An Introduction to Artificial Neural Networks

Outline

- What are Neural Networks?
- Biological Neural Networks
- ANN – The basics
- Feed forward net
- Training
- Example – Voice recognition
- Applications – Feed forward nets
- Recurrency
- Elman nets
- Hopfield nets
- Central Pattern Generators
- Conclusion

What are Neural Networks?

- Models of the brain and nervous system
- Highly parallel
  - Process information much more like the brain than a serial computer
- Learning

- Very simple principles
- Very complex behaviours

- Applications
  - As powerful problem solvers
  - As biological models

What Are They Used For?

- Classification

  Pattern recognition, feature extraction, image matching

- Noise Reduction

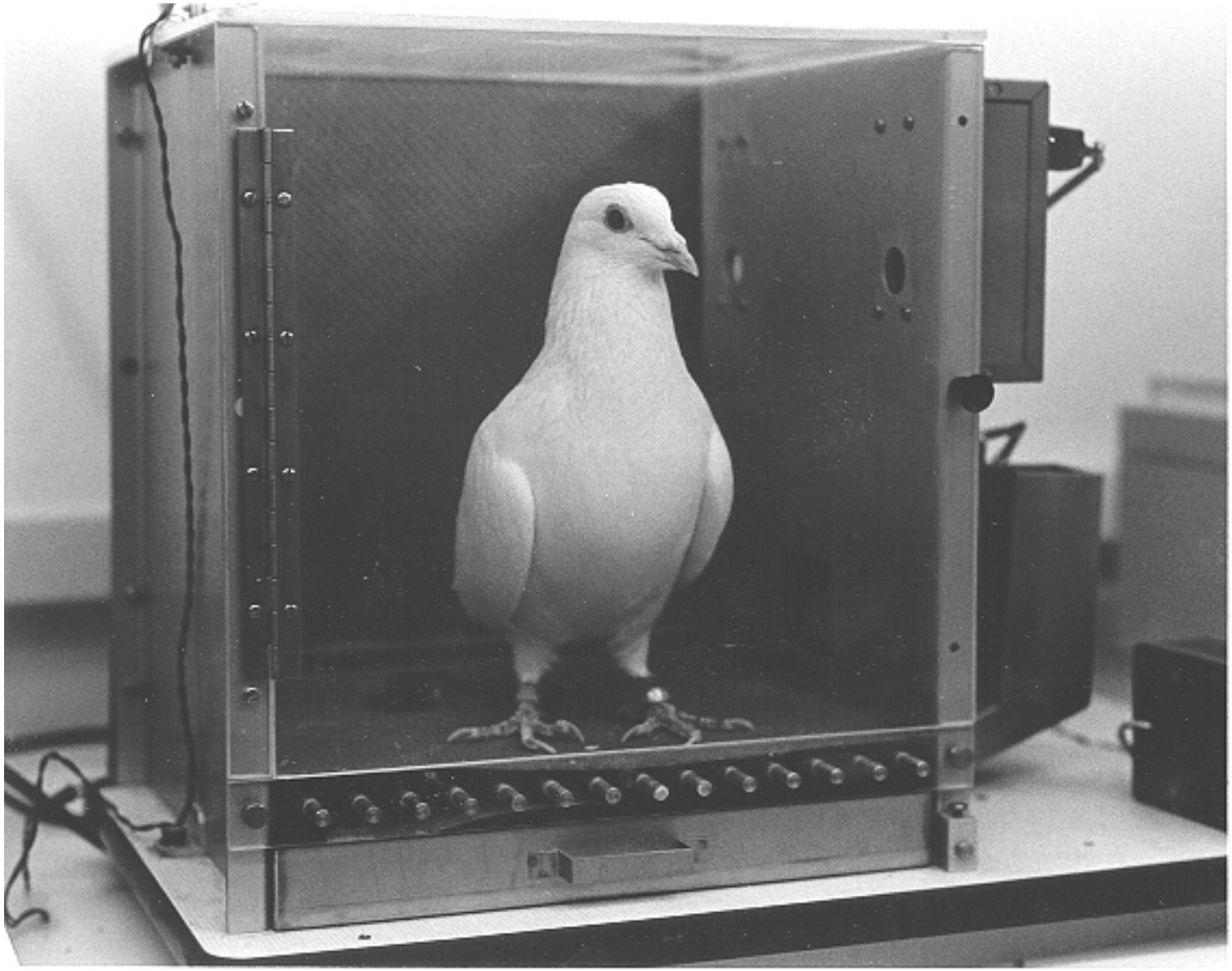
  Recognize patterns in the inputs and produce noiseless outputs

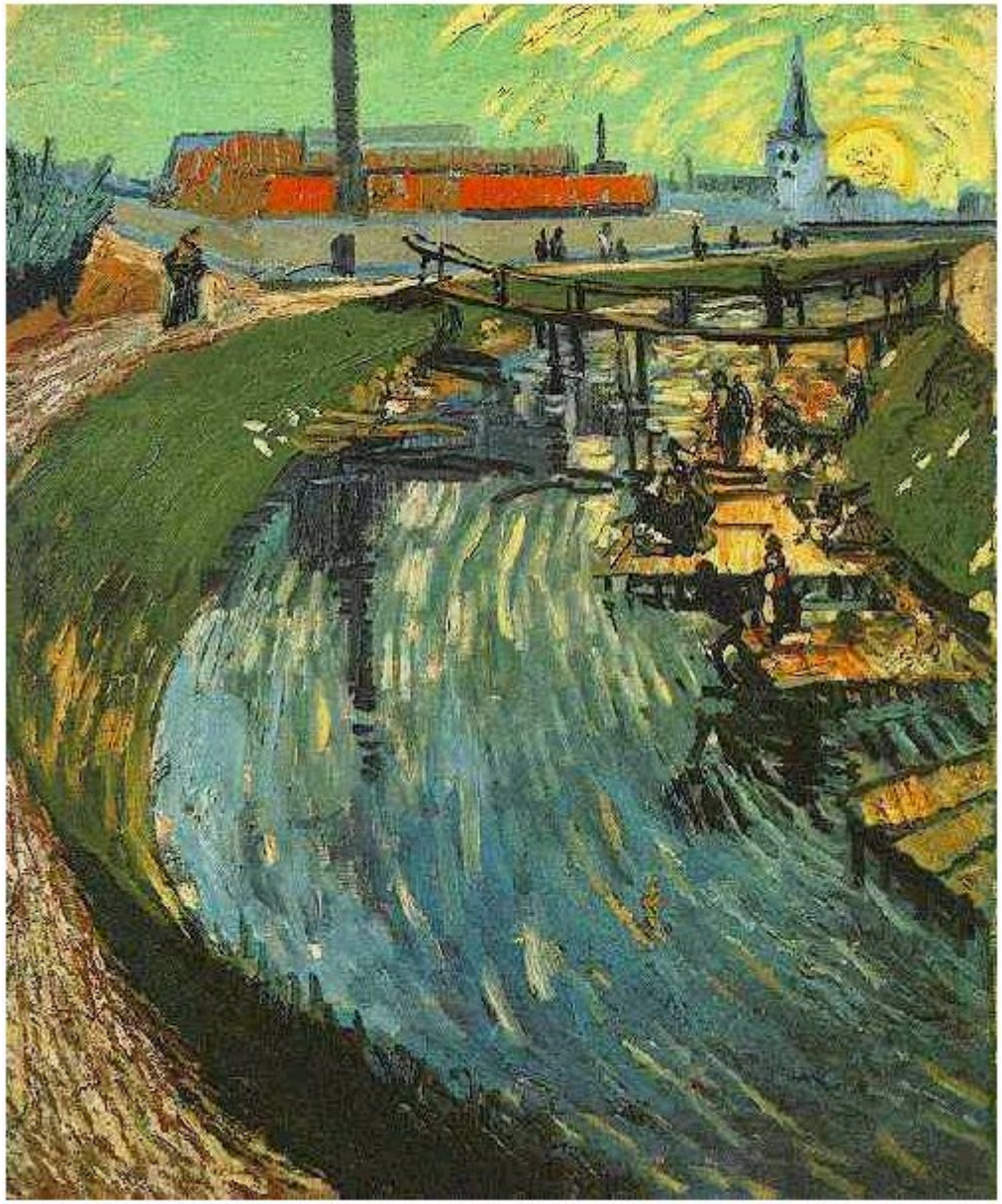
- Prediction

  Extrapolation based on historical data

Biological Neural Nets

- Pigeons as art experts (Watanabe *et al.* 1995)
  - Experiment:
    - Pigeon in Skinner box
    - Present paintings of two different artists (e.g. Chagall / Van Gogh)
    - Reward for pecking when presented a particular artist (e.g. Van Gogh)





- Pigeons were able to discriminate between Van Gogh and Chagall with 95% accuracy (when presented with pictures they had been trained on)

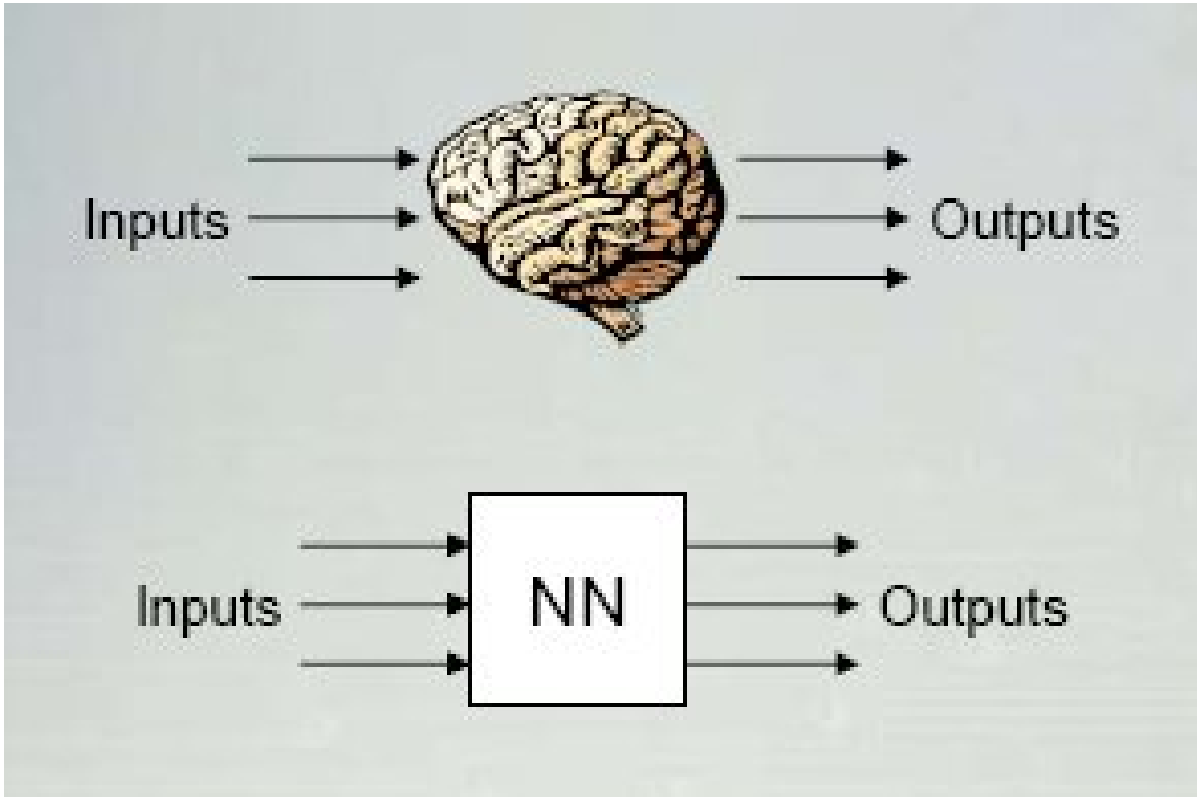
- Discrimination still 85% successful for previously unseen paintings of the artists

- Pigeons do not simply memorise the pictures
- They can extract and recognise patterns (the ‘style’)
- They generalise from the already seen to make predictions

- This is what neural networks (biological and artificial) are good at (unlike conventional computer)

What Are Artificial Neural Networks?

- An extremely simplified model of the brain
- Essentially a function approximator
  - Transforms input into outputs to the best of its ability

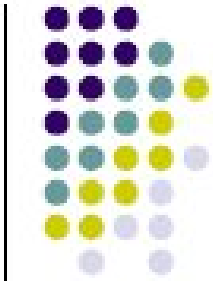


Why Use Neural Networks?

- Ability to learn
  - NN□s figure out how to perform their function on their own
  - Determine their function based only upon sample inputs

- Ability to generalize
  - i.e. produce reasonable outputs for inputs it has not been taught how to deal with

Paradigmi di apprendimento



Esistono diversi paradigmi di apprendimento:

- Apprendimento *supervisionato* (*supervised*):

  per ogni campione del training set è provvista la classe di appartenenza. Obiettivo dell'apprendimento è quello di minimizzare gli errori (o il costo di classificazione).

- Apprendimento *non supervisionato* (*unsupervised*):

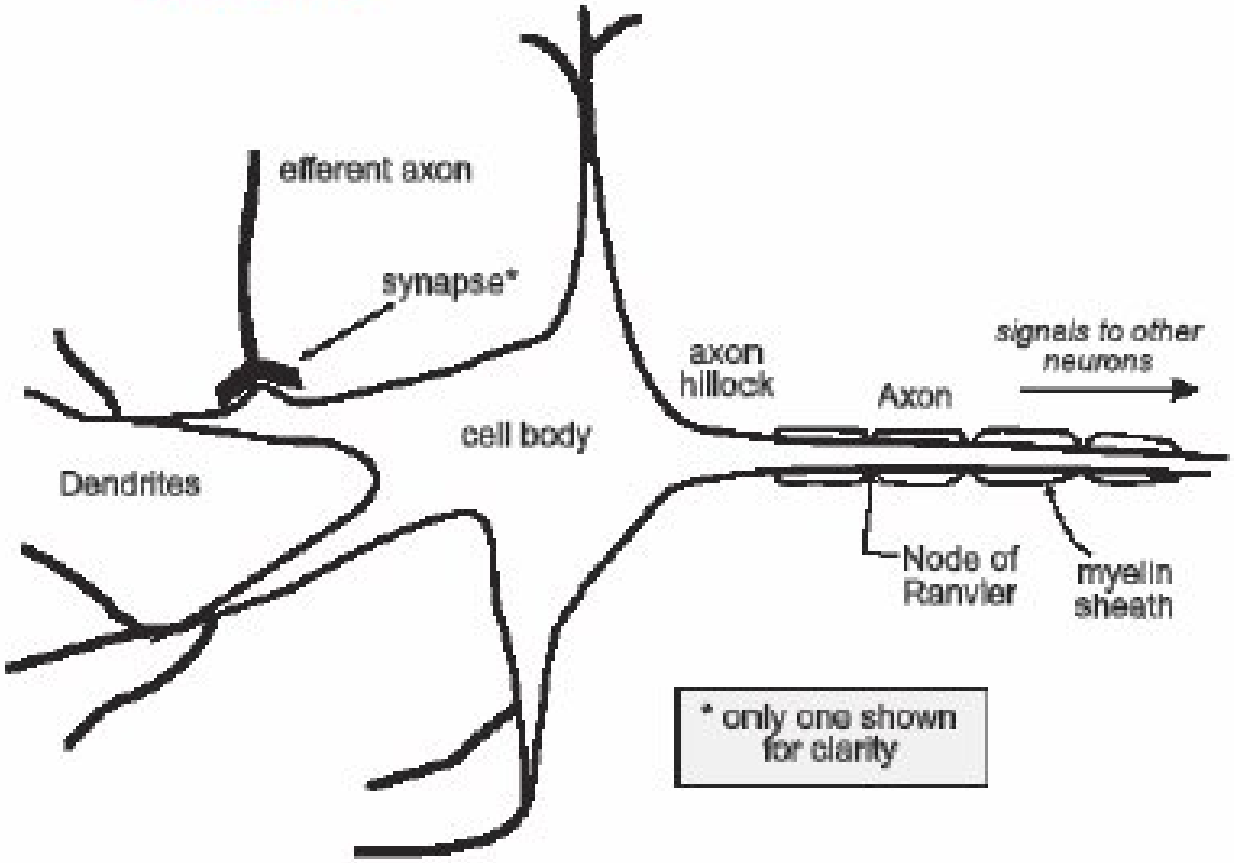
  non sono fornite esplicite informazioni sulla classe dei campioni del training set. Obiettivo dell'apprendimento è quello di formare dei *raggruppamenti* (*clusters*) dei campioni generalmente sulla base di una distanza. Spesso è definito dall'esterno il numero dei clusters da produrre. Questo paradigma viene definito anche *clustering*.

Il neurone biologico

Un neurone è composto di tre parti: il *corpo cellulare* (o *soma*), le *dendriti* e l'*assone* (o *cilindrasse*).

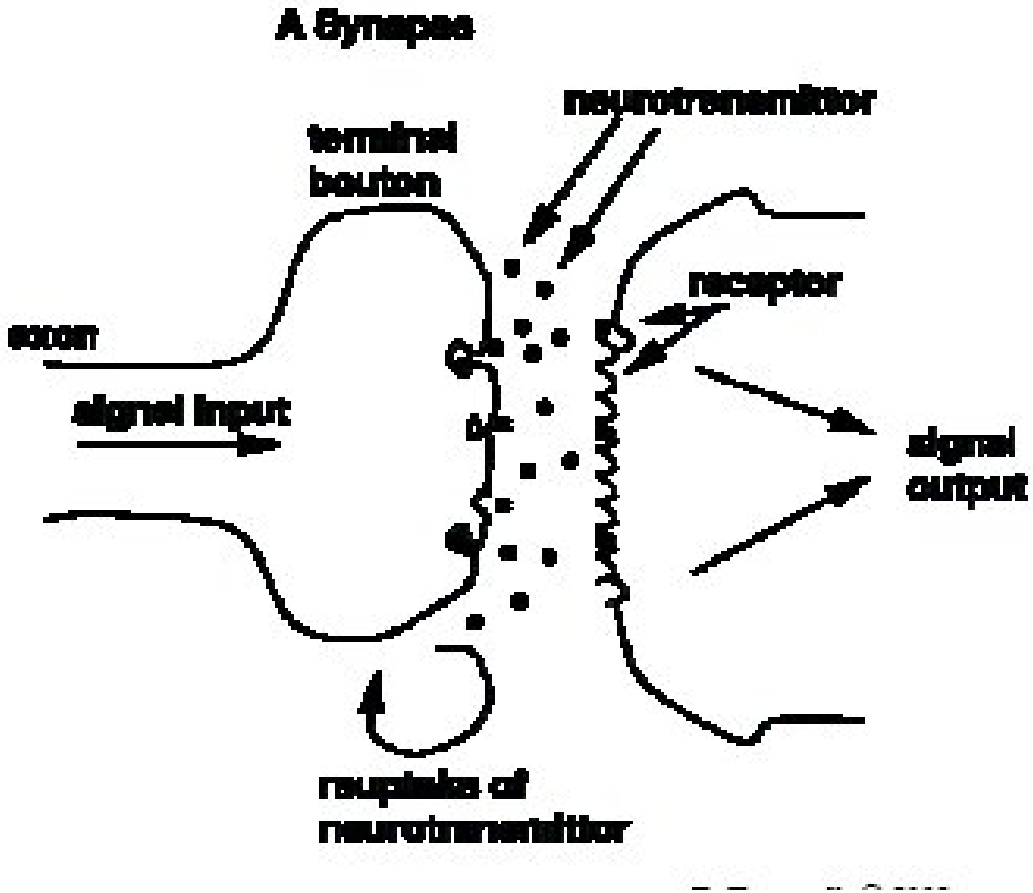
La comunicazione tra neuroni avviene tramite segnali elettrici che sono trasmessi lungo l'assone.

