

Università di Verona

A.A. 2020-21

Machine Learning & Artificial Intelligence

Reti Neurali Artificiali

Vittorio Murino

Reference & Credits

- Libro di riferimento:

C.M. Bishop: “Neural Networks for Pattern Recognition”,
Clarendon Press 1995

- Credits:

Dr. Manuele Bicego

Approcci non algoritmici all'elaborazione dell'informazione

- Dato un problema da risolvere P , spesso esso può essere formalizzato da $\{P, C, f\}$,
 - P = formulazione del problema
 - $C = \{c_1, \dots, c_n\}$ insieme di configurazioni, ognuna delle quali rappresenta una possibile soluzione
 - $f: C \rightarrow \mathbb{R}$ funzione che fornisce una misura della bontà delle configurazioni rispetto allo scopo di risolvere il problema.
- Risolvere il problema significa massimizzare o minimizzare la funzione f nello spazio C

Esempi

- Problema 1: “ordinare in modo crescente un insieme di numeri $x_1 \dots x_n$ ”.
 - In questo caso la formalizzazione è semplice
 - P = ordinare i numeri $x_1 \dots x_n$;
 - C = insieme delle $n!$ permutazioni di $x_1 \dots x_n$;
 - f = funzione che somma le distanze tra ogni numero e il successivo

- Problema 2: “Prevedere l’indice della borsa domani”
 - La formalizzazione è più difficile

Soluzione algoritmica ai problemi

- Trovare un algoritmo che risolva il problema
- L'algoritmo rappresenta esso stesso la soluzione: dato l'algoritmo trova sempre la soluzione
- Esempio: Problema dell'ordinamento dei numeri, l'algoritmo *Bubble Sort* rappresenta esso stesso la soluzione del problema:
 - dato un insieme di numeri, seguendo passo passo le istruzioni contenute nella descrizione dell'algoritmo, si arriva alla soluzione, l'insieme dei numeri ordinati.

- Domanda: Esiste sempre una soluzione algoritmica?
- Risposta: Sì!
 - È la *Ricerca Esaustiva*, o approccio a forza bruta:
 - Si calcola $f(c_i)$ per ogni possibile configurazione, e si sceglie quella che massimizza la funzione di bontà f
 - Problema: ovviamente questo non è applicabile se lo spazio C è grande

Elaborazione non algoritmica

- Il problema 2 non può essere risolto in modo algoritmico (eccetto la forza bruta) :
 - informazione incompleta
 - non specificabilità rigorosa del problema
- Si può quindi utilizzare un approccio di elaborazione non algoritmica dell'informazione:
 - forniscono una codifica delle configurazioni C e della funzione f , e una serie di regole che fanno evolvere il sistema verso la soluzione (dato l'algoritmo, non si ha la soluzione)
 - utilizza informazioni che sono tipiche del problema
 - può utilizzare istanze di soluzioni

Esempi

- **Algoritmi Genetici:**

- si basa sulla teoria dell'evoluzione della specie
- la bontà di un individuo è legata alla sua “adattabilità”.


- **Simulated Annealing:**

- si basa sui modelli fisici: abbassando gradualmente la temperatura T di un sistema, esso tende a cristallizzare in una configurazione di minima energia

