge

Theoretical / Practical Questions

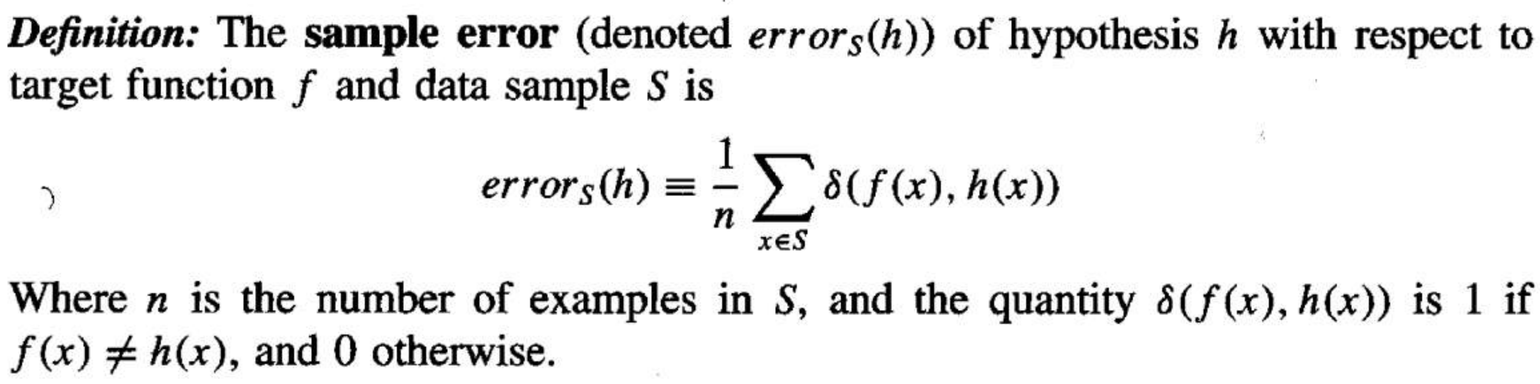
- How many layers are needed for a given task?
- How many units per layer?
- To what extent does representation matter?
- What do we mean by generalization?
- What can we expect a network to generalize?
  - Generalization: performance of the network on data not included in the training set
  - Size of the training set: how large a training set should be for "good" generalization?
  - Size of the network: too many weights in a network result in poor generalization

True *vs* Sample Error



Definition: The **true error** (denoted $error_{\mathcal{D}}(h)$) of hypothesis h with respect to target function f and distribution $\mathcal{D}$, is the probability that h will misclassify an instance drawn at random according to $\mathcal{D}$.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$



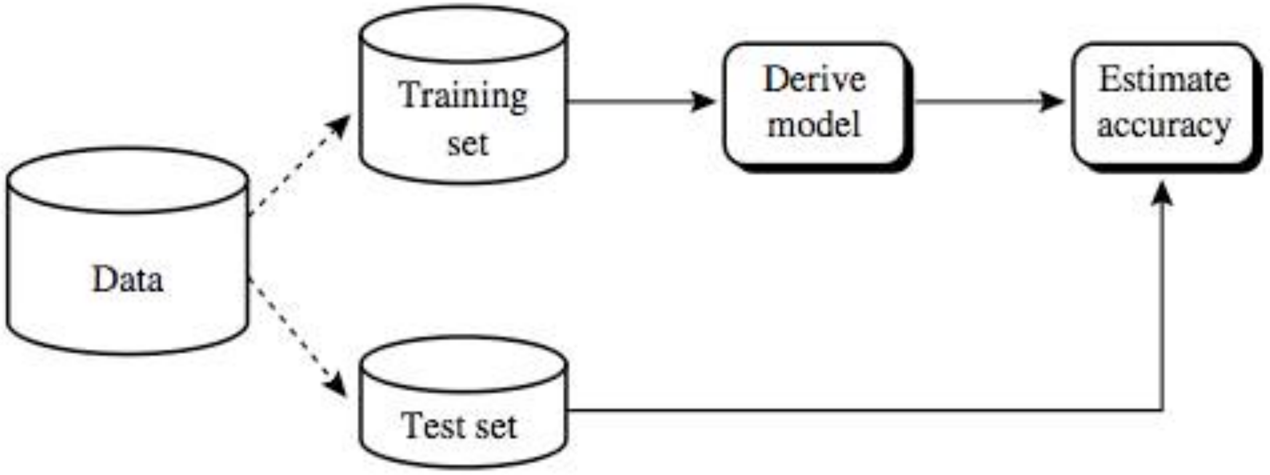
Definition: The **sample error** (denoted $error_S(h)$) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