Questo si connette ad una dendrite di un altro neurone tramite un terminale detto *sinapsi*.



Attivazione del neurone

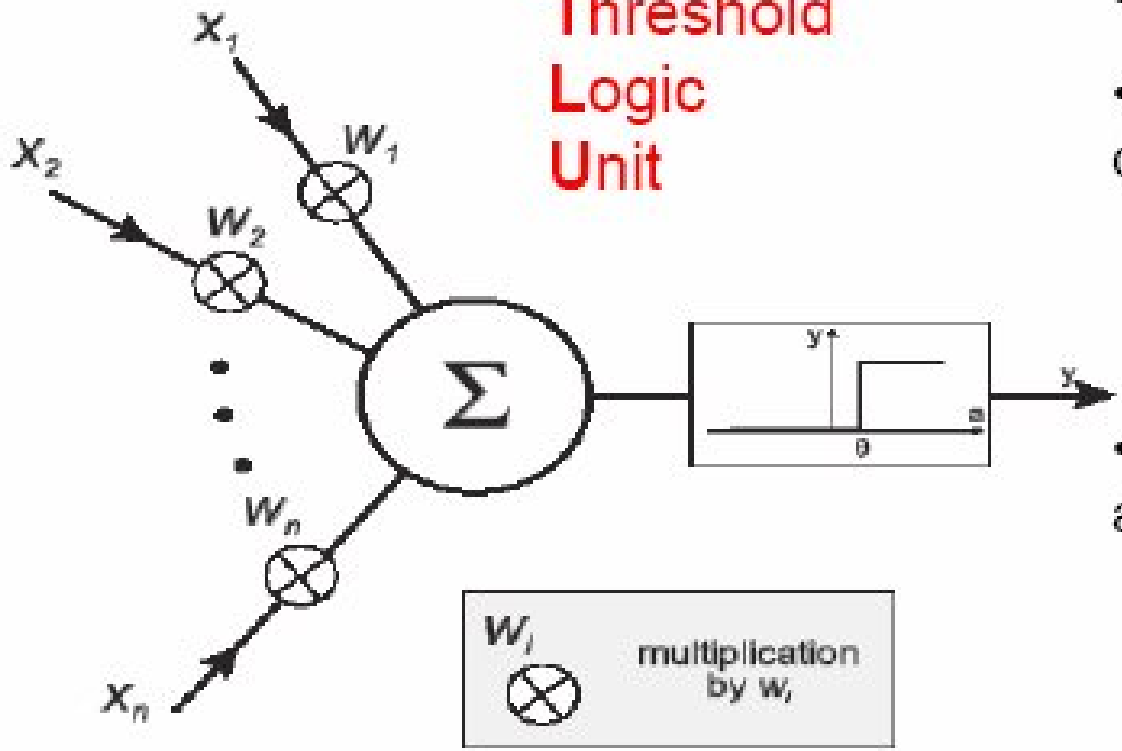
- L'impulso arrivato alla sinapsi stimola il rilascio di neurotrasmettitori
- Questi, captati dai ricettori posti sulla dendrite, inducono una modifica nel potenziale della membrana dendritica (PSP)
- Il PSP può iperpolarizzare o depolarizzare la membrana dendritica, tendendo a inibire o a eccitare la generazione di nuovi impulsi.



Attivazione del neurone

- Ogni PSP attraversa la dendrite su cui è stato generato e si diffonde sul soma, raggiungendo la base dell'assone. Il neurone integra gli effetti delle migliaia di PSP presenti sulle sue dendriti
- Nel caso superi una soglia, genera un impulso che inizia a propagarsi lungo l'assone, così da ricominciare l'intero processo in un nuovo neurone.

Il modello di neurone artificiale di McCulloch-Pitts (1943)



TLU:
Threshold
Logic
Unit

Caratteristiche:

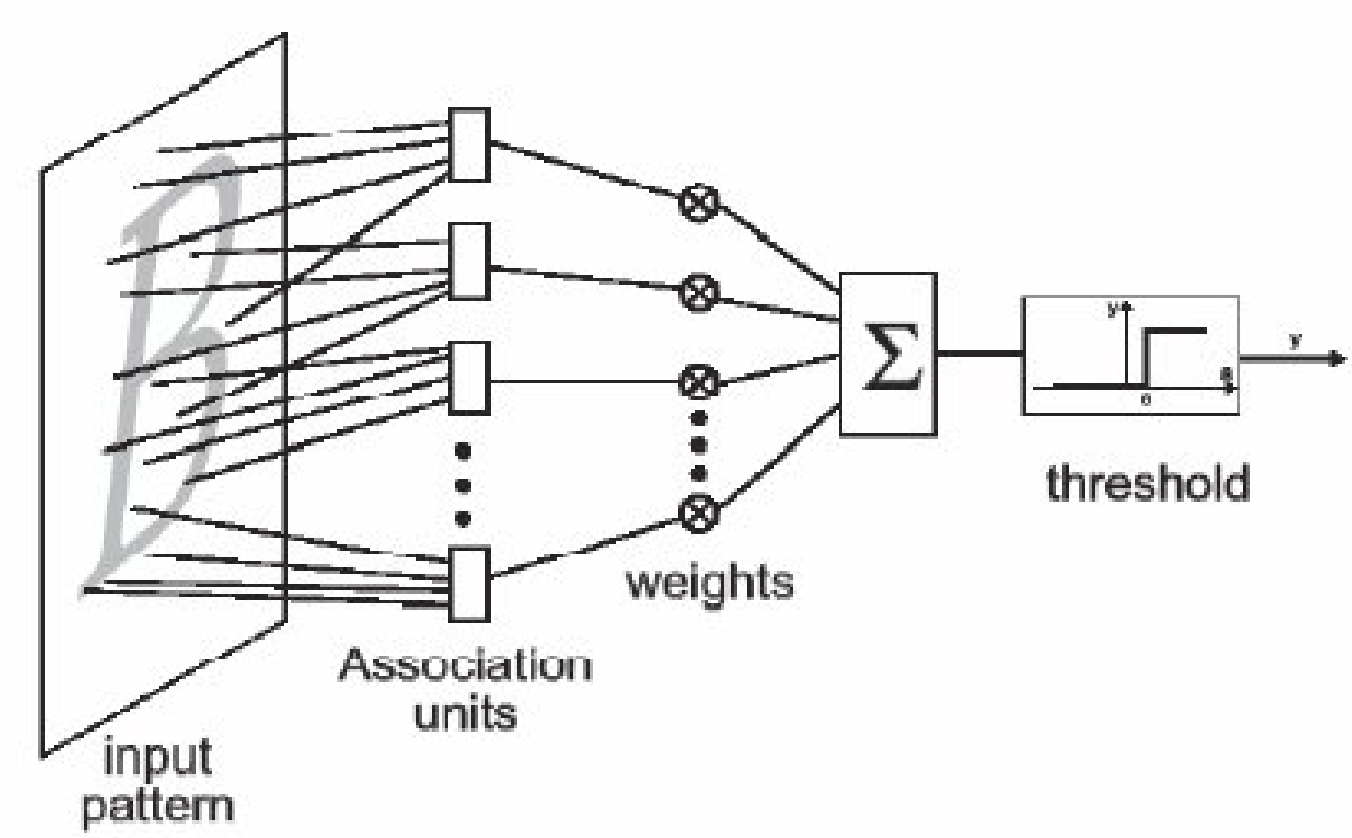
- I segnali sono binari (x_i=0/1)
- La funzione di attivazione è definita come:

$$a = \sum_{i=1}^{n} w_i x_i$$

- L'uscita è definita in base alla regola:

$$y = \begin{cases} 1 & if \quad a \geq \vartheta \\ 0 & if \quad a < \vartheta \end{cases}$$

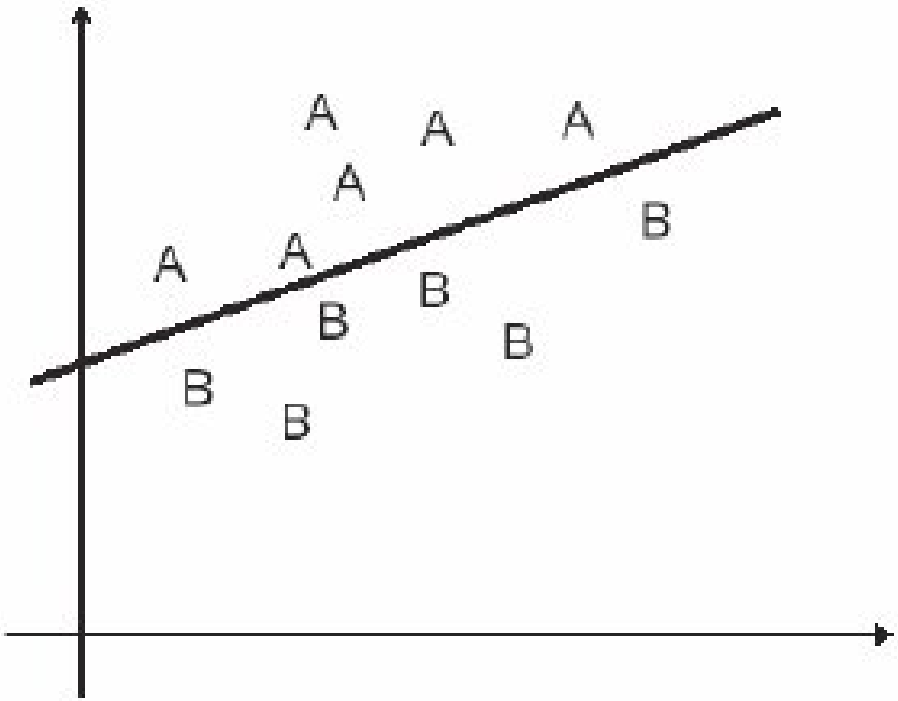
Il *Perceptron* (Rosenblatt, 1962)



2 classi linearmente separabili

Con una TLU è possibile risolvere i problemi in cui le classi siano linearmente separabili.

E se le classi sono più di 2?



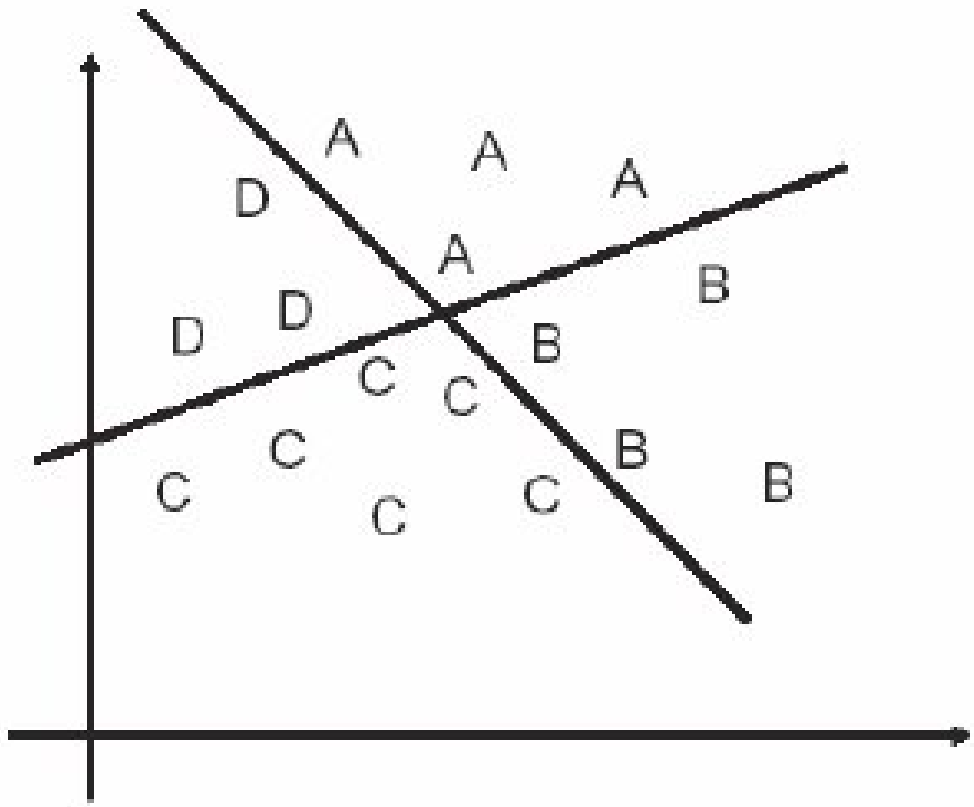
4 classi linearmente separabili

In questo problema le 4 classi sono separabili a coppie.

(A,B) - (C,D)

(A,D) - (B,C)

E' possibile costruire un classificatore basato su TLU ?



4 classi linearmente separabili

Possiamo utilizzare due variabili binarie per codificare le due partizioni.

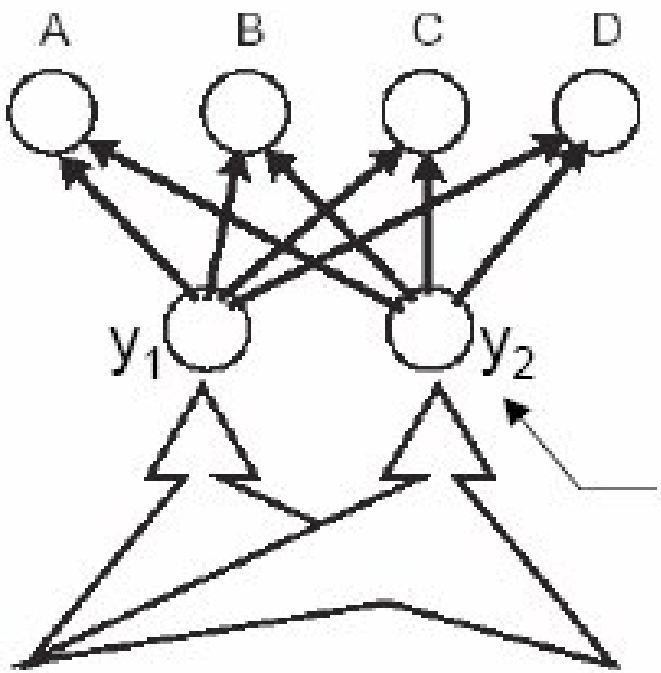
	1	0
y_1	(A B)	(C D)
y_2	(A D)	(B C)

Sulla base dei valori che queste assumono, è possibile risalire alla singola classe.

y_1	y_2	Class
0	0	C
0	1	D
1	0	B
1	1	A

4 classi linearmente separabili

Per realizzare il classificatore, bisogna addestrare due TLU che, in fase operativa, dovranno essere completate con altre unità che realizzano opportune funzioni logiche.

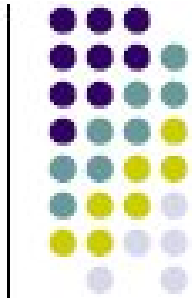


Es.:

A = y_1 and y_2

B = y_1 and not y_2

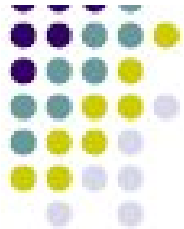
Informazioni necessarie



Possiamo costruire una rete di TLU che risolve il problema. Da notare:

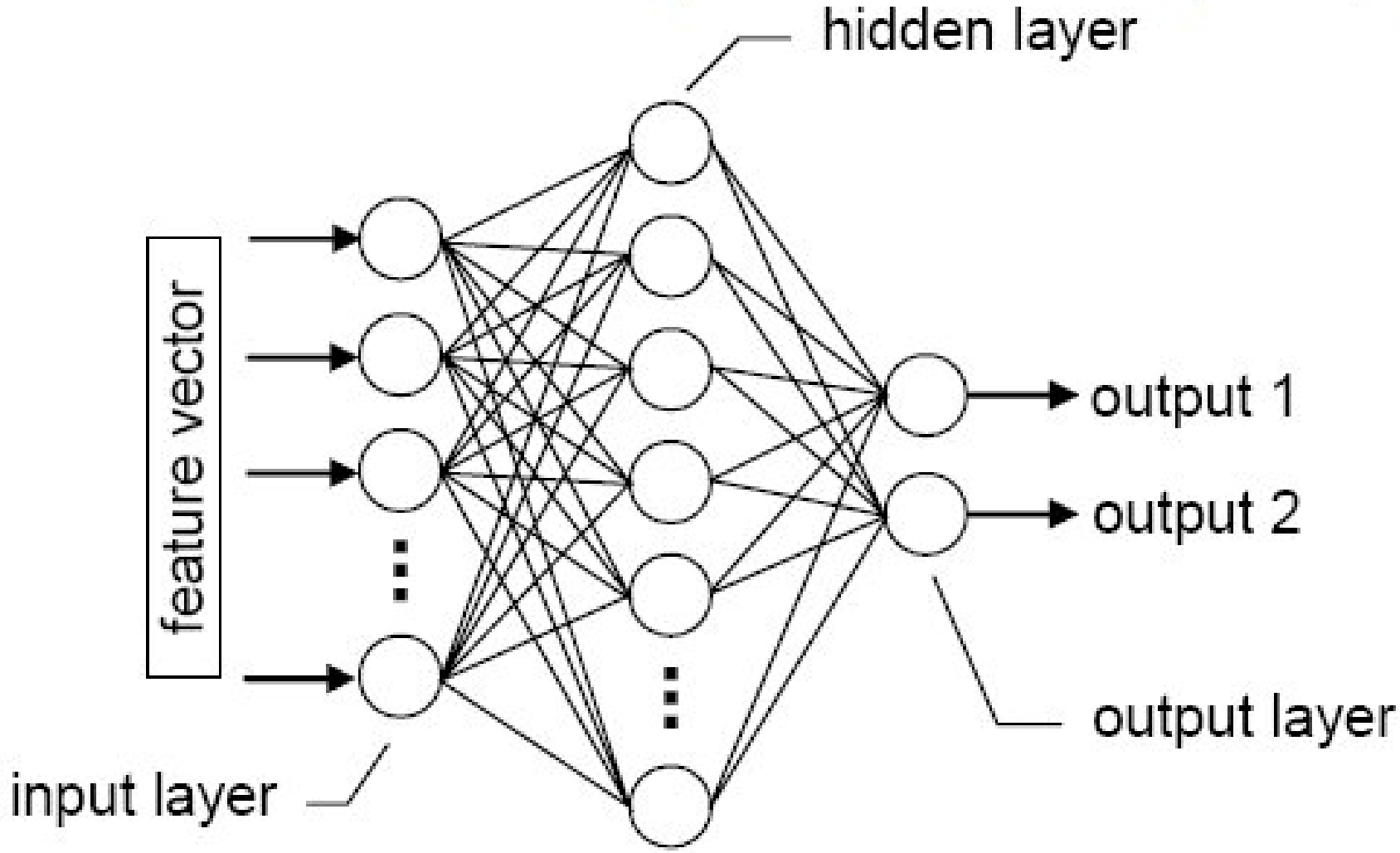
- E' necessario dapprima addestrare le TLU e quindi definire i pesi delle TLU del livello di uscita che realizzano le funzioni logiche
- Il raggruppamento (A,C) (D,B) non avrebbe funzionato
- Altre disposizioni delle classi nello spazio delle features avrebbe richiesto una differente organizzazione del classificatore
- Due tipi di informazione necessari per addestrare le due TLU:
  1. Le 4 classi possono essere separate da iperpiani
  2. AB è linearmente separabile da CD così come AD da BC

Esiste un modo automatico ?