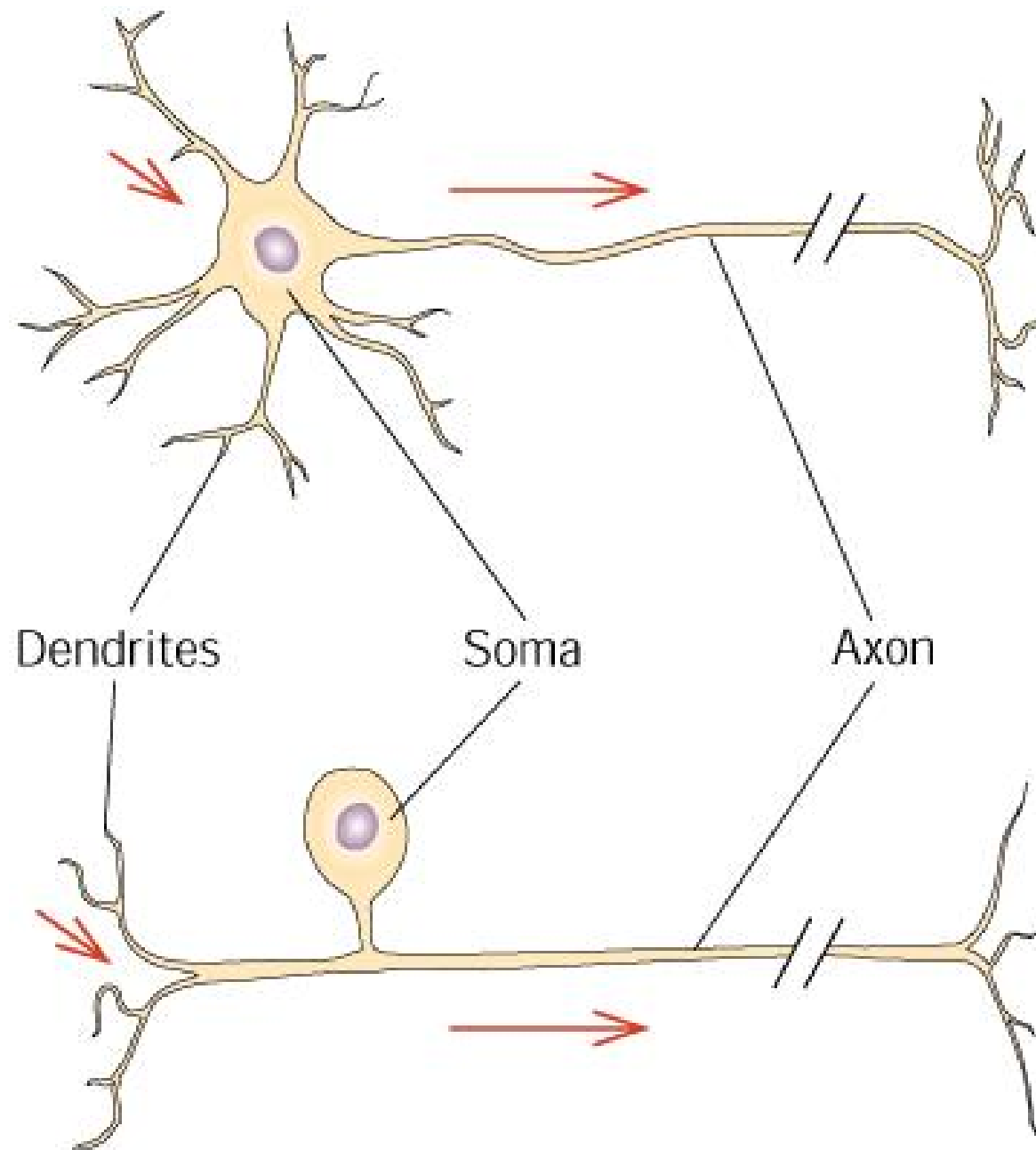
- **Reti Neurali:**

- Sistema artificiale di elaborazione dell'informazione che si pone come obiettivo l'emulazione del sistema nervoso animale, i.e., di emulare il sistema di computazione biologico.
- cercano di risolvere i problemi partendo da esempi di soluzioni.

Le reti neurali

- Il sistema nervoso animale presenta numerose caratteristiche che sono attraenti dal punto di vista di un sistema di calcolo:
 - è robusto e resistente ai guasti: ogni giorno muoiono alcuni neuroni senza che le prestazioni complessive subiscano un peggioramento sostanziale;
 - è flessibile: si adatta alle situazioni nuove imparando;
 - permette una computazione altamente parallela;
 - è piccolo, compatto e dissipa poca potenza.
- 
- può lavorare anche con informazione
 - **approssimata:** l'informazione rappresentata dal segnale non è descritta in maniera esatta
 - **incompleta:** il segnale potrebbe non arrivare in parte
 - **affetta da errore:** se i segnali sono affetti da errore il sistema deve essere in grado di rigenerarli (trascurare gli errori)

Il neurone



Il neurone

- Il flusso dell'informazione è unidirezionale:

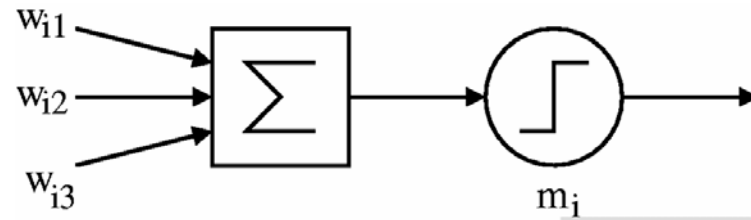
dendriti → corpo cellulare (soma) → assone

- Il neurone riceve input (potenziali) dai dendriti, pesandoli in qualche modo (sinapsi).
- Se la somma di questi input supera una certa soglia, il neurone si accende (emette un segnale 1).