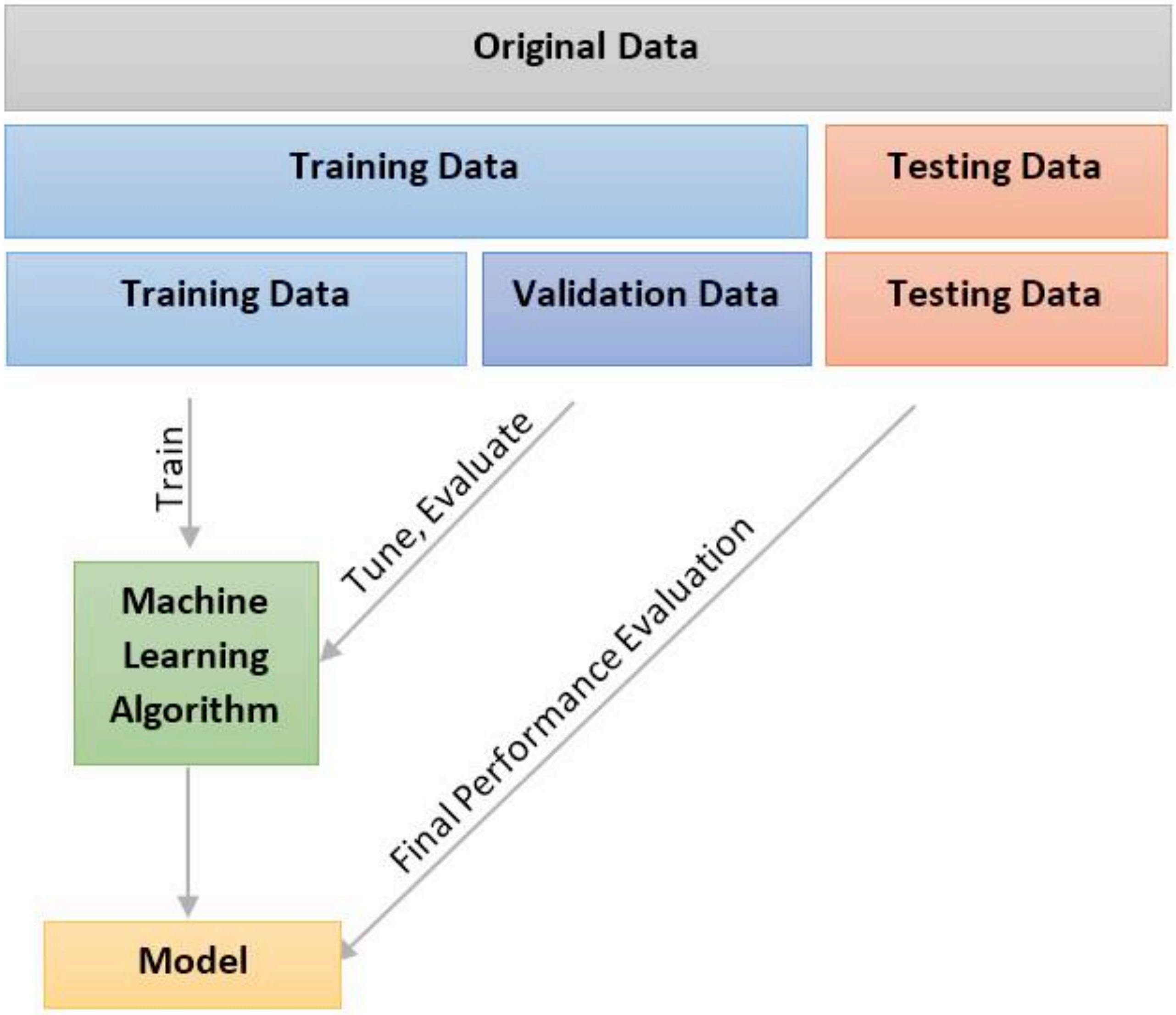
Where n is the number of examples in S, and the quantity $\delta(f(x), h(x))$ is 1 if $f(x) \neq h(x)$, and 0 otherwise.

The **true error** is unknown (and will remain so forever…).
On which sample should I compute the **sample error**?