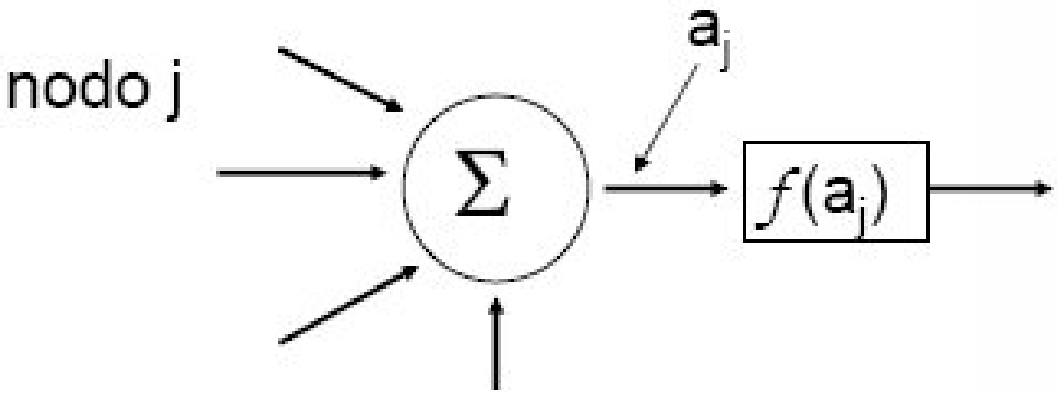
- E' possibile realizzare un algoritmo che si occupi *in toto* della costruzione della rete ?
- Tale algoritmo dovrebbe fissare i pesi delle TLU appartenenti ai due livelli
- Possiamo adattare l'algoritmo visto per una sola TLU ?
- Quali sono i problemi ?
  - Definire la funzione di errore
  - Relazione tra l'uscita ed i pesi
  - Rete su più livelli

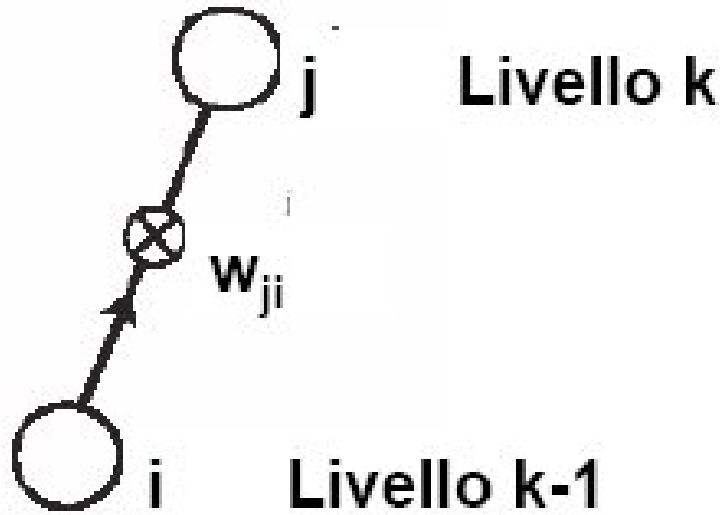
Struttura della rete (feed forward compl. connessa)



Definizione della rete

Ogni nodo è strutturato come:

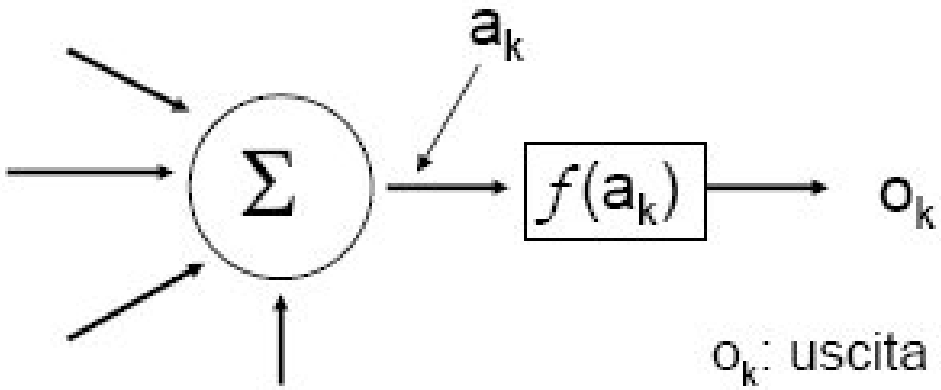




Per il momento, non facciamo ipotesi su $f(.)$

Definizione della rete

nodo di output



o_k: uscita effettiva del nodo di out. k-mo

d_k: uscita desiderata del nodo di out. k-mo

Errore = $|d_k - o_k|$

Definizione della funzione di errore

- Possiamo far riferimento alla somma dell'errore quadratico rilevato sul livello di uscita:

$$E = \frac{1}{2}\sum_{k}(d_k - o_k)^2$$

- L'errore misura lo scostamento tra output desiderato e output effettivo
- E' facilmente differenziabile
- E' adatto per molte applicazioni

L'algoritmo di training

- Cerchiamo di applicare l'algoritmo a gradiente discendente.
- Anche qui sarà valida l'equazione di aggiornamento dei pesi:

$$w_{ji}^{n+1} = w_{ji}^{n} - \eta \left. \frac{\partial E}{\partial w_{ji}} \right|_{w^n}$$

Calcoliamo il gradiente

- Per la *chain rule:*

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}}$$

- Calcoliamo i singoli termini:

$$\frac{\partial E}{\partial o_k} = -(d_k - o_k)$$

$$\frac{\partial o_k}{\partial a_k} = \frac{\partial f(a_k)}{\partial a_k} = f'(a_k)$$

$$\frac{\partial a_k}{\partial w_{kj}} = \frac{\partial}{\partial w_{kj}} \sum_j w_{kj} x_j$$

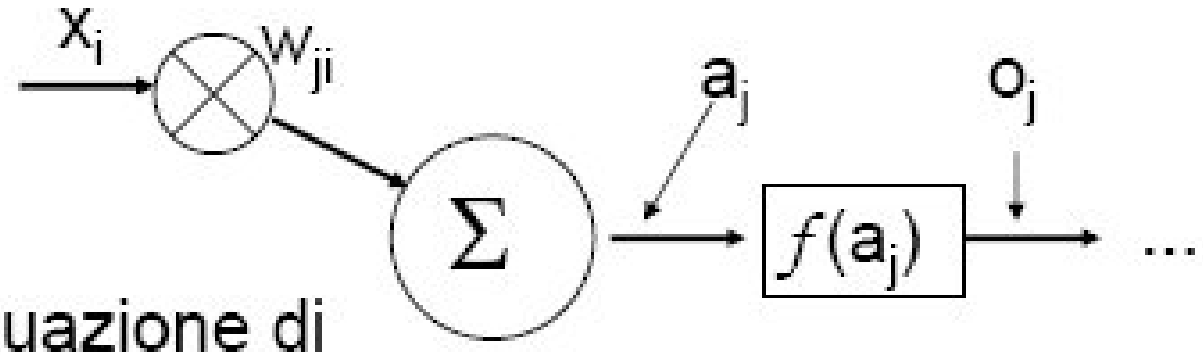
L'equazione di aggiornamento

- L'equazione di aggiornamento è allora:

$$w_{kj}^{n+1} = w_{kj}^{n} + \eta(d_k - o_k) f'(a_k) x_j \Big|_{w^n}$$

- **Achtung!**
  Questa regola è applicabile solo ai neuroni del livello di output. Perché ?
- Come si aggiornano i pesi degli altri neuroni ?

L'equazione di aggiornamento per i neuroni interni



Per valutare l'equazione di aggiornamento:

$$w_{ji}^{n+1} = w_{ji}^{n} - \eta \left. \frac{\partial E}{\partial w_{ji}} \right|_{w^n}$$

dobbiamo calcolare il gradiente: $\dfrac{\partial E}{\partial w_{ji}}$

che possiamo scrivere:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

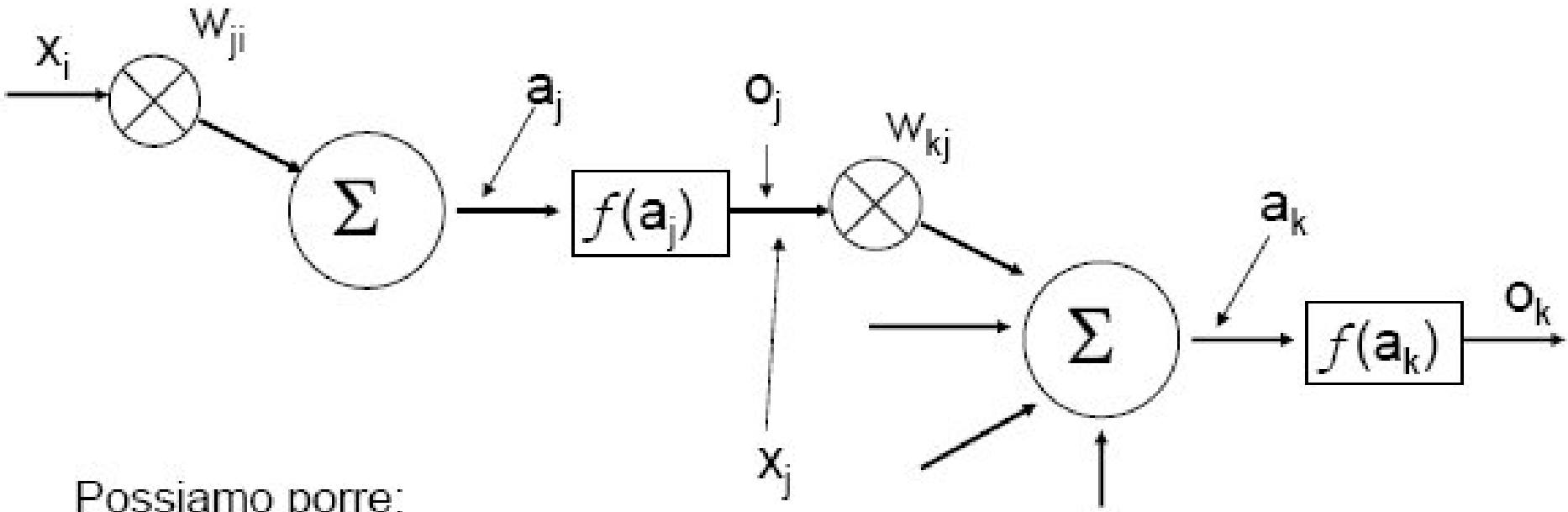
L'equazione di aggiornamento per i neuroni interni

- Gli ultimi termini sono semplicemente:

$$\frac{\partial o_j}{\partial a_j} = \frac{\partial f(a_j)}{\partial a_j} = f'(a_j) \qquad \frac{\partial a_j}{\partial w_{ji}} = \frac{\partial}{\partial w_{ji}} \sum_i w_{ji} x_i = x_i$$

- Come calcolare $\frac{\partial E}{\partial o_j}$?

L'equazione di aggiornamento per i neuroni interni



Possiamo porre:

$$\frac{\partial E}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_k (d_k - o_k)^2 \quad \text{dove:} \quad o_k = f\left(\sum_j w_{kj} o_j\right)$$

L'equazione di aggiornamento per i neuroni interni

Si ha, quindi: $$\frac{\partial E}{\partial o_j} = -\sum_k (d_k - o_k) f'(a_k) w_{kj}$$

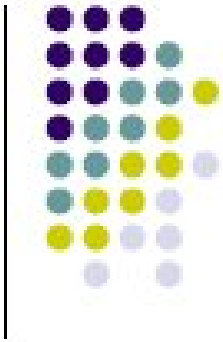
Per cui, il gradiente diventa:

$$\frac{\partial E}{\partial w_{ji}} = -\left[\sum_k (d_k - o_k) f'(a_k) w_{kj}\right] f'(a_j) x_i$$

E arriviamo finalmente all'equazione di aggiornamento:

$$w_{ji}^{n+1} = w_{ji}^{n} + \eta \left[\sum_k (d_k - o_k) f'(a_k) w_{kj}\right] f'(a_j) x_i \Big|_{w^n}$$

Riassumendo



- Neuroni di output:

$$w_{kj}^{n+1} = w_{kj}^{n} + \eta(d_k - o_k) f'(a_k) x_j \Big|_{w^n} = w_{kj}^{n} + \eta \delta_k x_j \Big|_{w^n}$$

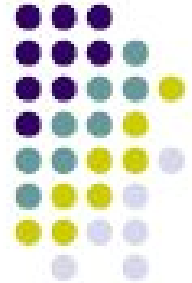
- Neuroni interni:

$$w_{ji}^{n+1} = w_{ji}^{n} + \eta \left[\sum_k (d_k - o_k) f'(a_k) w_{kj} \right] f'(a_j) x_i \Big|_{w^n} =$$

$$w_{ji}^{n} + \eta \left[\sum_k \delta_k w_{kj} \right] f'(a_j) x_i \Big|_{w^n} = w_{ji}^{n} + \eta \delta_j x_i \Big|_{w^n}$$

avendo posto: $\delta_k = (d_k - o_k) f'(a_k)$ e $\delta_j = \left[\sum_k \delta_k w_{kj} \right] f'(a_j)$

Algoritmo di Back Propagation



1. si pone in ingresso ai nodi di input un campione del TS
2. i nodi del primo livello nascosto calcolano le loro uscite che vengono poste in ingresso ai nodi del livello successivo (…)
3. i nodi di output calcolano le uscite della rete
4. conoscendo l'uscita desiderata, viene calcolato l'errore presente su ogni nodo del livello di output
5. si calcolano i $\delta_k = (d_k - o_k) f'(a_k)$ per ogni nodo del livello di output e si applica l'equazione di aggiornamento ai pesi relativi
6. per ogni nodo del penultimo livello si calcolano i $\delta_j = \left[\sum_k \delta_k w_{kj} \right] f'(a_j)$ e si applica l'equazione di aggiornamento ai pesi relativi

Punti in sospeso

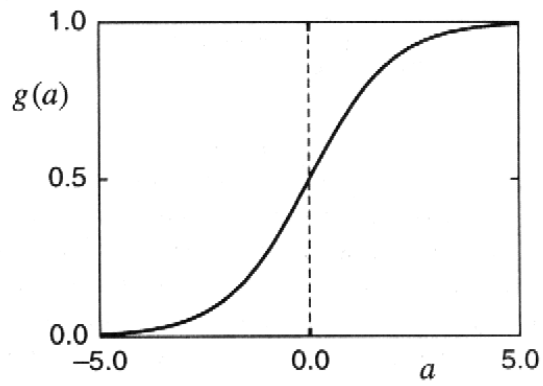
- Qual è la funzione di attivazione?
- Qual è la condizione di fine learning?