Un po' di storia

- 1943 - McCulloch e Pitts:

- primo modello di neurone: modello biologico



- somma pesata degli input: se superano la soglia m_i , allora l'output è 1, altrimenti è 0;

- 1949, Hebb:

- dagli studi sul cervello, emerge che l'apprendimento non è una proprietà dei neuroni, ma è dovuto a una modifica delle sinapsi.

- Anni 60: boom di lavoro sulle reti neurali, nel gruppo di Rosenblatt:
 - calcolo dei pesi
 - lavoro sui percettroni, reti in cui i neuroni sono organizzati in livelli di elaborazione sequenziale
- 1969, Minsky e Papert:
 - dimostrano i limiti del percettrone: crolla l'entusiasmo sulle reti neurali.
- Anni 70: *associative content-addressable memory*:
 - reti associative, dove patterns simili venivano in qualche modo associati

- 1982, Hopfield:
 - propone un modello di rete per realizzare memorie associative.
 - viene introdotto un concetto di funzione di energia
 - la rete converge sempre verso il più vicino punto stabile di energia
- 1985, Rumelhart, Hinton e Williams:
 - formalizzano l'apprendimento di reti neurali con supervisione (Back-Propagation: metodo standard per l'addestramento delle reti neurali)

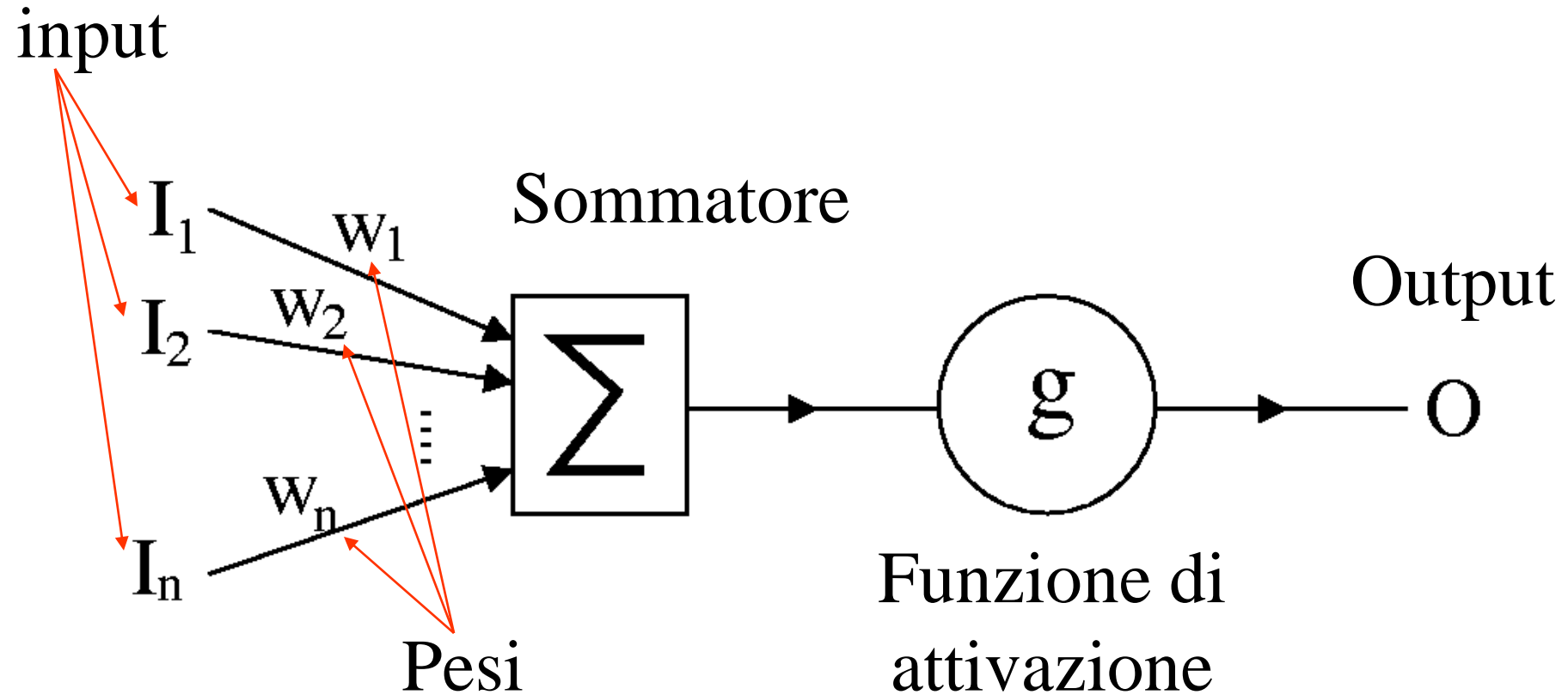
Schema di base

- Rete Neurale:
 - struttura complessa
 - composta da tante unità elementari di calcolo, chiamate *neuroni*
 - i neuroni sono collegati tra di loro tramite connessioni pesate, dette *sinapsi*
- Ci sono dei neuroni che sono connessi all'ambiente esterno (input o output)