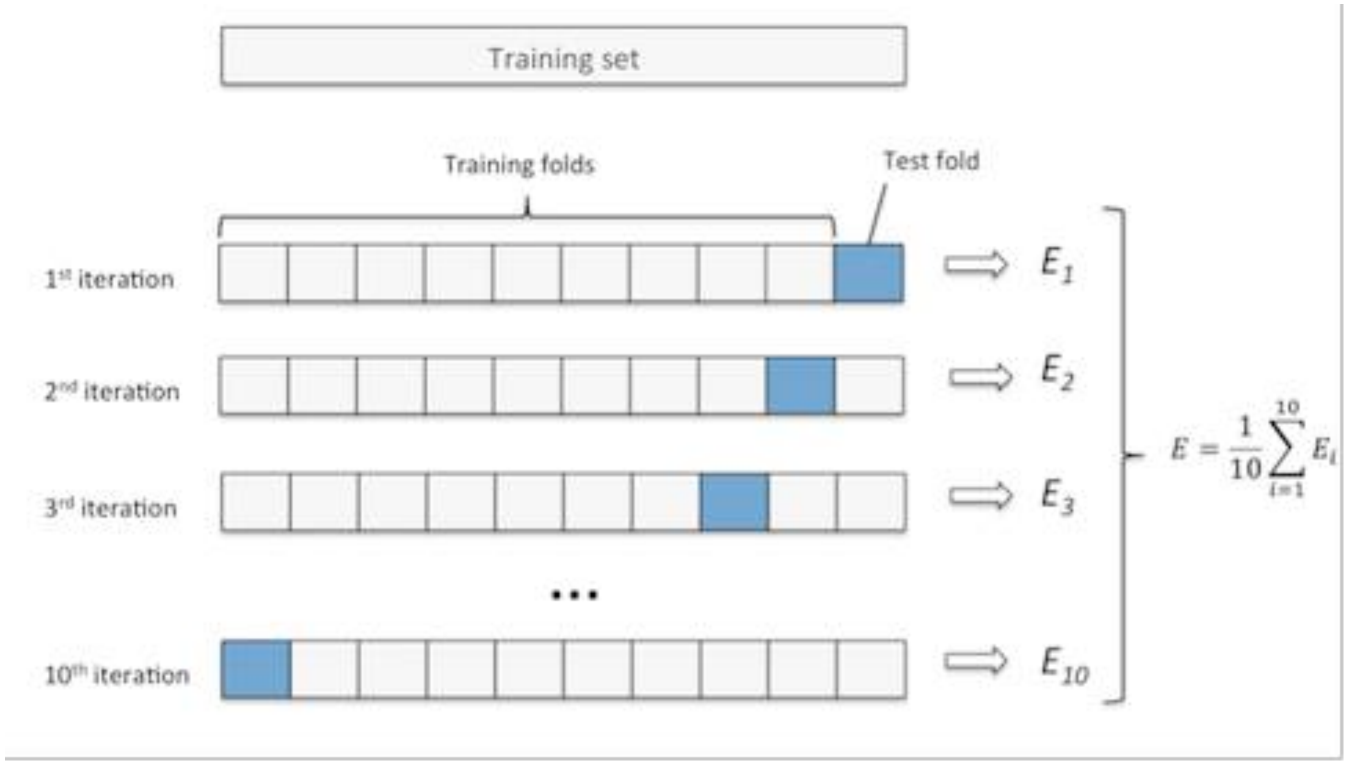
Training *vs* Test Set



Training, Validation and Test Sets

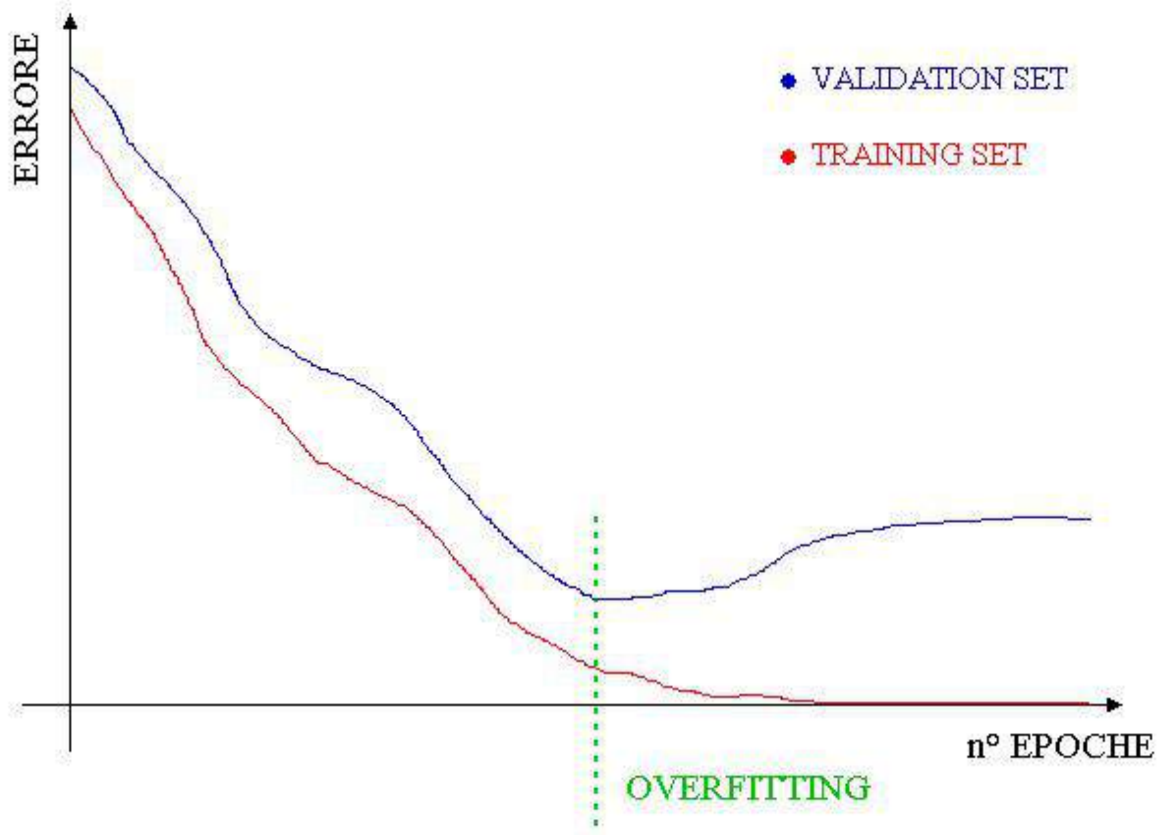


Cross-validation

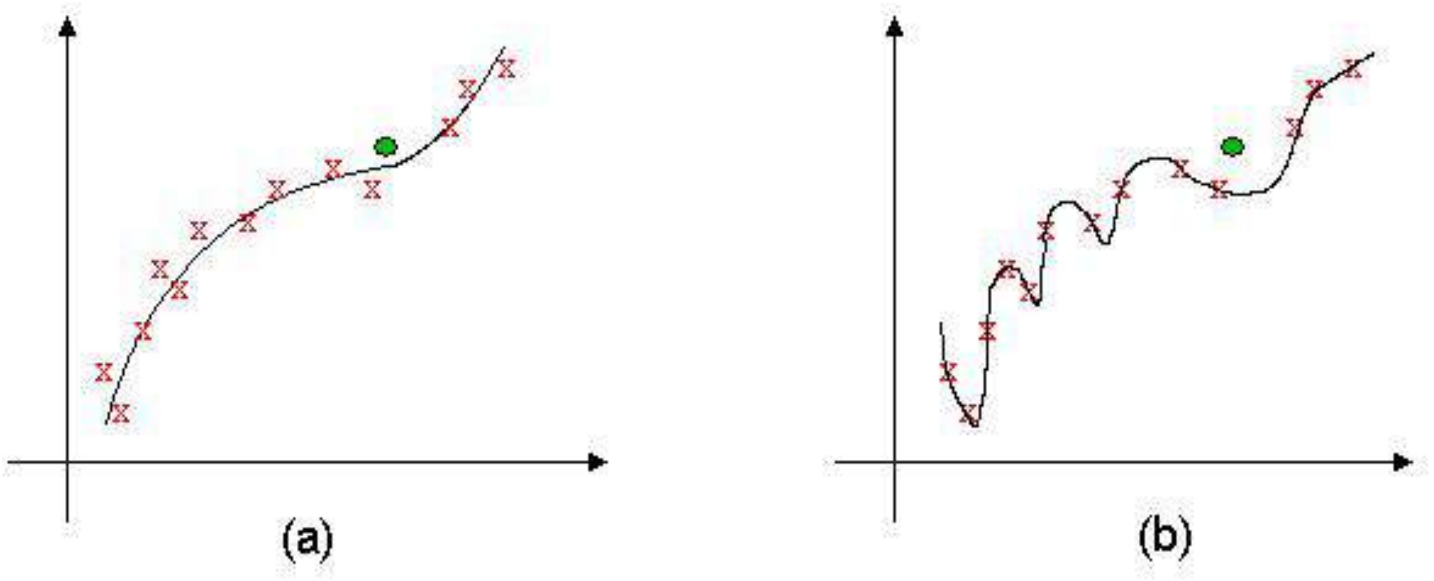


Leave-one-out: using as many test folds as there are examples (size of test fold = 1)

Early Stopping



Overfitting



(a) A good fit to noisy data.(b) Overfitting of the same data: the fit is perfect on the "training set" (x' s), but is likely to be poor on "test set" represented by the circle.