Funzione di Attivazione

- La più comune funzione sigmoide usata è la funzione logistica
  - ► f(x) = 1/(1 + e-x)
  - ► Il calcolo della derita è importante nelle reti neurali e la funzione logistica ha una derivata 'comoda' derivative

    f'(x) = f(x)(1 - f(x))

- Altre funzioni sigmoide usate sono
  - ► tangente iperbolica
  - ► arctangent



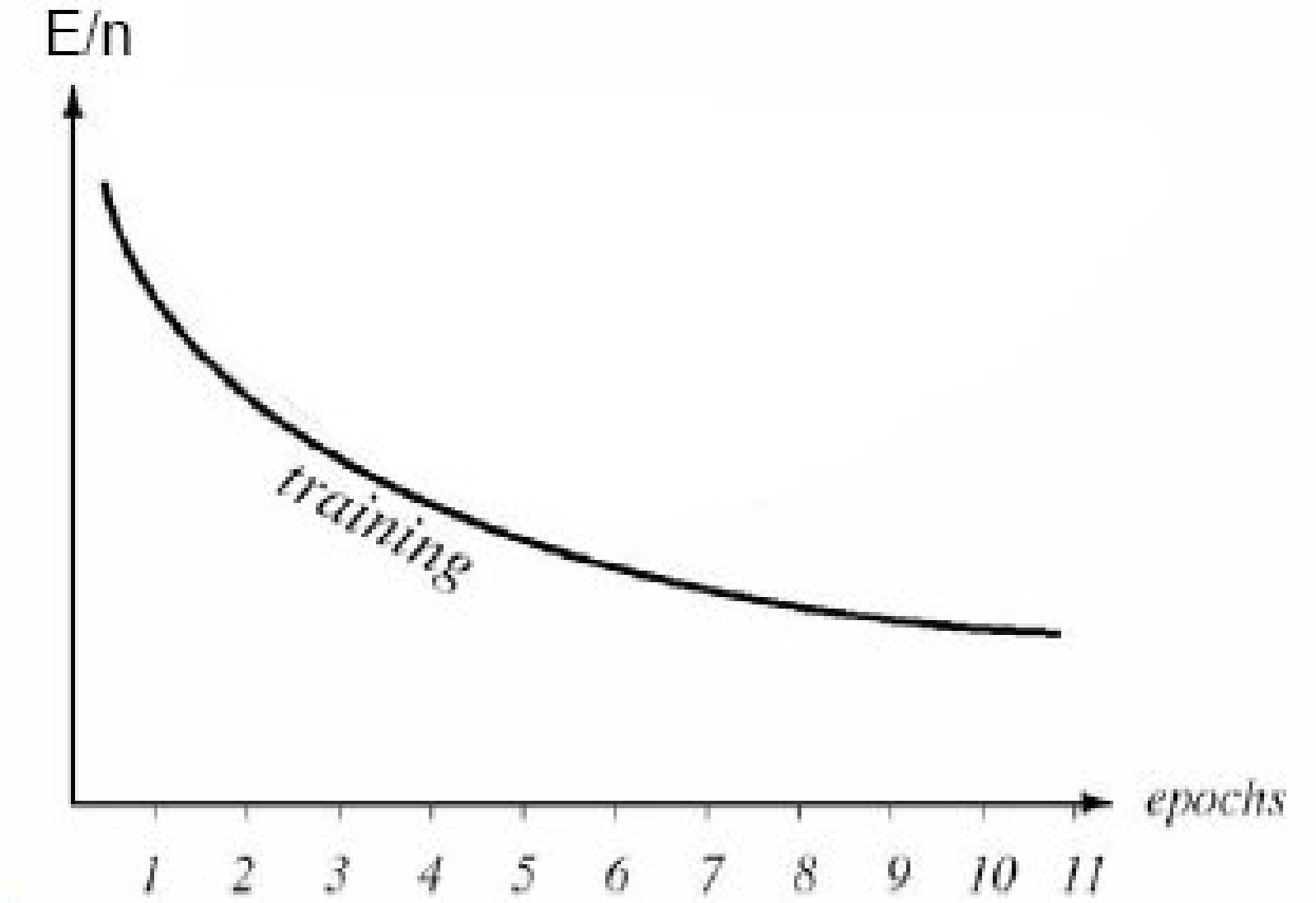
Condizione di termine

- Valore di E
- Numero di epoche
- Andamento del gradiente
- Problemi di generalizzazione
  - validazione

Curve di apprendimento

- All'inizio della fase di training, l'errore sul training set è tipicamente alto
- Con il procedere dell'apprendimeno, l'errore tende a diminuire, raggiungendo alla fine un valore asintotico che dipende da
  - Errore statistico sul training set
  - Dimensioni del training set
  - Numero dei pesi della rete
  - Valore iniziale dei pesi
- L'andamento dell'errore rispetto al numero di epoche realizzato è visualizzato su una curva di apprendimento (learning curve)

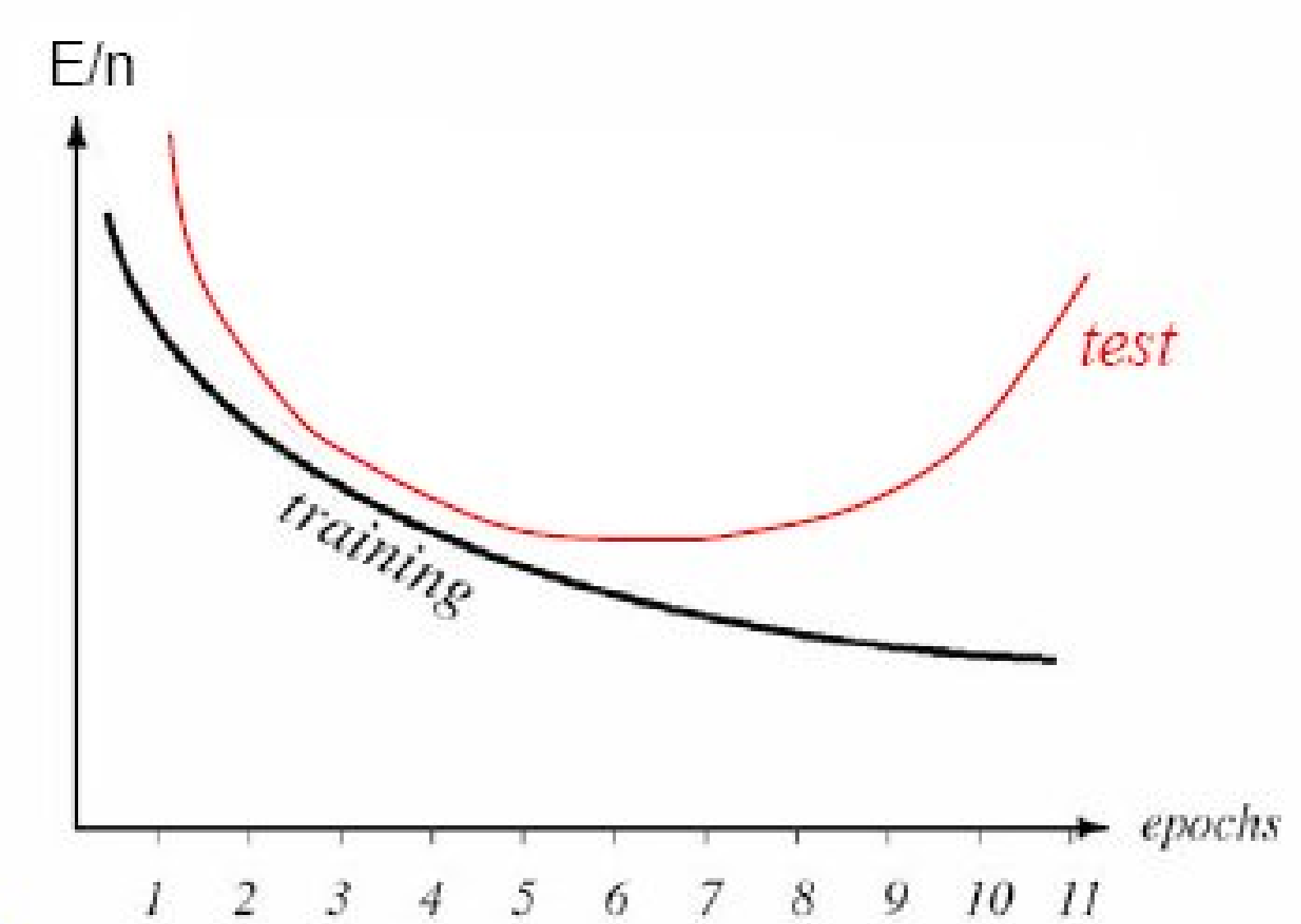
Curve di apprendimento



Curve di apprendimento e condizioni di termine

- Potrebbe essere possibile utilizzare le curve di apprendimento per decidere quando terminare il training?
- A prima vista, potremmo arrestare l'apprendimento quando si è raggiunto un valore soddisfacente dell'errore o quando si raggiunge l'asintoto

Curve di apprendimento



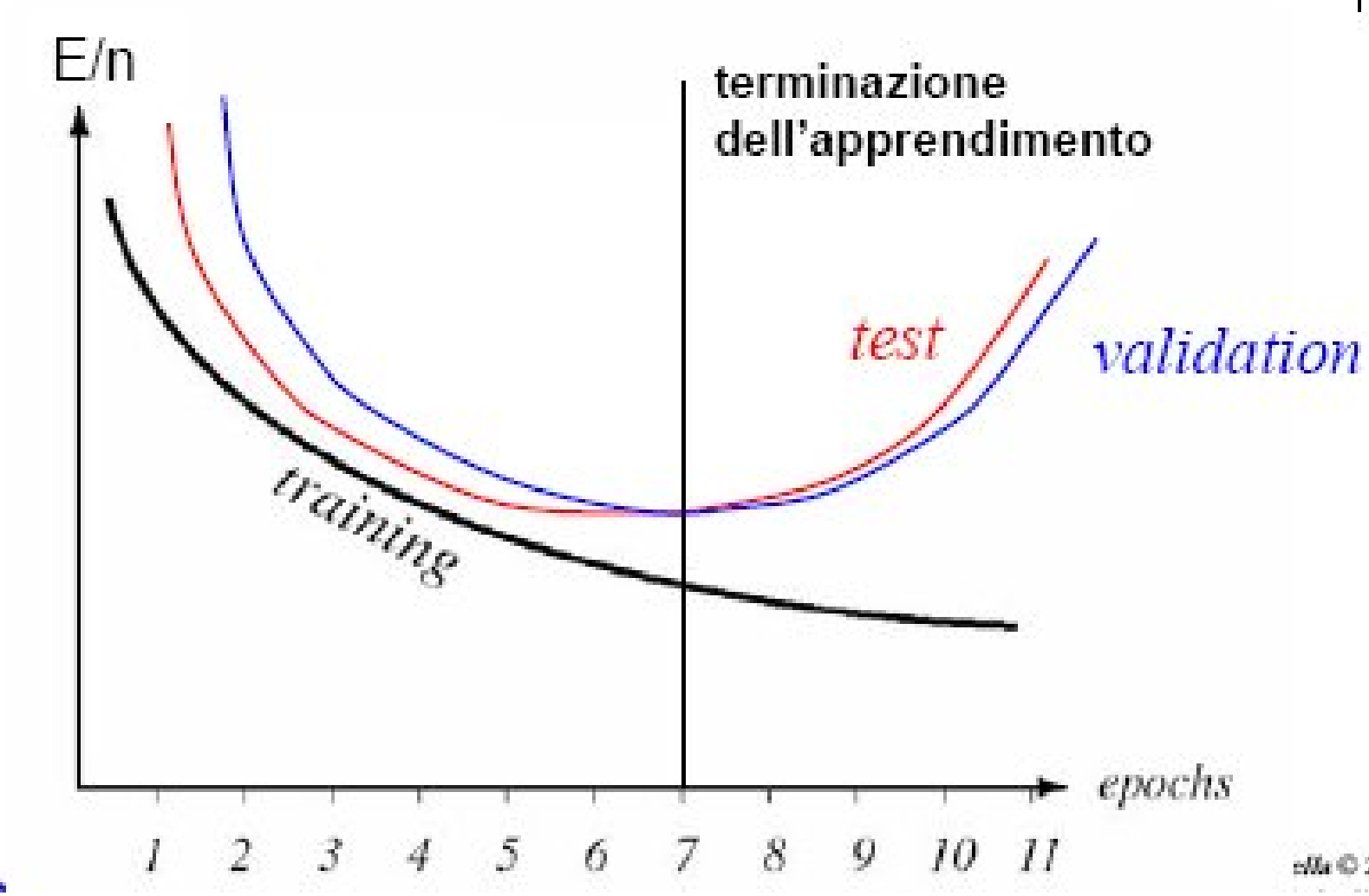
Curve di apprendimento e condizioni di termine

- Al procedere del learning, la rete si specializza sempre meglio sul training set, migliorando l'accuratezza e garantendo una buona generalizzazione
- Da un certo punto, però la rete comincia a specializzarsi troppo sul training set perdendone in generalizzazione

Apprendimento con un insieme di validazione

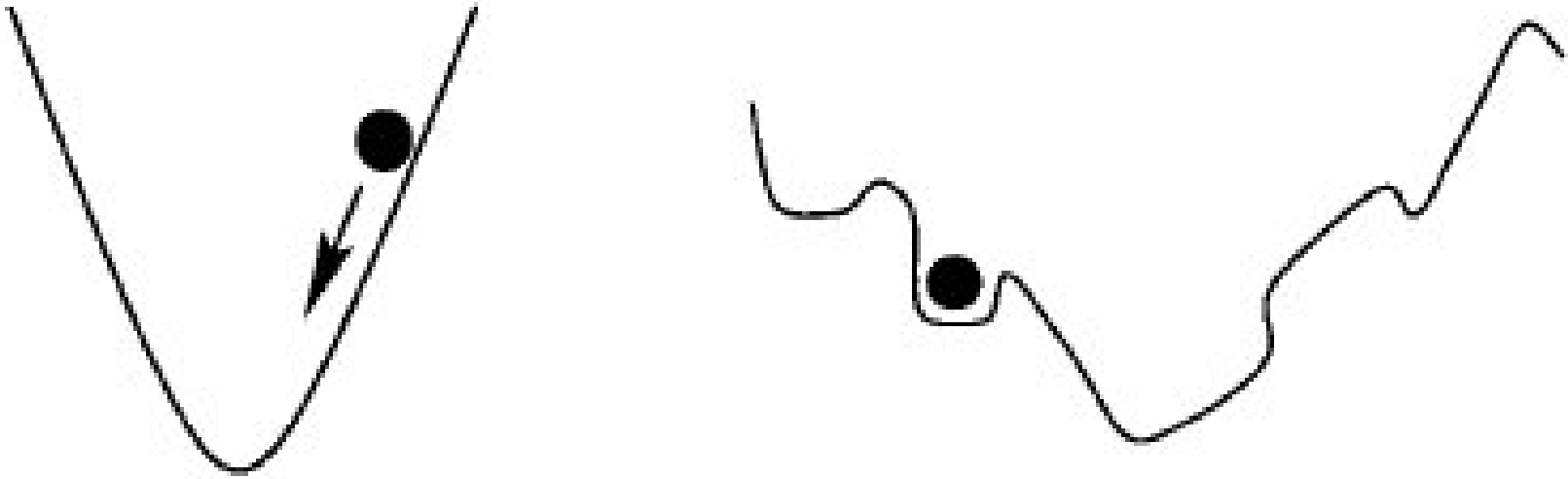
- Per controllare la capacità di generalizzazione che la rete sta acquisendo durante in training si effettua periodicamente la classificazione di un insieme di campioni non appartenenti al training set (insieme di validazione o validation set)
- L'andamento dell'apprendimento è quindi isualizzato da due curve, una valutata sul training set e una sul validation set
- L'arresto dell'apprendimento si puo' quindi realizzare in corrispondenza di un minimo sulla reltiva al validation set

Terminazione dell'apprendimento



Migliorare il training

- Problema dei minimi locali



Migliorare il training

- Tecnica del momento

  All' aggiornamento del peso, viene aggiunta una percentuale dell'aggiornamento precedente

$$\Delta w_{ji}^{n+1} = -\eta \left. \frac{\partial E}{\partial w_{ji}} \right|_{w^n} + \mu \Delta w_{ji}^{n}$$

Altri algoritmi di training

- Gradiente coniugato
- Newton
- quasi – Newton
- Marquardt - Levenberg

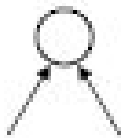
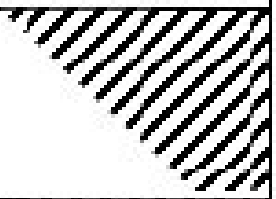
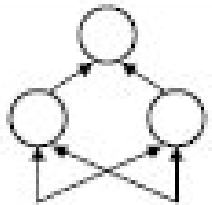
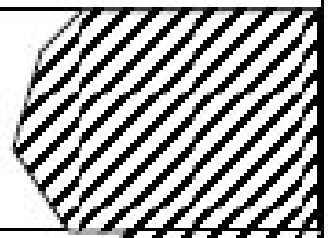
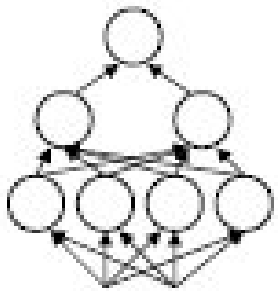
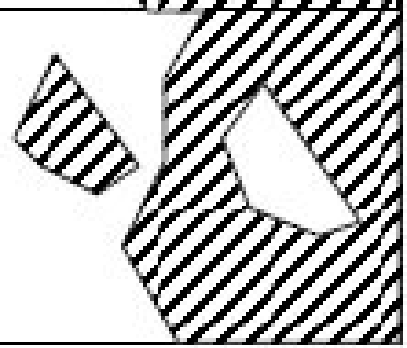
Normalizzazione degli ingressi

- E' necessaria per rendere omogenei i vari ingressi ed evitare che ingressi di valore maggiore influiscano più di ingressi a valore minore.
- Una tipica normalizzazione consiste nel trasformare i valori degli ingressi in modo da rendere nulla la media e unitaria la varianza:

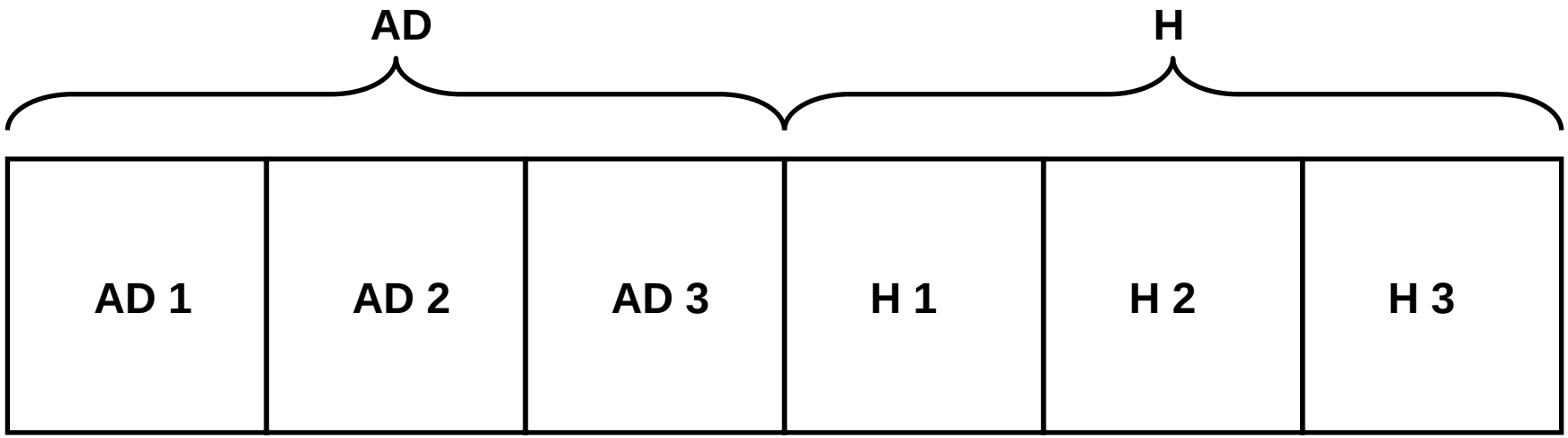
$$\langle x_1, x_2, \ldots, x_n \rangle \rightarrow \langle x'_1, x'_2, \ldots, x'_n \rangle$$

dove: $$x'_i = \frac{x_i - \overline{x}_i}{\sigma_{x_i}}$$

Regioni di decisione delle reti neurali

Struttura	Regioni di decisione	Forma generale
	Semispazi delimitati da iperpiani	
	Regioni convesse	
	Regioni di forma arbitraria	