Size Matters

- The size (i.e. the number of hidden units) of an artificial neural network affects both its functional capabilities and its generalization performance

- Small networks could not be able to realize the desired input / output mapping

- Large networks lead to poor generalization performance

The Pruning Approach

Train an over-dimensioned net and then remove redundant nodes and / or connections:

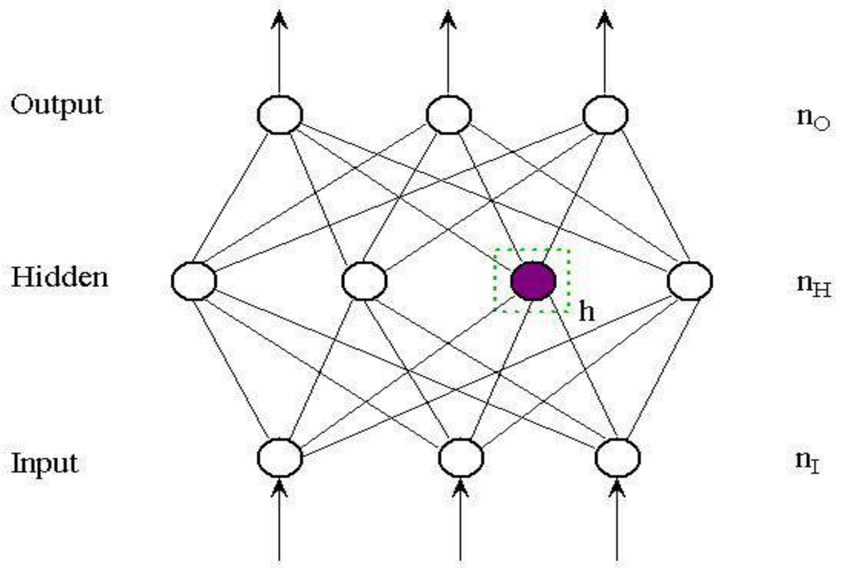
- Sietsma & Dow (1988, 1991)
- Mozer & Smolensky (1989)
- Burkitt (1991)

Adavantages:

- arbitrarily complex decision regions
- faster training
- independence of the training algorithm

An Iterative Pruning Algorithm

Consider (for simplicity) a net with one hidden layer:



Suppose that unit *h* is to be removed:

IDEA: Remove unit *h* (and its in/out connections) and adjust the remaining weights so that the I/O behavior is the same

G. Castellano, A. M. Fanelli, and M. Pelillo, An iterative pruning algorithm for feedforward neural networks, *IEEE Transactions on Neural Networks* 8(3):519-531, 1997.

An Iterative Pruning Algorithm

This is equivalent to solving the system:

$$\underbrace{\sum_{j=1}^{n_h} w_{ij}\, y_j^{(\mu)}}_{\textit{before}} = \underbrace{\sum_{\substack{j=1 \\ j \neq h}}^{n_h} \left(w_{ij} + \delta_{ij}\right) y_j^{(\mu)}}_{\textit{after}} \qquad i = 1 \ldots n_O \ , \ \mu = 1 \ldots P$$

which is equivalent to the following linear system (in the unknown δ's):

$$\sum_{j \neq h} \delta_{ij}\, y_j^{(\mu)} = w_{ih}\, y_h^{(\mu)} \qquad i = 1 \ldots n_O \ , \ \mu = 1 \ldots P$$

An Iterative Pruning Algorithm

In a more compact notation:

$$Ax = b$$

where $A \in \Re^{Pn_o \times n_o(n_h - 1)}$

But solution does not always exists.

Least-square solution :

$$\min_x \| Ax - b \|$$

Detecting Excessive Units

- Residual-reducing methods for LLSPs start with an initial solution x_0 and produces a sequences of points $\{x_k\}$ so that the residuals

$$\| Ax_k - b \| = r_k$$

decrease: $r_k \leq r_{k-1}$

- Starting point: $x_0 = 0 \quad \left(\Rightarrow r_0 = \| b \| \right)$

- Excessive units can be detected so that $\| b \|$ is minimum

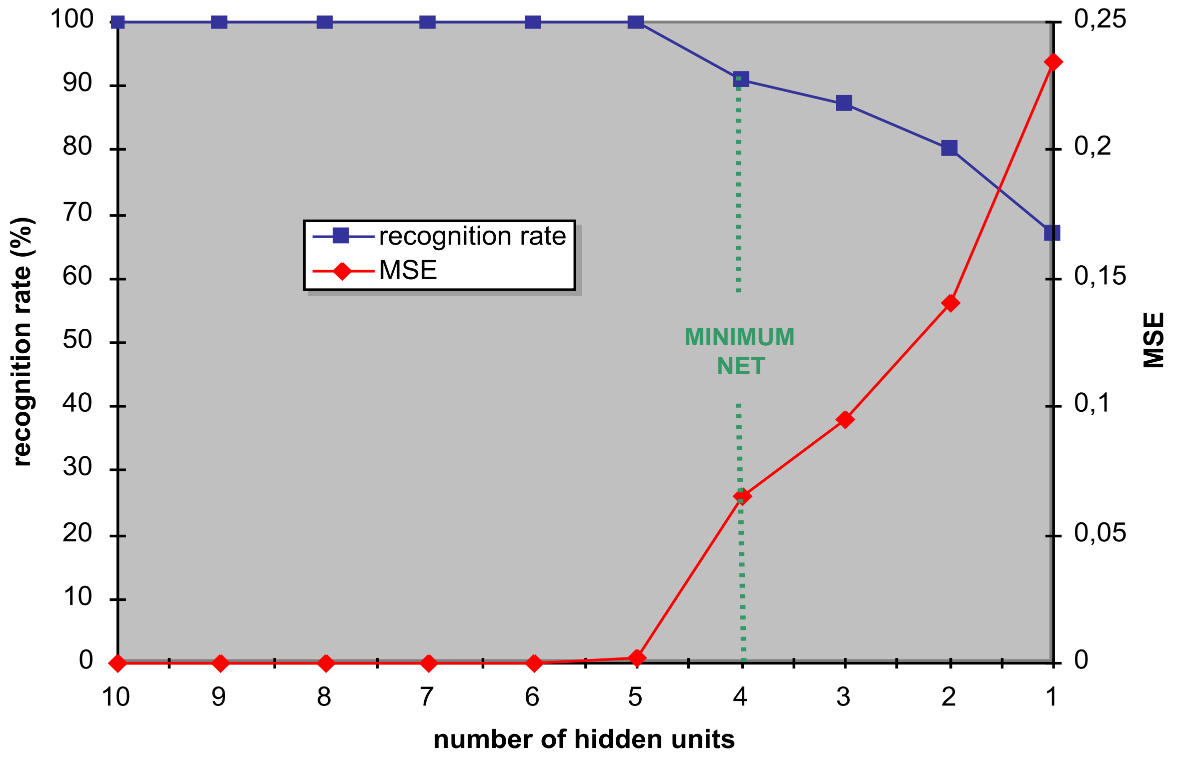
The Pruning Algorithm

```
1)  Start with an over-sized trained network

2)  Repeat

        2.1) find the hidden unit h for which $\|b\|$ is minimum
        2.2) solve the corresponding system
        2.3) remove unit h

    Until  Perf(pruned) – Perf(original) < epsilon

3)  Reject the last reduced network
```

Example: 4-bit parity

Ten initial 4-10-1 networks

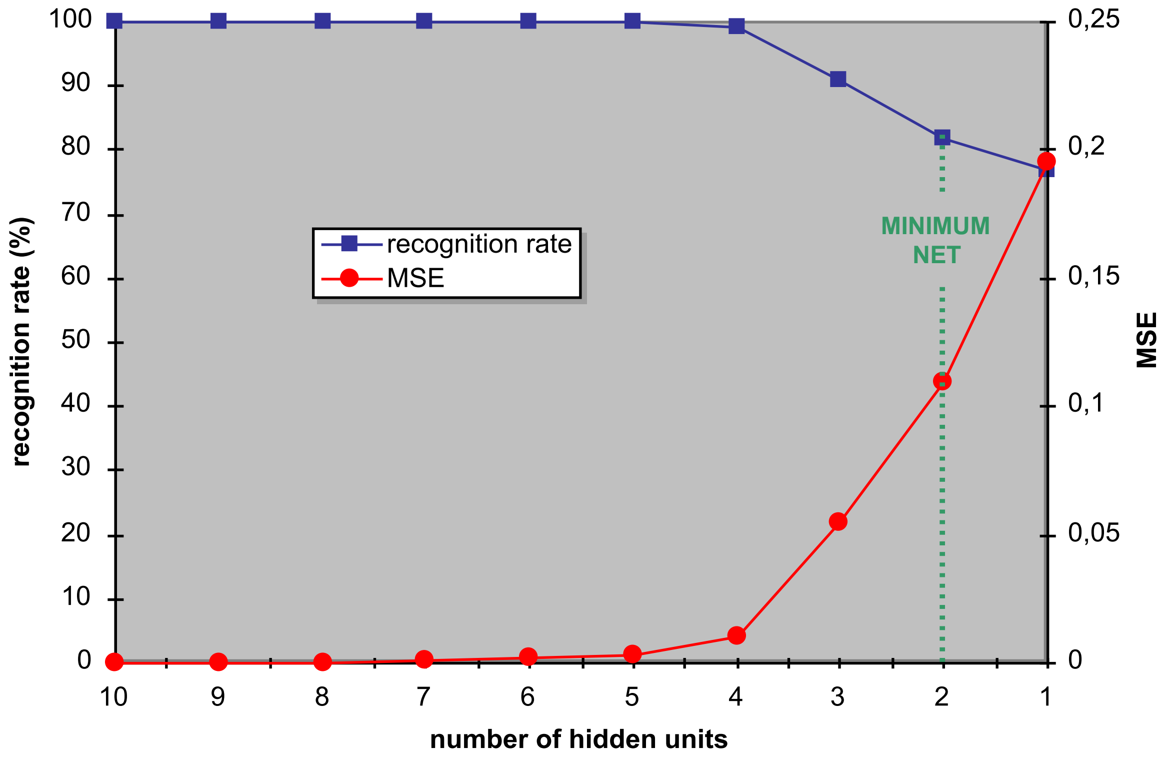
Pruned nets { nine 4-5-1 / one 4-4-1 } 5 hidden nodes (average)



Example: 4-bit simmetry

Ten initial 4-10-1 networks

Pruned nets $\Rightarrow$ 4.6 hidden nodes (average)