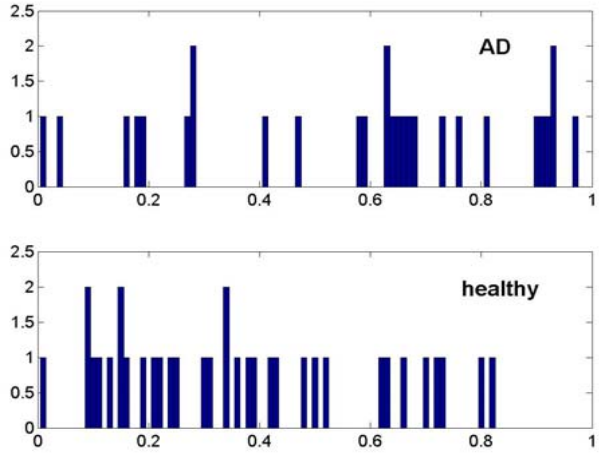
Cross validation

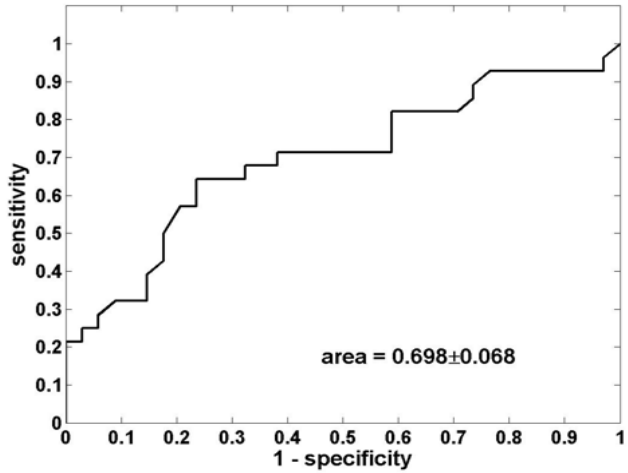


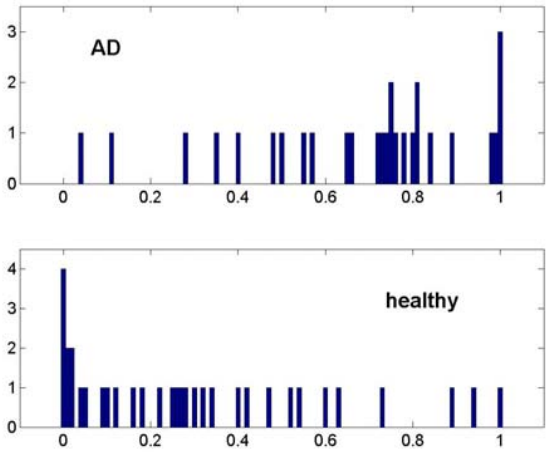
- TRAINING: AD 1 + H 1
- TEST: AD 2 + H2
- VALIDATION: AD 3 + H 3

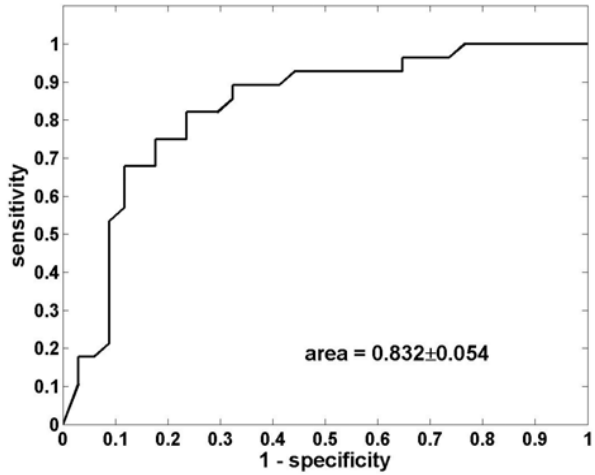
… and cyclical permutations

Neural outputs and ROC curves (1)









MEAN AREA = 0.792 ± 0.008 (20 classifications)

Where Do The Weights Come From?

- There are two main types of training
  - ► Supervised Training
  - ■ Supplies the neural network with inputs and the desired outputs
  - ■ Response of the network to the inputs is measured
    - The weights are modified to reduce the difference between the actual and desired outputs
  - ► Unsupervised Training
  - ■ Only supplies inputs
  - ■ The neural network adjusts its own weights so that similar inputs cause similar outputs
    - – The network identifies the patterns and differences in the inputs without any external assistance

Where Do The Weights Come From?

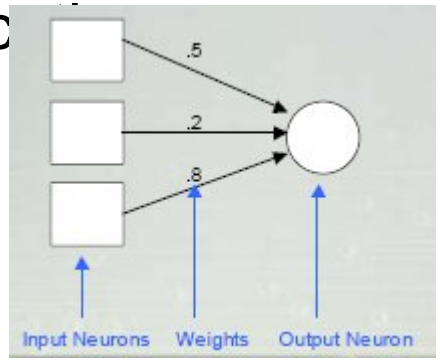
The weights in a neural network are the most important factor in determining its function

- Training is the act of presenting the network with some sample data and modifying the weights tobetter approximate the desired function

- ● Epoch
  ■ One iteration through the process of providing the network with an input and updating the network's weights
  ■ Typically many epochs are required to train the neural network

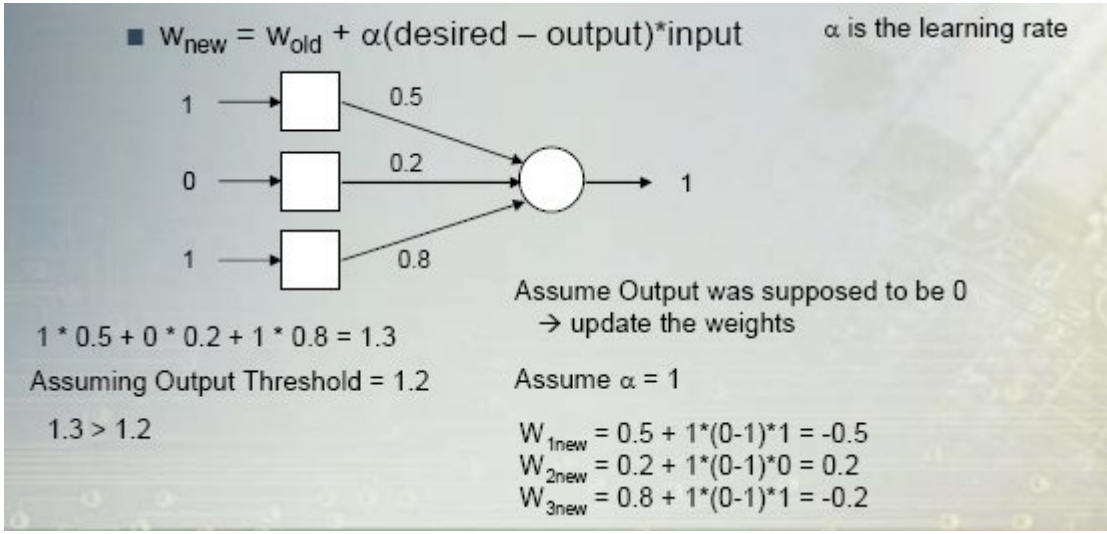
Perceptrons

- First neural network with the ability to learn
- ● Made up of only input neurons and output neurons
- ● Input neurons typically have two states: ON and OFF
- ● Output neurons use a simple threshold activation function
- ● In basic form, can only solve linear problems
- ► Limited applic



How Do Perceptrons Learn?

- Uses supervised training
- ● If the output is not correct, the weights are
- adjusted according to the formula:

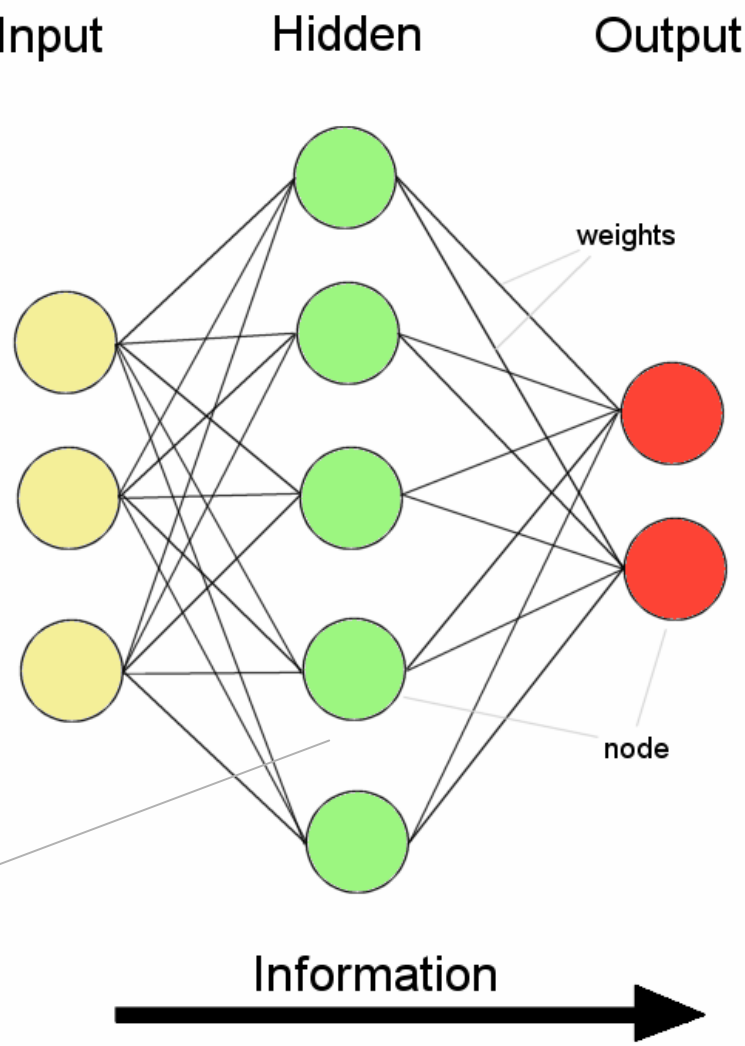


Multilayer Feedforward Networks

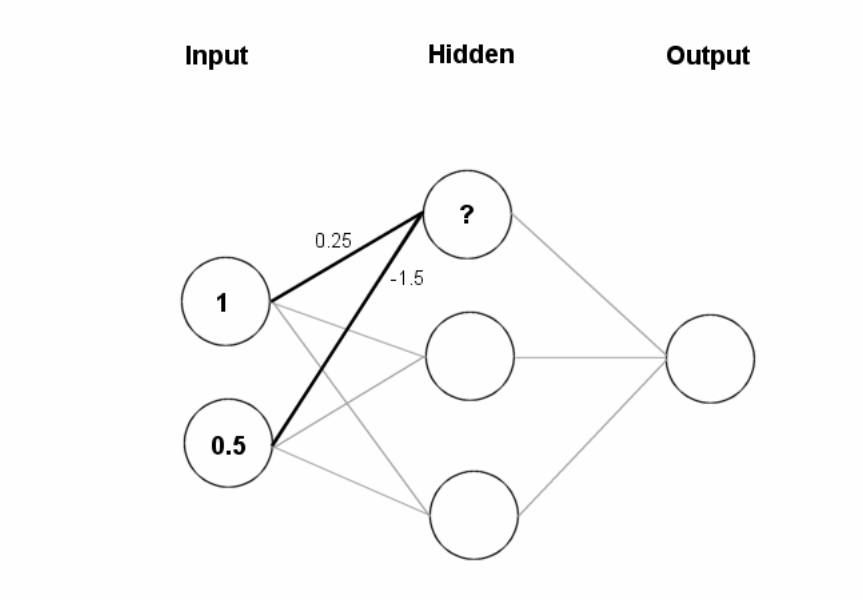
- Most common neural network
- An extension of the perceptron
  - ► Multiple layers
    - – The addition of one or more “hidden” layers in between the input and output layers
  - ► Activation function is not simply a threshold
  - - Usually a sigmoid function
- ► A general function approximator
  - - Not limited to linear problems
- Information flows in one direction
  - ► The outputs of one layer act as inputs to the next layer

Feed-forward nets

- Information flow is unidirectional
  - Data is presented to *Input layer*
  - Passed on to *Hidden Layer*
  - Passed on to *Output layer*
- Information is distributed
- Information processing is parallel



- Feeding data through the net:



$(1 \times 0.25) + (0.5 \times (-1.5)) = 0.25 + (-0.75) = \mathbf{-0.5}$

Squashing: $\frac{1}{1+e^{0.5}} = 0.3775$

- Data is presented to the network in the form of activations in the input layer

- Examples
  - Pixel intensity (for pictures)
  - Molecule concentrations (for artificial nose)
  - Share prices (for stock market prediction)

- Data usually requires preprocessing
  - Analogous to senses in biology

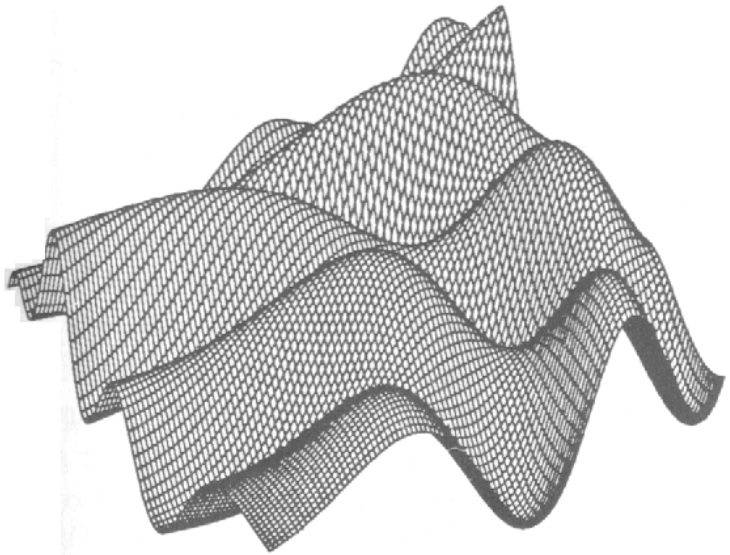
- How to represent more abstract data, e.g. a name?
  - Choose a pattern, e.g.
    - 0-0-1 for “Chris”
    - 0-1-0 for “Becky”

- Weight settings determine the behaviour of a network

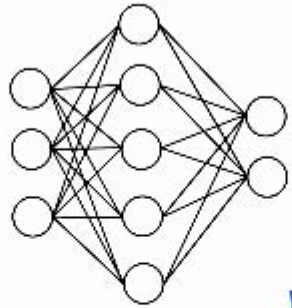
  → How can we find the right weights?

Training the Network - Learning

- Backpropagation
  - Requires training set (input / output pairs)
  - Starts with small random weights
  - Error is used to adjust weights (supervised learning)
  - → Gradient descent on error landscape



1.

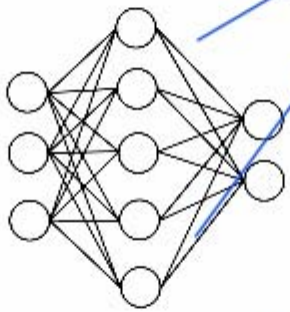


Wallace

Wallace - Darwin (calculate error)

adjust weights

2.

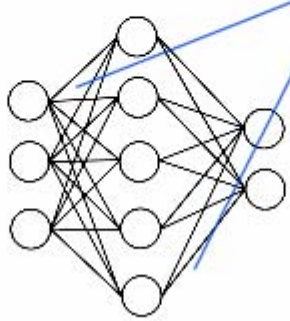


Wallace

Wallace - Darwin (calculate error)

adjust weights

3.