Feature selection

Using neural network pruning

Credits: Alberto Scalco

Introduction

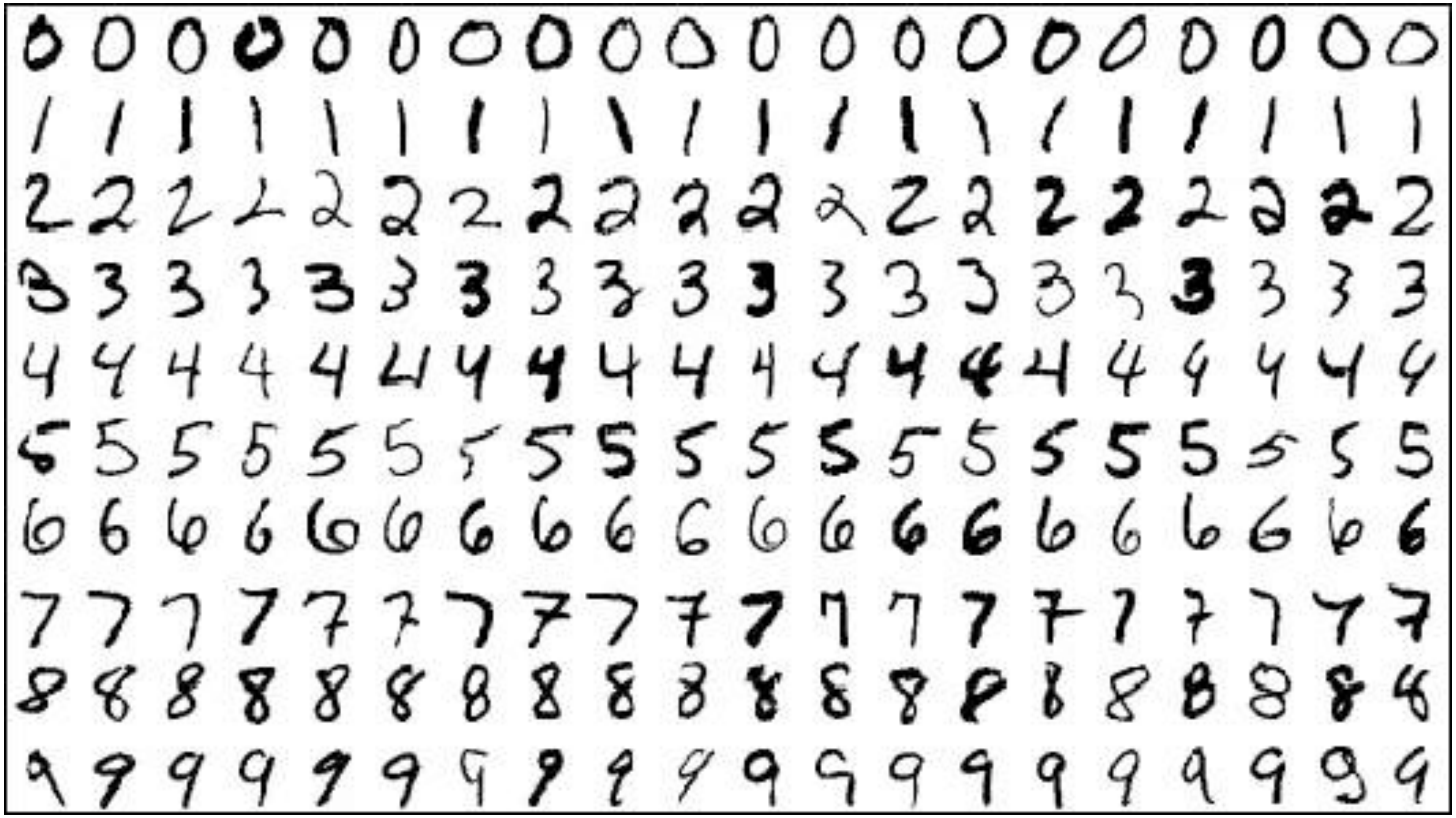
Feature selection

- Classifiers are sensitive to the features used
- Removal of irrelevant and redundant information
- Improve generalization by overfitting reduction
- Extract key feature enhancing the problem interpretation

Idea: apply the pruning algorithm on the input layer

Experimental results

MNIST



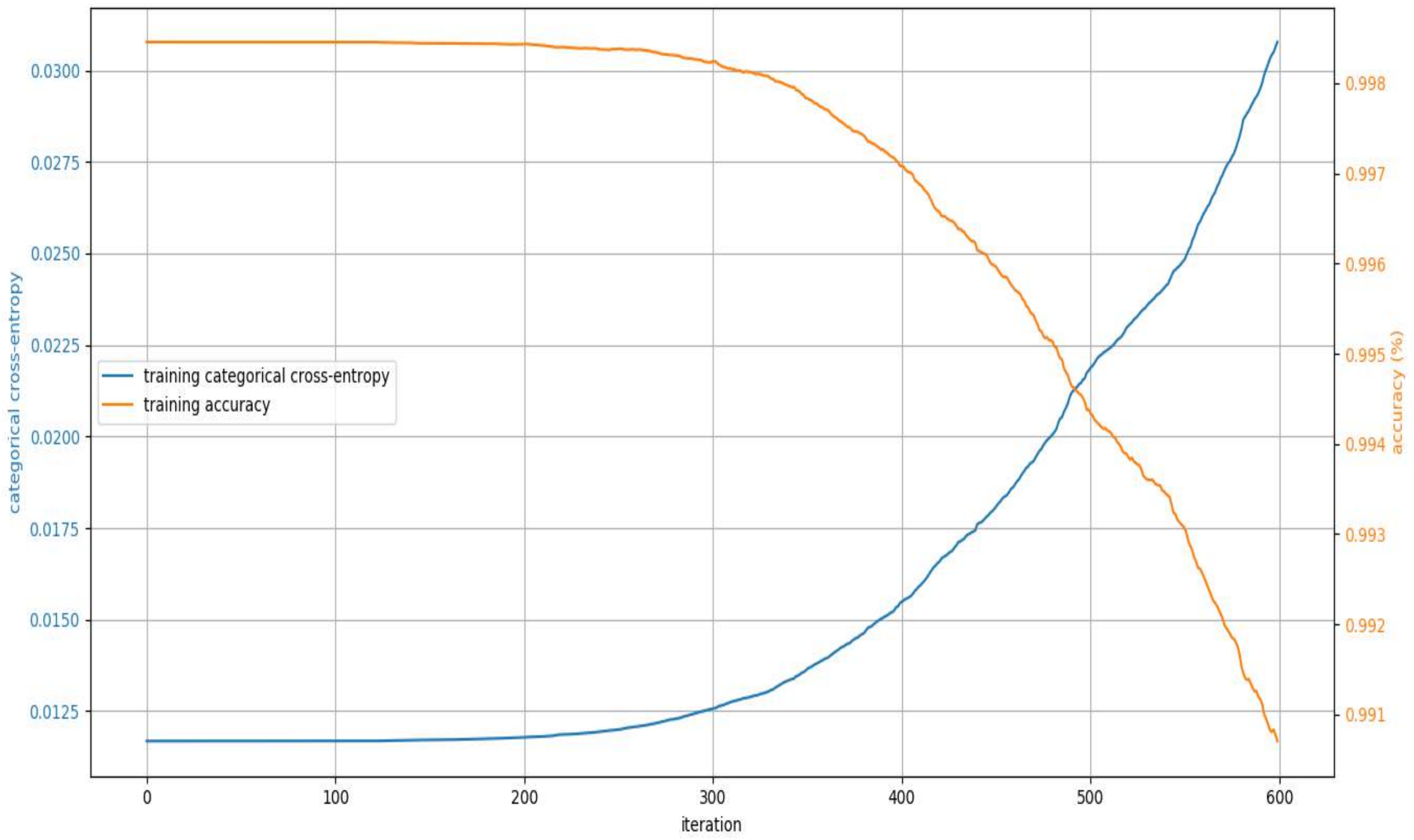
Experimental results

MNIST

- 70,000 images of size 28x28 pixel of handwritten digits
- Training set 48,000, validation set 12,000, test set 10,000
- Fully connected 784-256-10 Network input, hidden and output layer respectively
- Categorical cross-entropy loss function