Darwin

- **Advantages**
  - It works!
  - Relatively fast

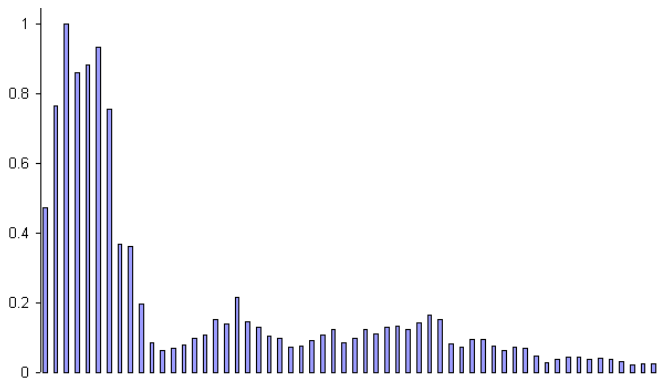
- **Downsides**
  - Requires a training set
  - Can be slow
  - Probably not biologically realistic

- **Alternatives to Backpropagation**
  - Hebbian learning
    - Not successful in feed-forward nets
  - Reinforcement learning
    - Only limited success
  - Artificial evolution
    - More general, but can be even slower than backprop

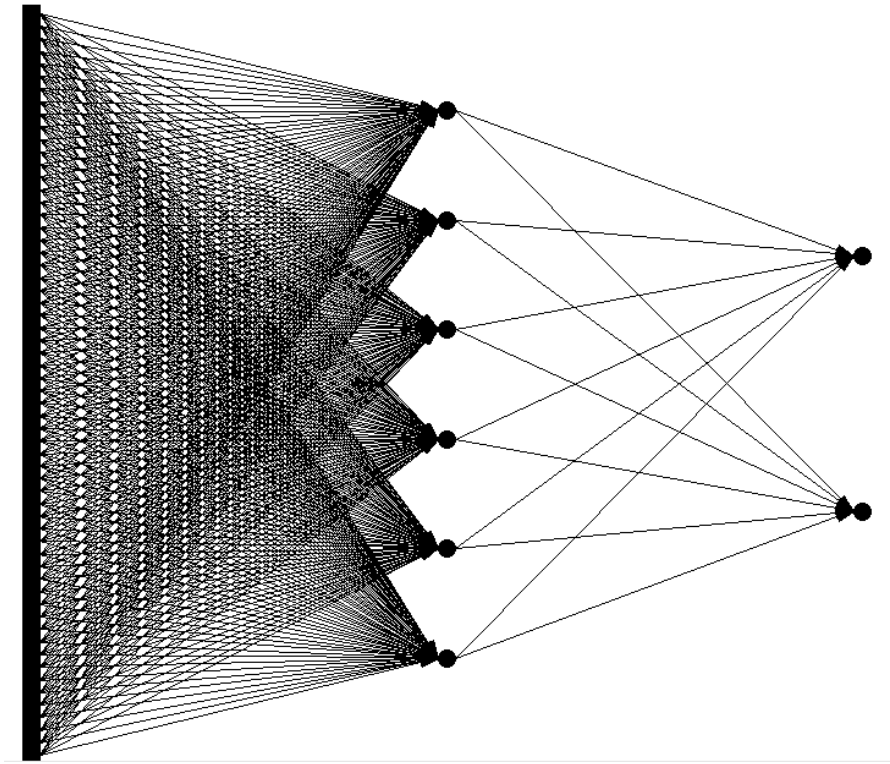
Example: Voice Recognition

- Task: Learn to discriminate between two different voices saying “Hello”

- Data
  - Sources
    - Steve Simpson
    - David Raubenheimer
  - Format
    - Frequency distribution (60 bins)
    - Analogy: cochlea

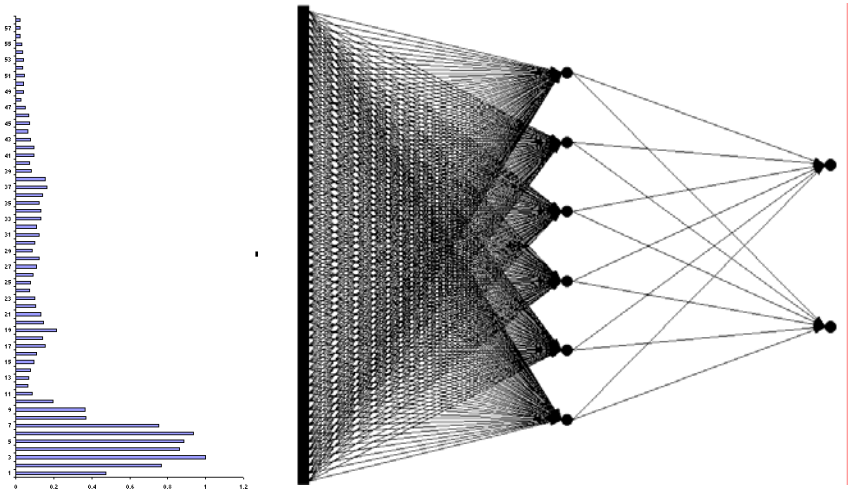


- Network architecture
  - Feed forward network
    - 60 input (one for each frequency bin)
    - 6 hidden
    - 2 output (0-1 for “Steve”, 1-0 for “David”)

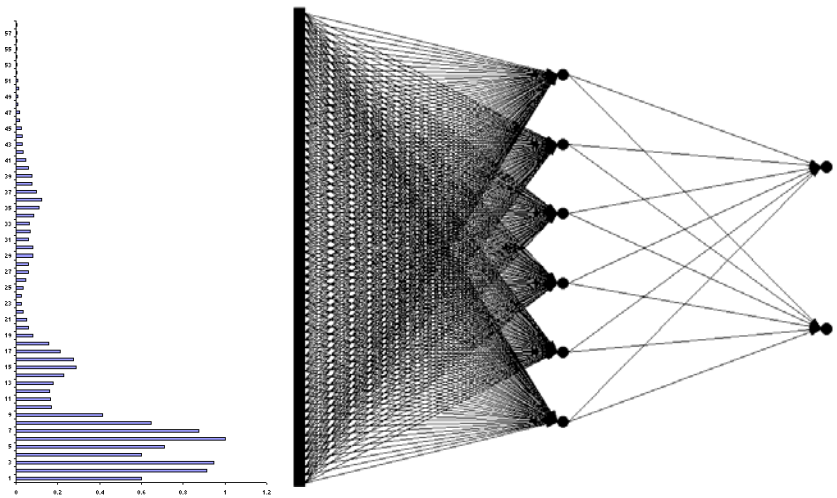


- Presenting the data

Steve

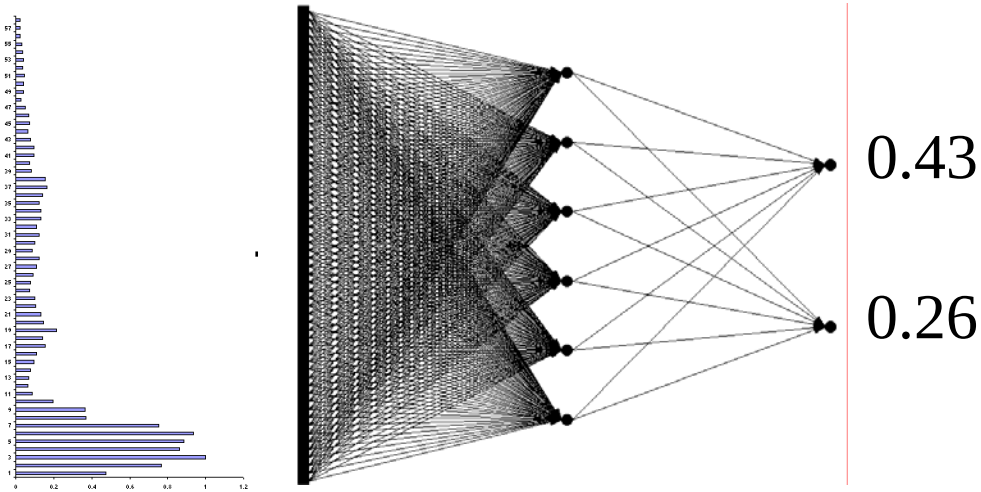


David

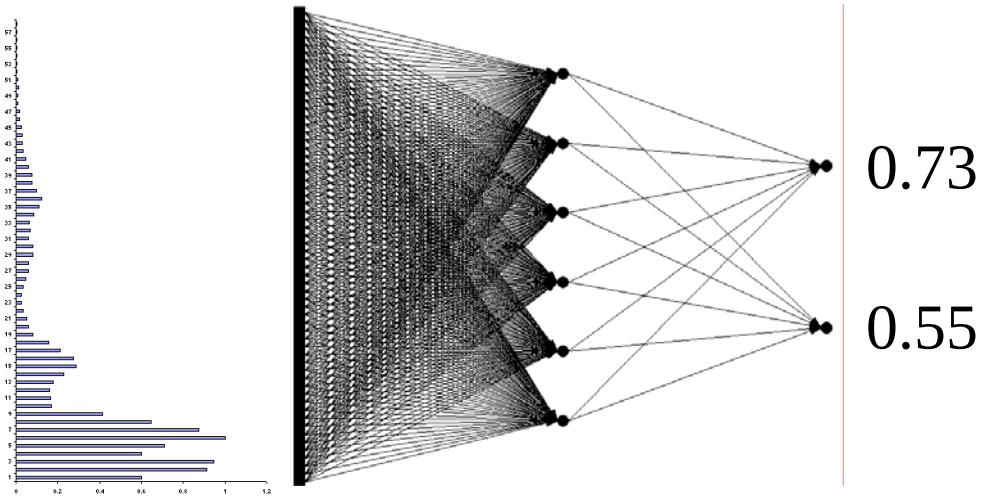


- Presenting the data (untrained network)

Steve

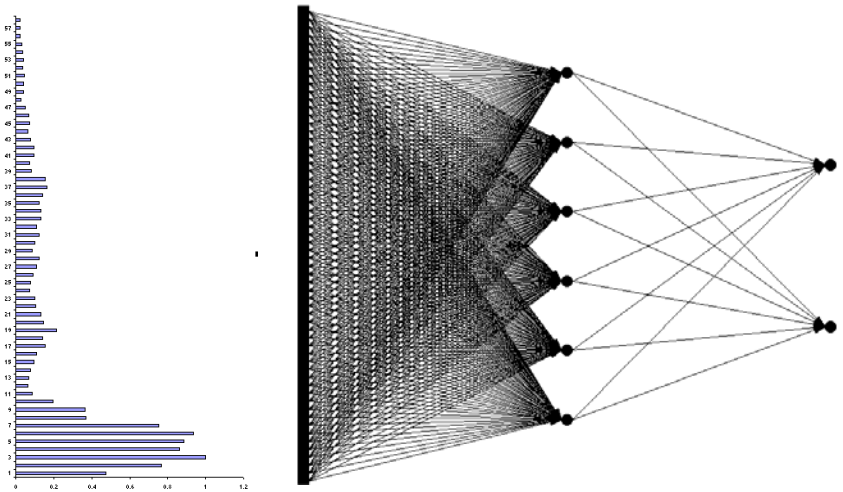


David



- Calculate error

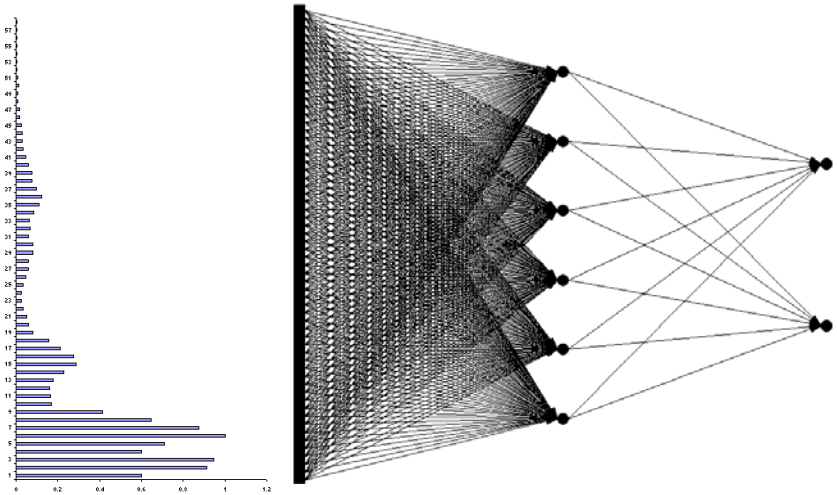
Steve



$0.43 - 0 = 0.43$

$0.26 - 1 = 0.74$

David

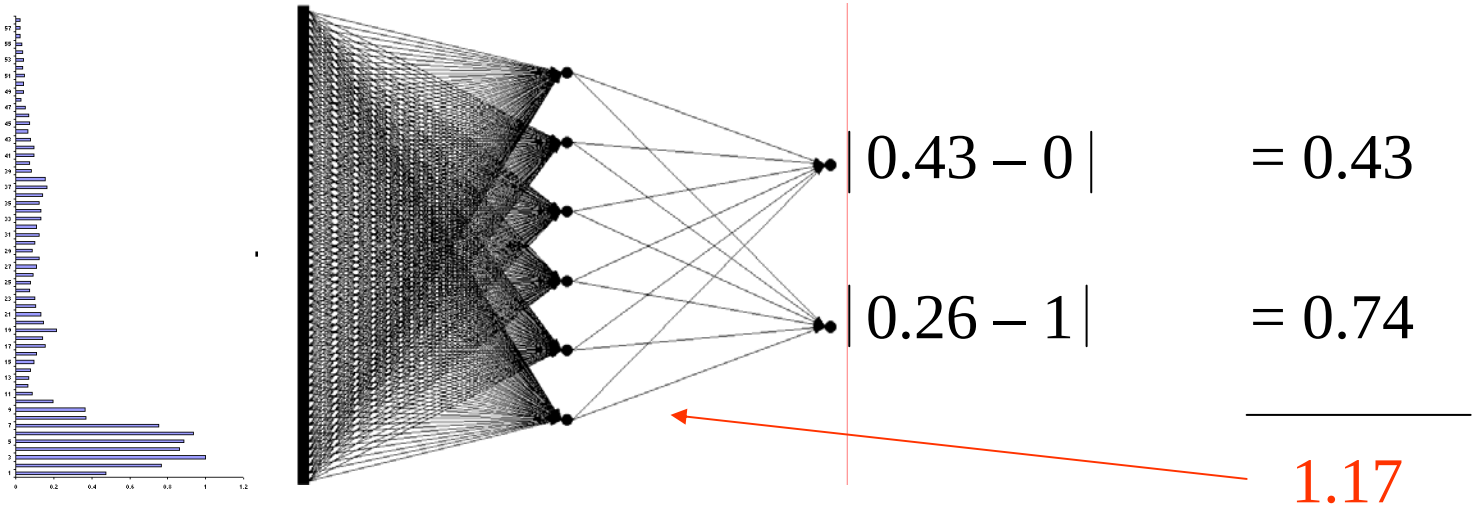


$0.73 - 1 = 0.27$

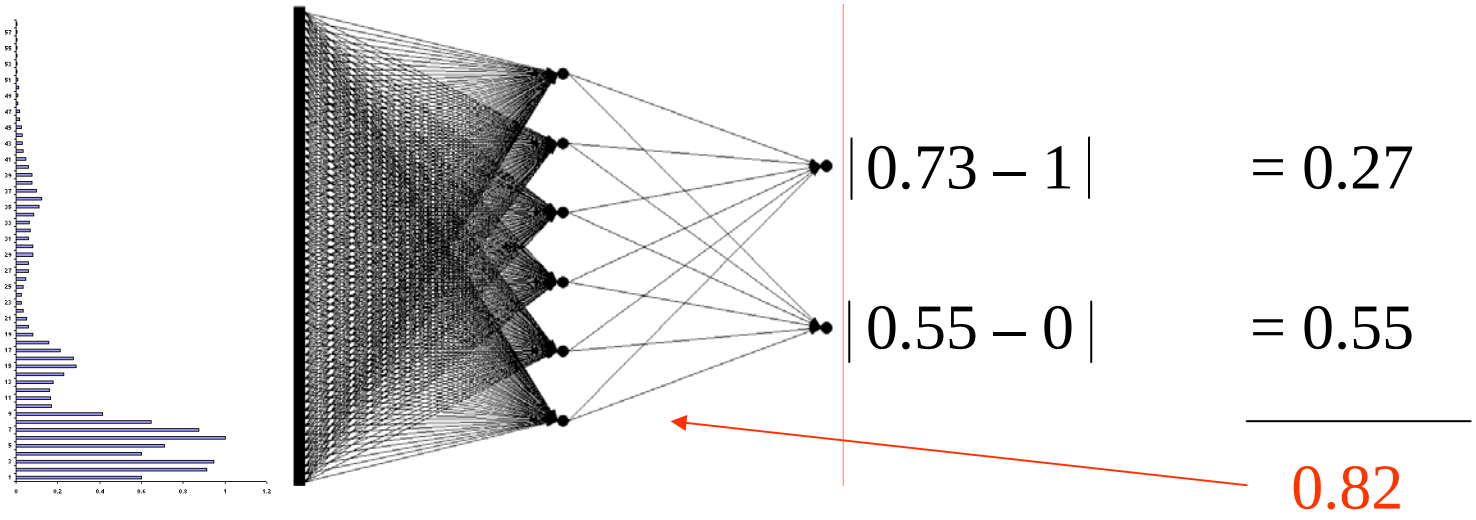
$0.55 - 0 = 0.55$

- Backprop error and adjust weights

Steve

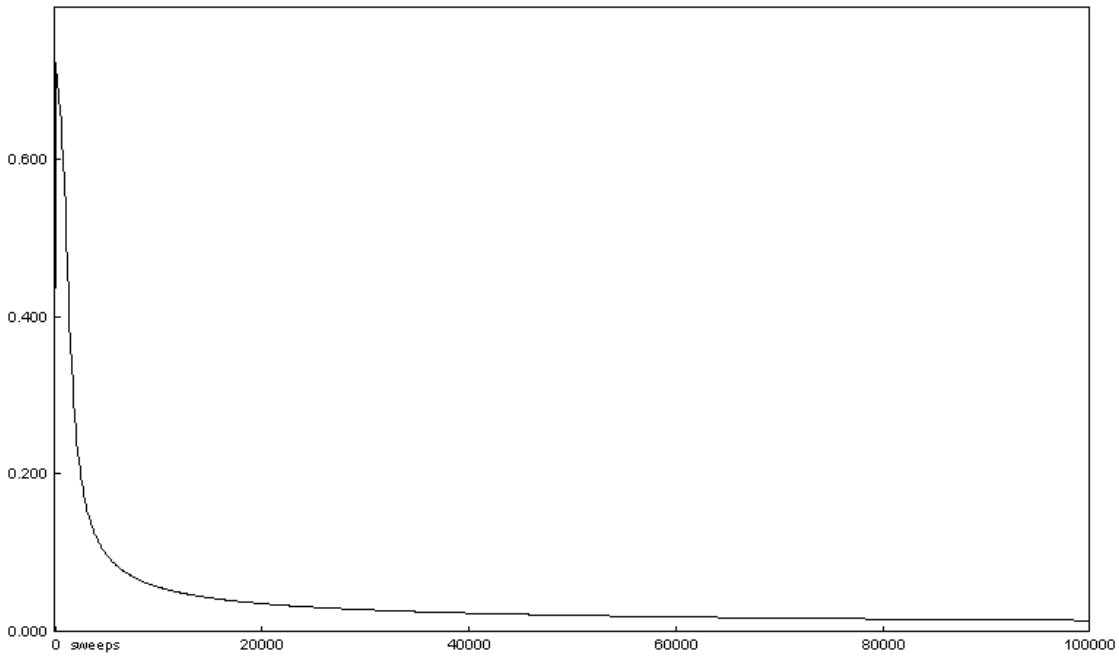


David



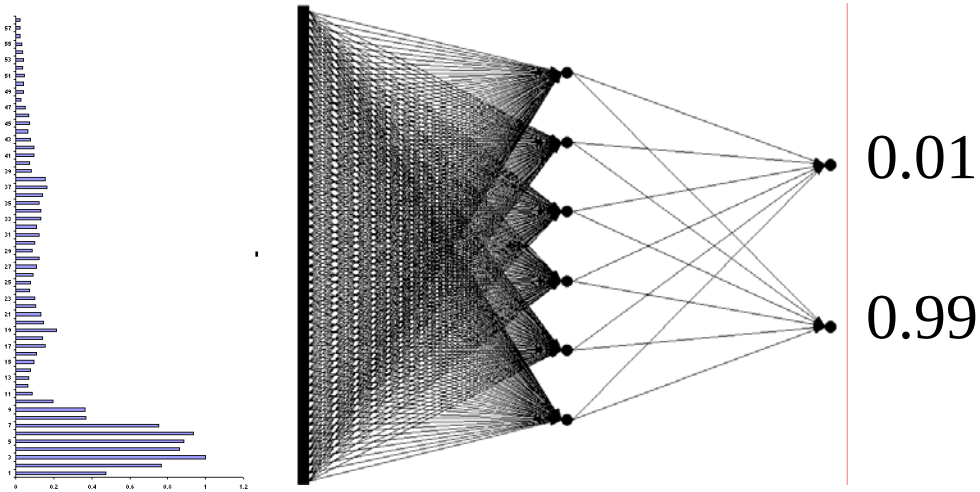
- Repeat process (sweep) for all training pairs
  - Present data
  - Calculate error
  - Backpropagate error
  - Adjust weights