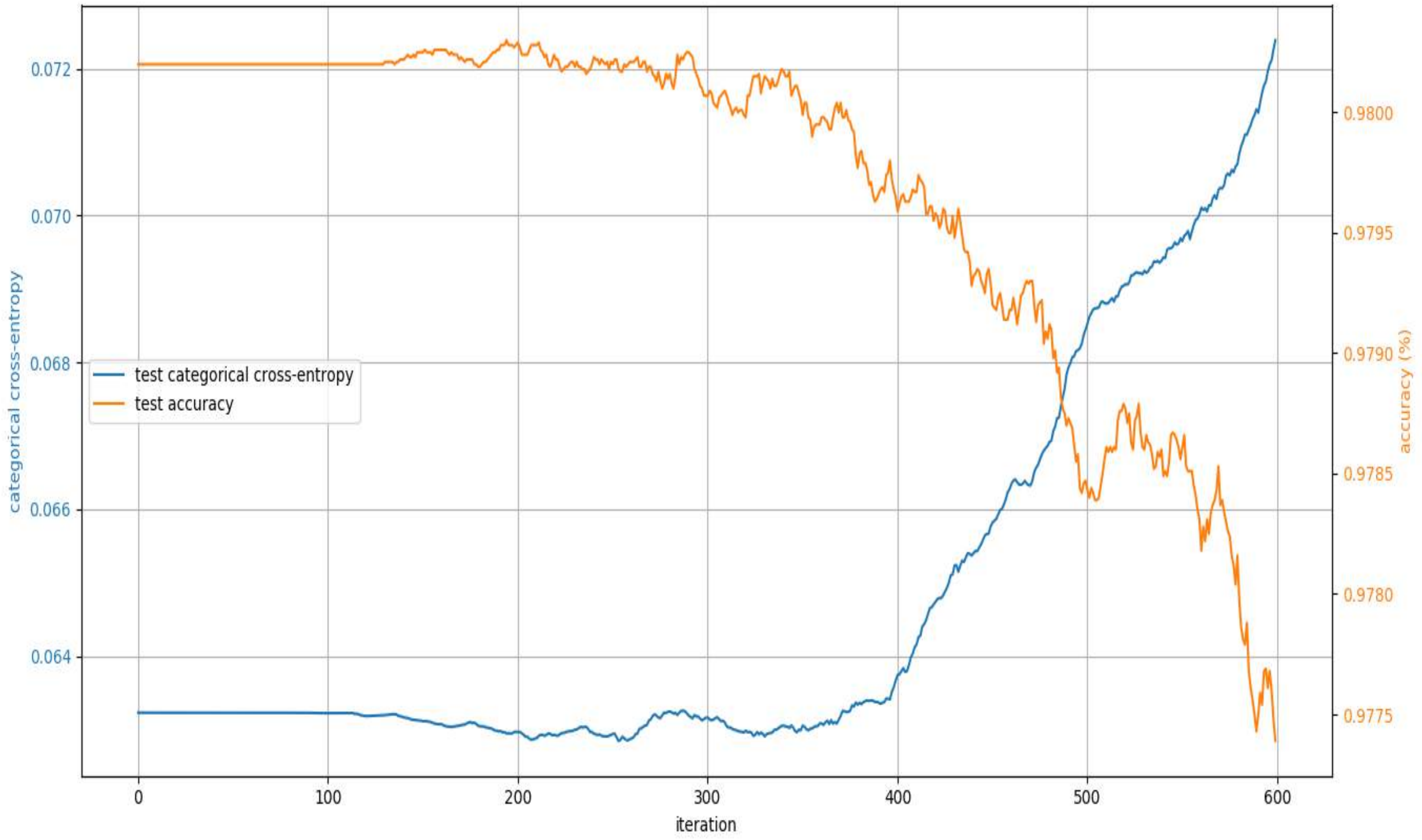
Experimental results

MNIST – training loss and recognition rate



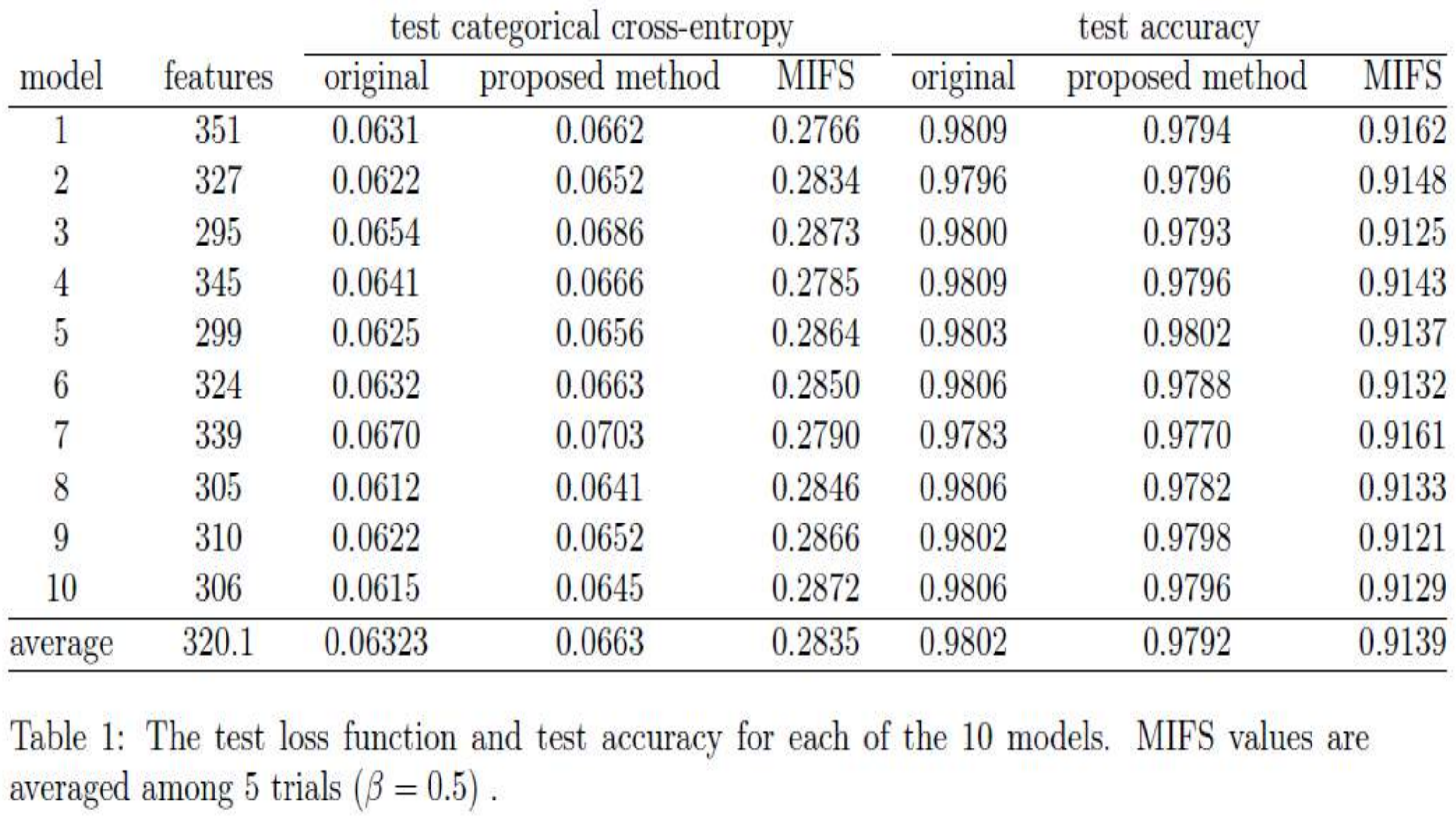
Experimental results

MNIST – test loss and recognition rate



Experimental results

MNIST – algorithm comparison

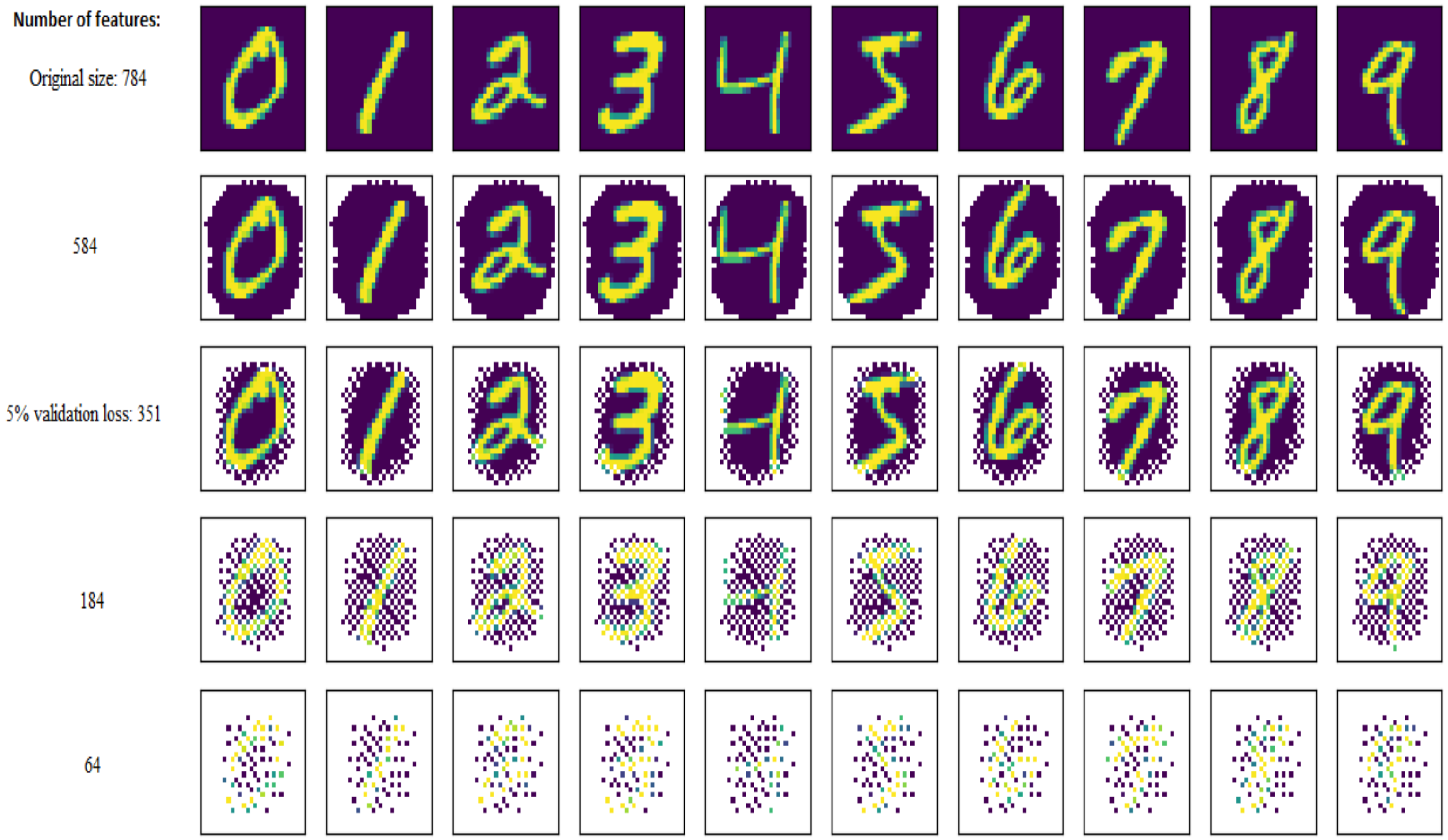


		test categorical cross-entropy			test accuracy		
model	features	original	proposed method	MIFS	original	proposed method	MIFS
1	351	0.0631	0.0662	0.2766	0.9809	0.9794	0.9162
2	327	0.0622	0.0652	0.2834	0.9796	0.9796	0.9148
3	295	0.0654	0.0686	0.2873	0.9800	0.9793	0.9125
4	345	0.0641	0.0666	0.2785	0.9809	0.9796	0.9143
5	299	0.0625	0.0656	0.2864	0.9803	0.9802	0.9137
6	324	0.0632	0.0663	0.2850	0.9806	0.9788	0.9132
7	339	0.0670	0.0703	0.2790	0.9783	0.9770	0.9161
8	305	0.0612	0.0641	0.2846	0.9806	0.9782	0.9133
9	310	0.0622	0.0652	0.2866	0.9802	0.9798	0.9121
10	306	0.0615	0.0645	0.2872	0.9806	0.9796	0.9129
average	320.1	0.06323	0.0663	0.2835	0.9802	0.9792	0.9139

Table 1: The test loss function and test accuracy for each of the 10 models. MIFS values are averaged among 5 trials ($\beta = 0.5$) .

Experimental results

MNIST – selected features



Number of features:
Original size: 784
584
5% validation loss: 351
184
64