- Repeat process multiple times

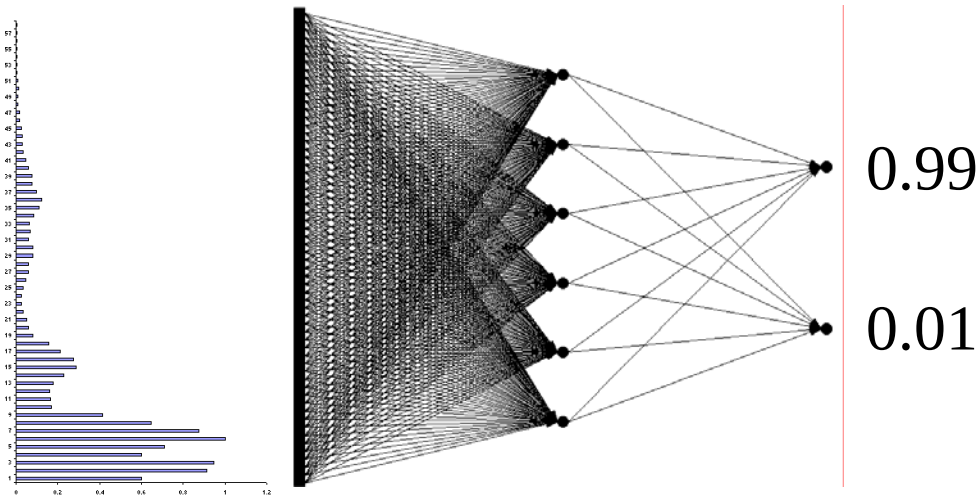


- Presenting the data (trained network)

Steve



David

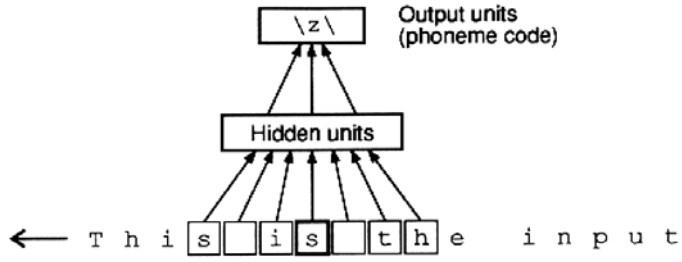


- Results – Voice Recognition
  - Performance of trained network
    - Discrimination accuracy between known “Hello”s
      - 100%
    - Discrimination accuracy between new “Hello”’s
      - 100%
- Demo

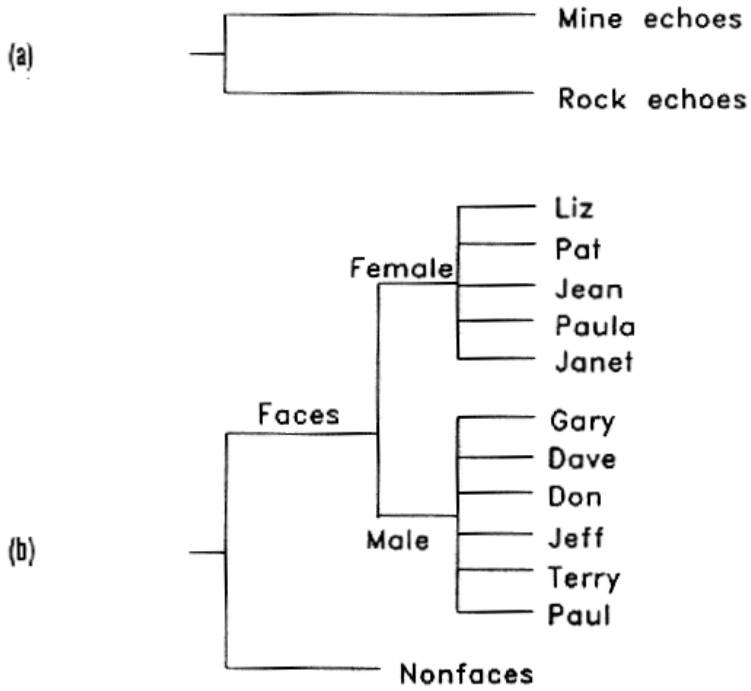
- Results – Voice Recognition (ctnd.)
  - Network has learnt to generalise from original data
  - Networks with different weight settings can have same functionality
  - Trained networks ‘concentrate’ on lower frequencies
  - Network is robust against non-functioning nodes

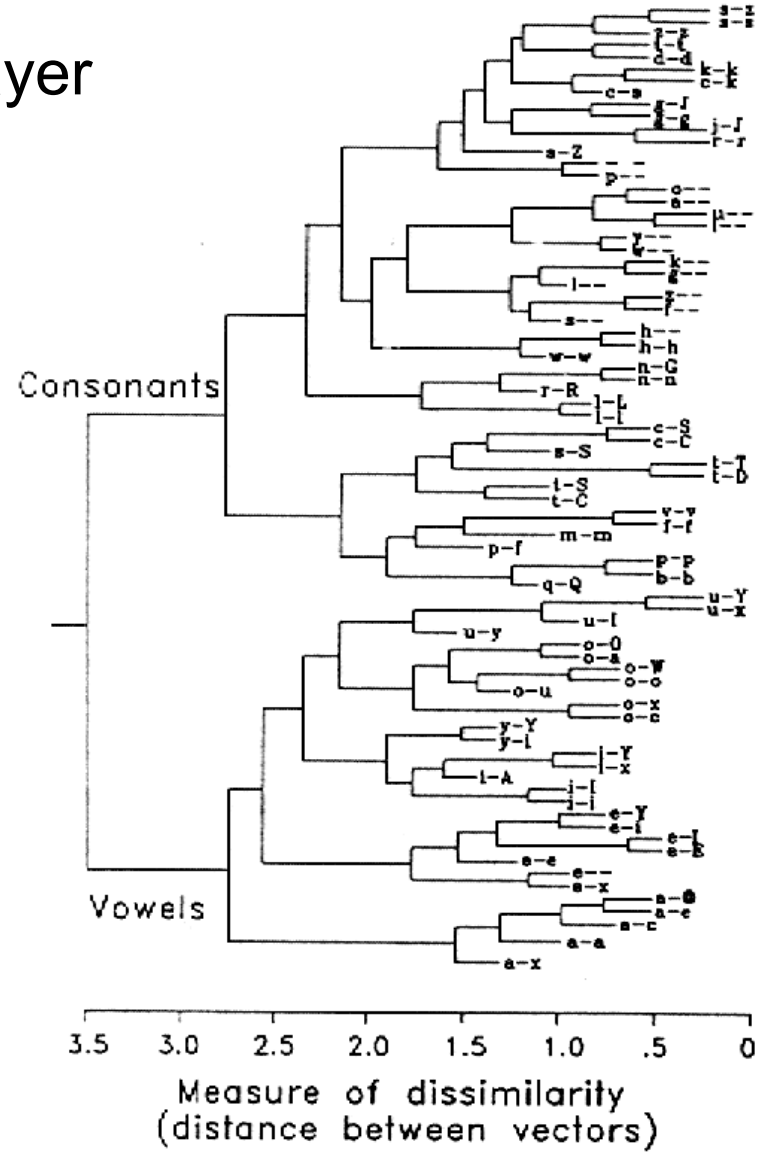
Applications of Feed-forward nets

- Pattern recognition
  - Character recognition
  - Face Recognition
- Sonar mine/rock recognition (Gorman & Sejnowksi, 1988)
- Navigation of a car (Pomerleau, 1989)
- Stock-market prediction
- Pronunciation (NETtalk)
  (Sejnowksi & Rosenberg, 1987)



Cluster analysis of hidden layer





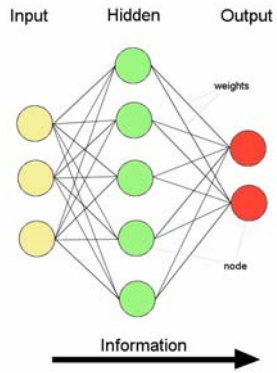
FFNs as Biological Modelling Tools

- Signalling / Sexual Selection
  - Enquist & Arak (1994)
    - Preference for symmetry not selection for ‘good genes’, but instead arises through the need to recognise objects irrespective of their orientation
  - Johnstone (1994)
    - Exaggerated, symmetric ornaments facilitate mate recognition

(but see Dawkins & Guilford, 1995)

Recurrent Networks

- Feed forward networks:
  - Information only flows one way
  - One input pattern produces one output
  - No sense of time (or memory of previous state)

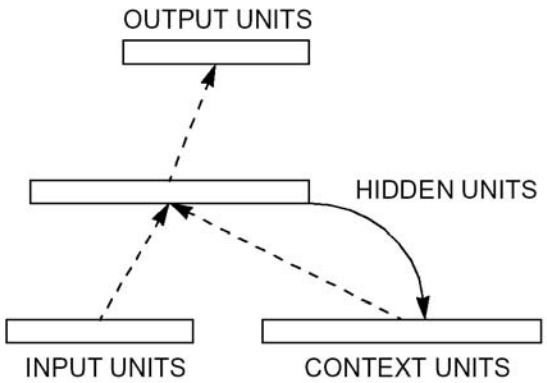


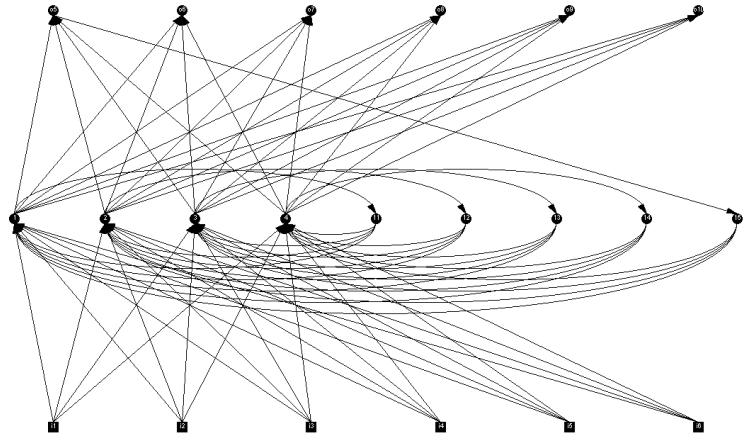
- Recurrency
  - Nodes connect back to other nodes or themselves
  - Information flow is multidirectional
  - Sense of time and memory of previous state(s)

- Biological nervous systems show high levels of recurrency (but feed-forward structures exists too)

Elman Nets

- *Elman nets* are feed forward networks with partial recurrency



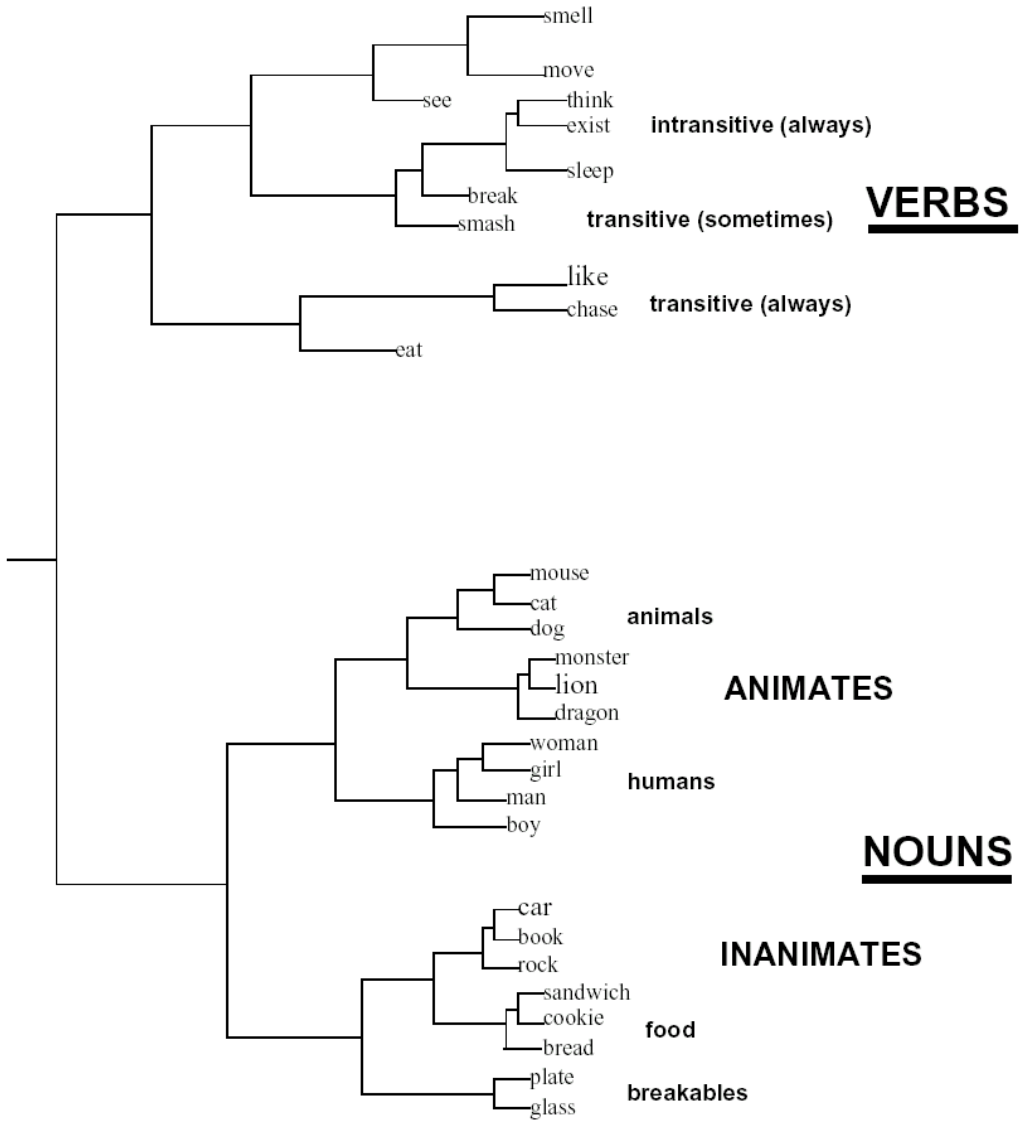


- Unlike feed forward nets, Elman nets have a *memory* or *sense of time*

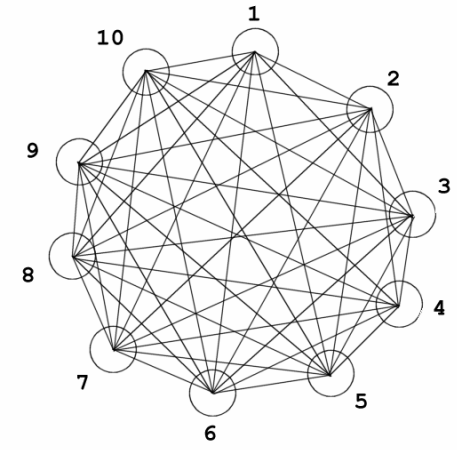
Classic experiment on language acquisition and processing (Elman, 1990)

- **Task**
  - Elman net to predict successive words in sentences.
- **Data**
  - Suite of sentences, e.g.
    - "The boy catches the ball."
    - "The girl eats an apple."
  - Words are input one at a time
- **Representation**
  - Binary representation for each word, e.g.
    - 0-1-0-0-0 for "girl"
- **Training method**
  - Backpropagation

- Internal representation of words



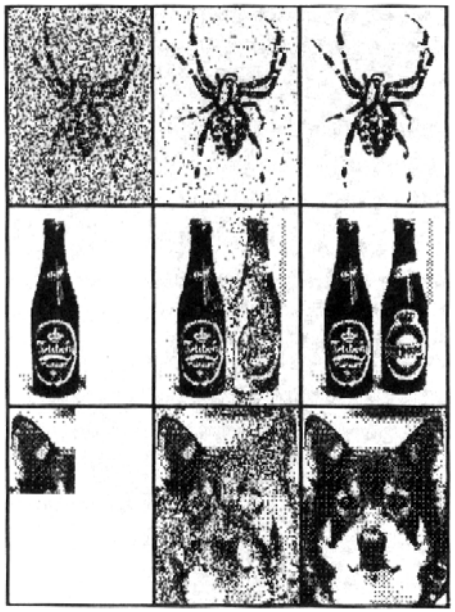
Hopfield Networks



- Sub-type of recurrent neural nets
  - Fully recurrent
  - Weights are symmetric
  - Nodes can only be *on* or *off*
  - Random updating

- Learning: **Hebb rule** (cells that fire together wire together)
  - Biological equivalent to LTP and LTD

- Can recall a memory, if presented with a corrupt or incomplete version



→ **auto-associative** or **content-addressable memory**

Task: store images with resolution of 20x20 pixels

→ Hopfield net with 400 nodes

Memorise:

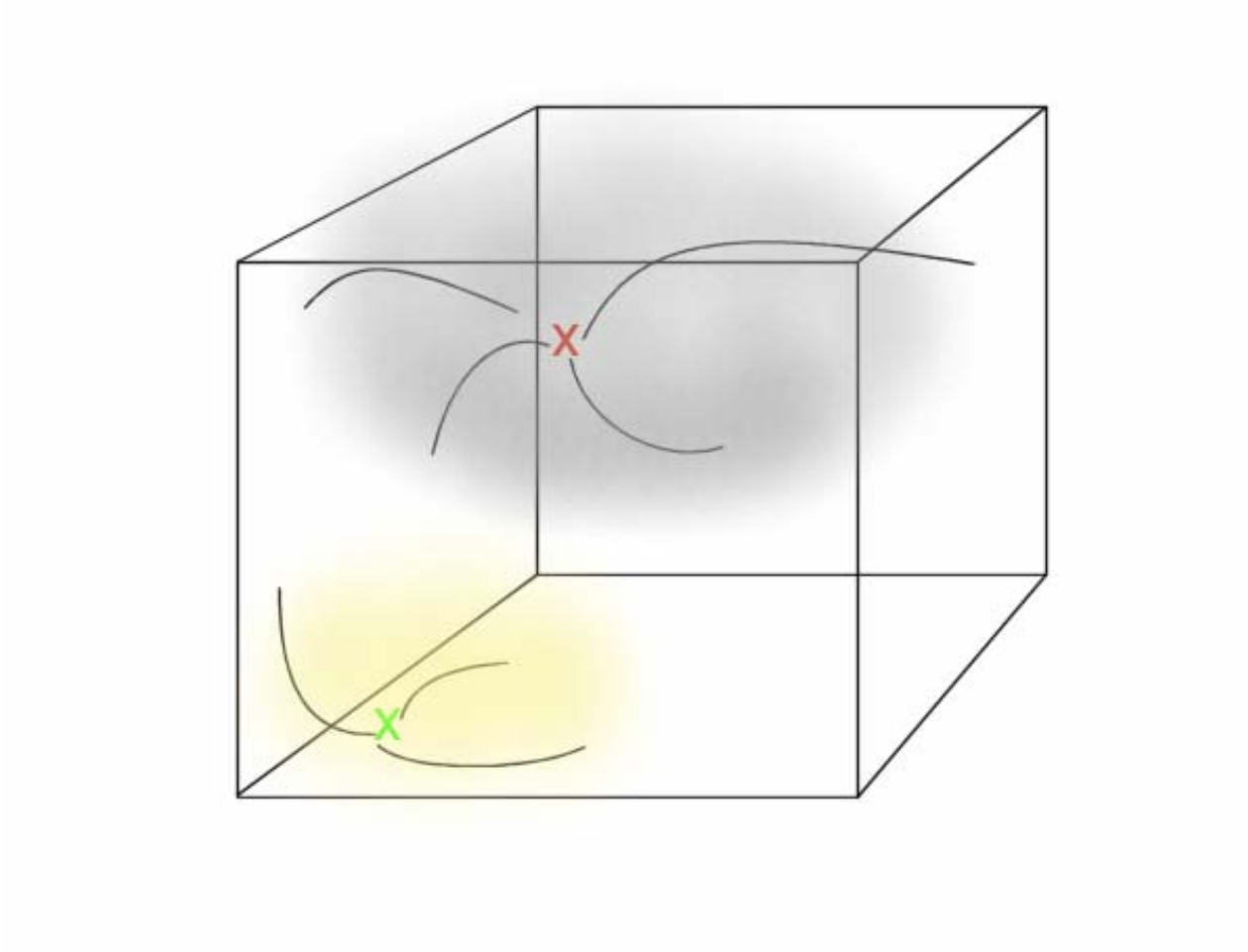
1. Present image
2. Apply Hebb rule (*cells that fire together, wire together*)
   - Increase weight between two nodes if both have same activity, otherwise decrease
3. Go to 1

Recall:

1. Present incomplete pattern
2. Pick random node, update
3. Go to 2 until settled

DEMO

- Memories are attractors in state space



Catastrophic forgetting

- *Problem*: memorising new patterns corrupts the memory of older ones
  → Old memories cannot be recalled, or spurious memories arise

- *Solution*: allow Hopfield net to **sleep**

- Two approaches (both using randomness):

  - **Unlearning** (Hopfield, 1986)
    - Recall old memories by random stimulation, but use an *inverse* Hebb rule

    →'Makes room' for new memories (basins of attraction shrink)

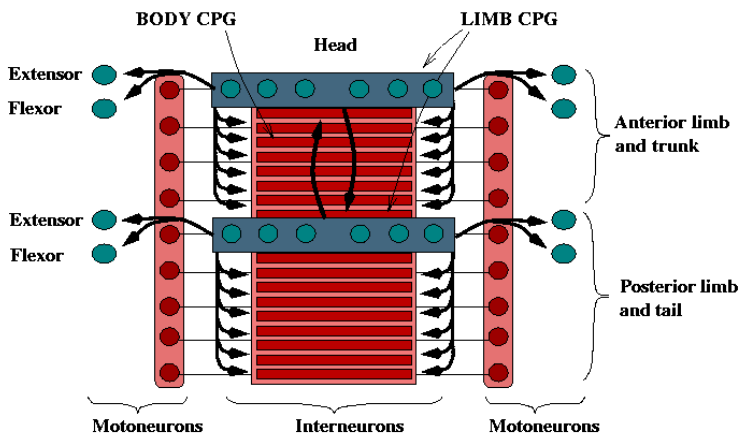
  - **Pseudorehearsal** (Robins, 1995)
    - While learning new memories, recall old memories by random stimulation
    - Use *standard* Hebb rule on new and old memories

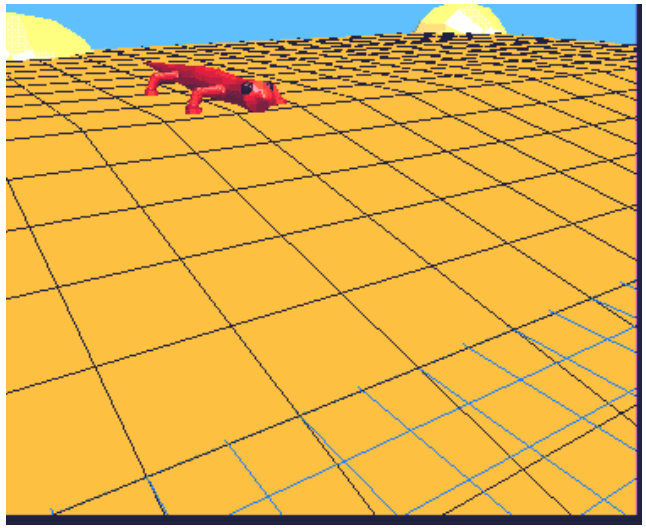
    → Restructure memory
    - Needs short-term + long term memory
    - Mammals: hippocampus plays back new memories to neo-cortex, which is randomly stimulated at the same time

RNNs as Central Pattern Generators

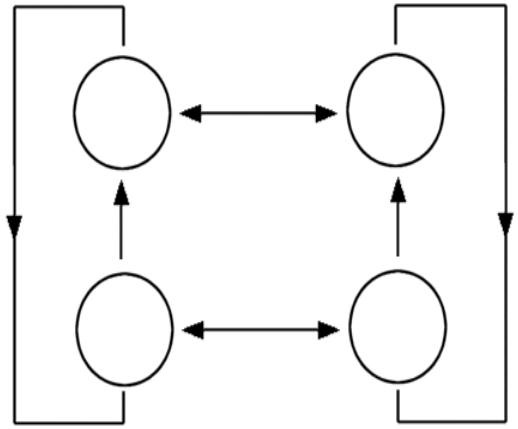
- **CPGs:** group of neurones creating rhythmic muscle activity for locomotion, heart-beat etc.
- Identified in several invertebrates and vertebrates
- Hard to study

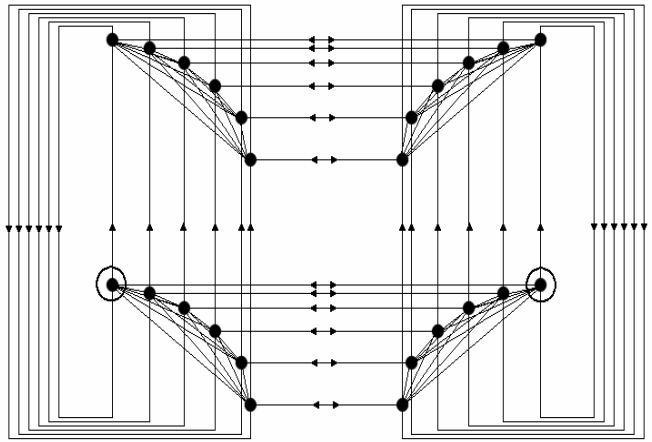
- → Computer modelling
  - E.g. lamprey swimming (Ijspeert *et al.*, 1998)

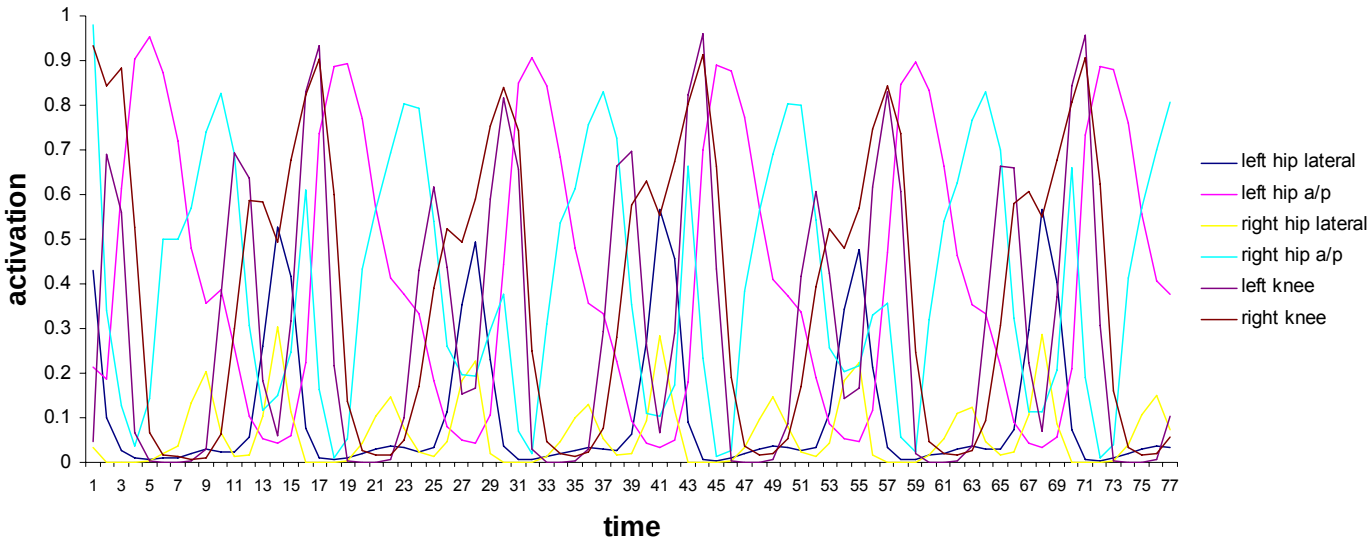


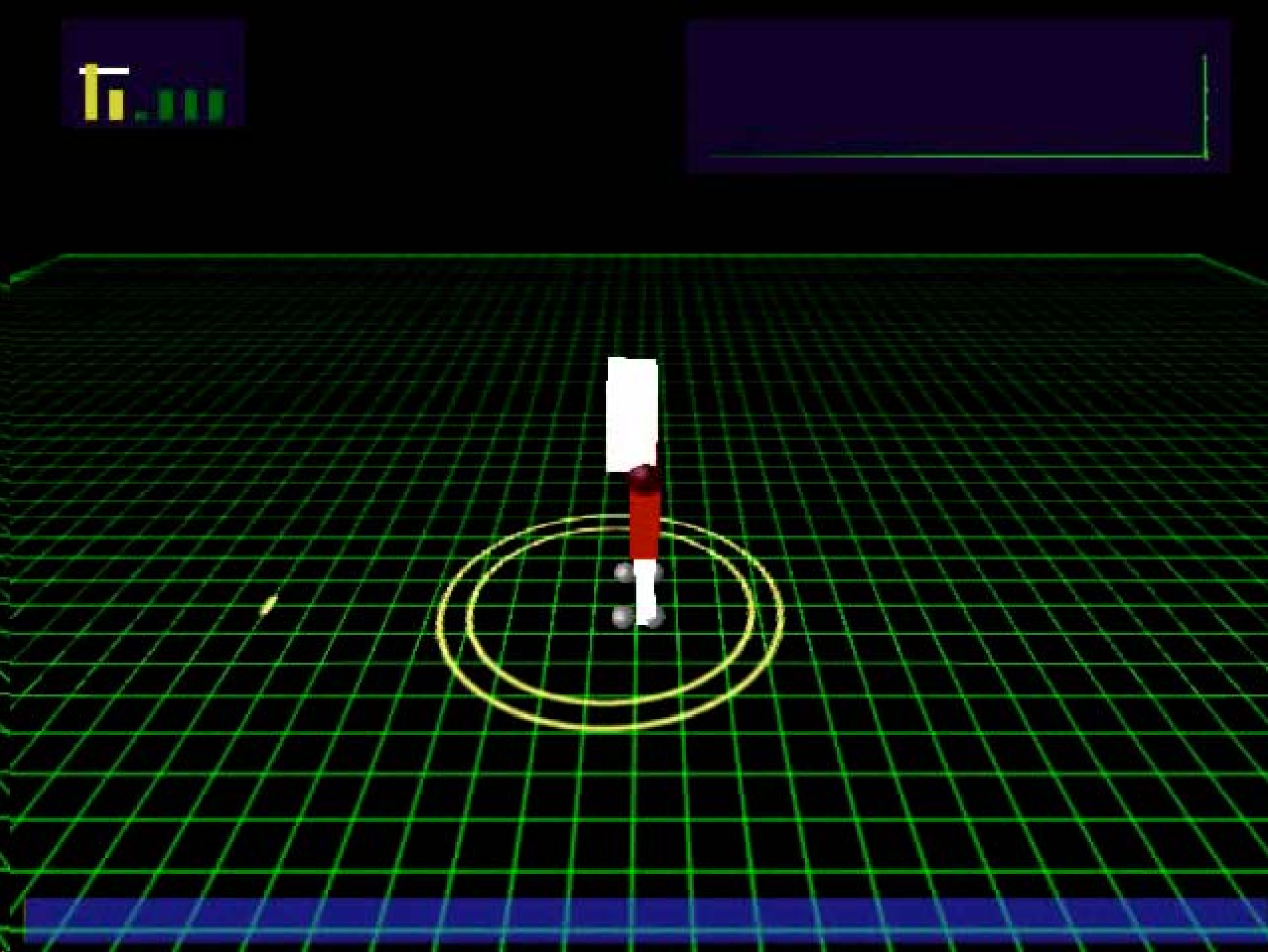


- Evolution of Bipedal Walking (Reil & Husbands, 2001)

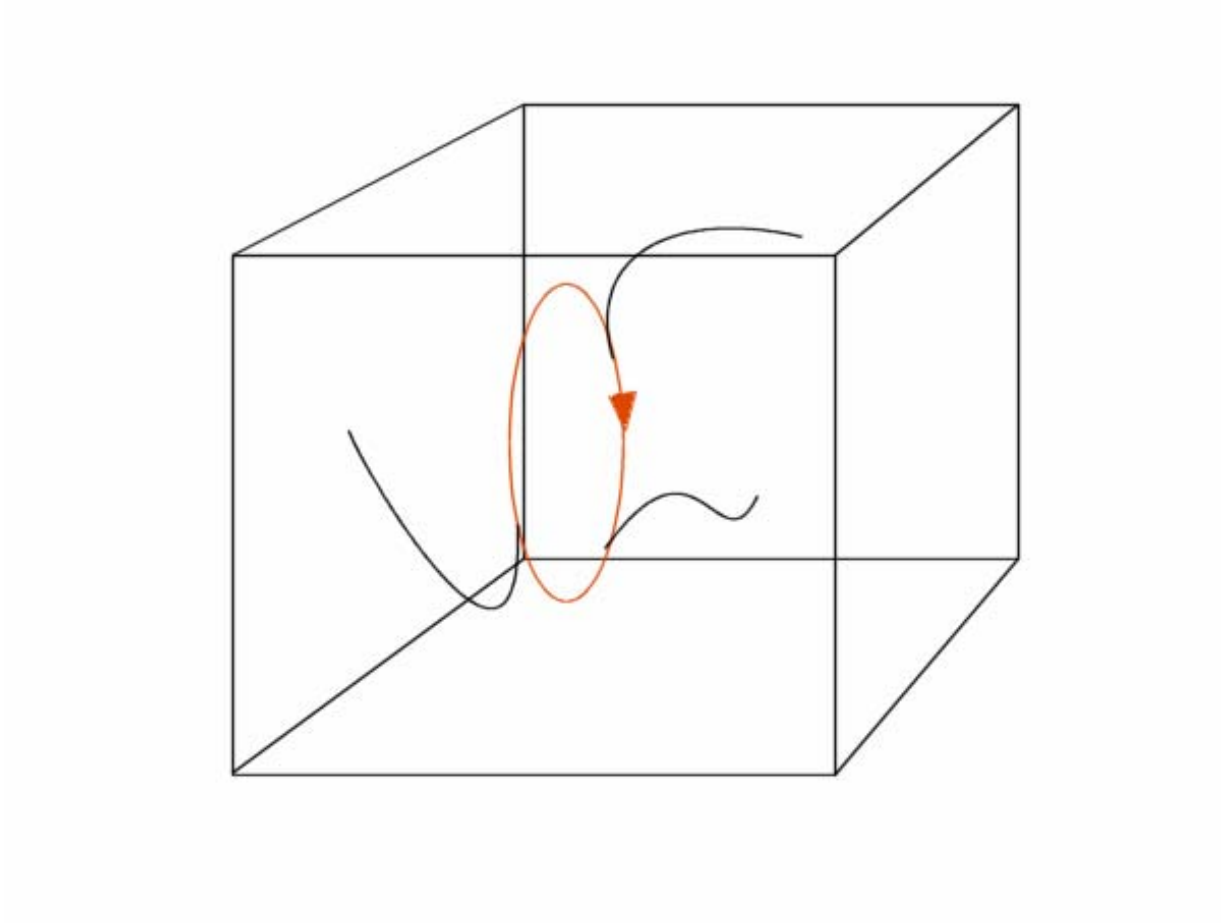






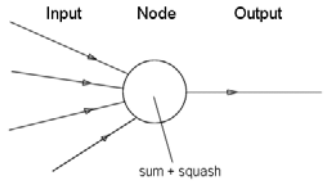


- CPG cycles are cyclic attractors in state space

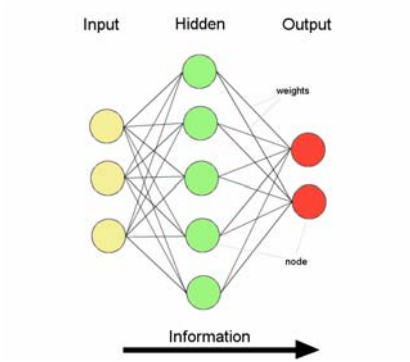


Recap – Neural Networks

- Components – biological plausibility
  - Neurone / node
  - Synapse / weight



- Feed forward networks
  - Unidirectional flow of information
  - Good at extracting patterns, generalisation and prediction
  - Distributed representation of data
  - Parallel processing of data
  - Training: Backpropagation
  - Not exact models, but good at demonstrating principles



- Recurrent networks
  - Multidirectional flow of information
  - Memory / sense of time
  - Complex temporal dynamics (e.g. CPGs)
  - Various training methods (Hebbian, evolution)
  - Often better biological models than FFNs