- Ogni neurone possiede:
 - un insieme di ingressi provenienti dagli altri neuroni
 - un insieme di uscite verso gli altri neuroni
 - un livello di attivazione
 - un sistema per calcolare il livello di attivazione al tempo successivo

- Le diverse reti neurali differiscono per:
 - tipologia di neuroni
 - tipologia di connessioni tra i neuroni

Il Neurone



Il neurone: componenti

- *Input I_i* : sono i segnali in ingresso al neurone:
 - input del problema
 - uscite di altri neuroni
- *Pesi o Sinapsi w_i* : ogni input viene pesato tramite il peso della connessione; fornisce una misura di quanto “conta” tale input nel neurone
- *Sommatore Σ* : modulo che effettua la somma pesata degli ingressi

- *Funzione di attivazione g*: funzione che determina l'output del neurone sulla base dell'uscita del sommatore.

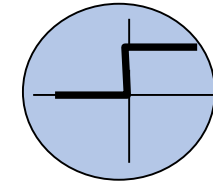
Riassumendo, l'*output* O del neurone si ottiene come

$$O = g \left(\sum_{i=1}^n w_i I_i \right)$$

Il Percettrone (*Perceptron*)

- Nel caso più semplice (e più biologicamente plausibile) la funzione di attivazione è una funzione *a soglia* θ :
 - se la somma pesata degli input è maggiore di una certa soglia t , allora il neurone “si accende” (output 1), altrimenti no.

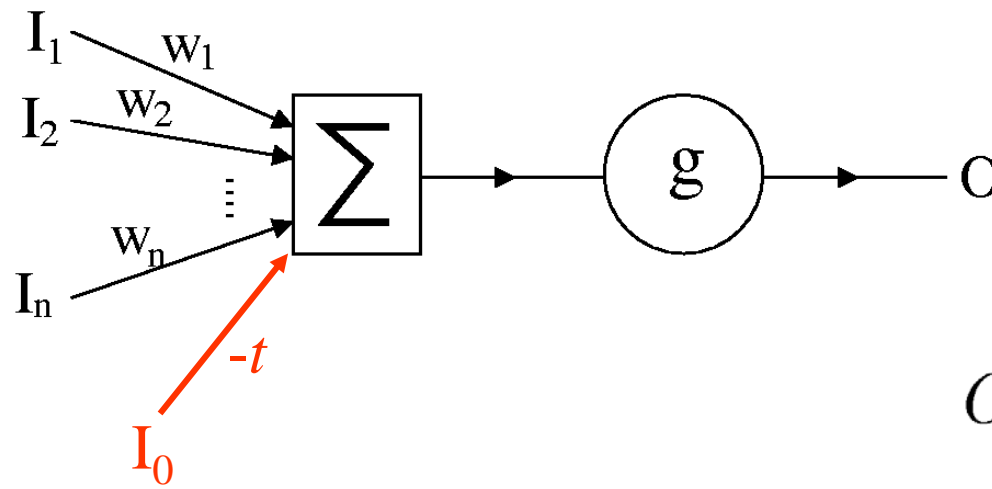
$$O = \theta \left(\sum_{i=1}^n w_i I_i - t \right)$$



dove θ è la funzione di *Heaviside* $\theta : \mathcal{R} \longrightarrow \{0, 1\}$

$$\theta(x) = \begin{cases} 0 & \text{se } x < 0 \\ 1 & \text{se } x \geq 0 \end{cases}$$

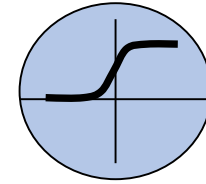
- Questo semplice neurone si chiama *Percettrone (Perceptron)*, introdotto da Rosenblatt nel 1962.
- Spesso si include la soglia nel modello del neurone, aggiungendo un ingresso fittizio:
 - il valore su tale ingresso è fissato a 1
 - il peso della connessione è dato da $-t$;



$$O = g \left(\sum_{i=0}^n w_i I_i \right)$$

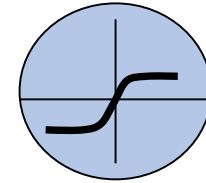
Altre funzioni di attivazione

- Funzione logistica: $g(x) = \frac{1}{1 + e^{-x}}$



- Funzione tangente iperbolica:

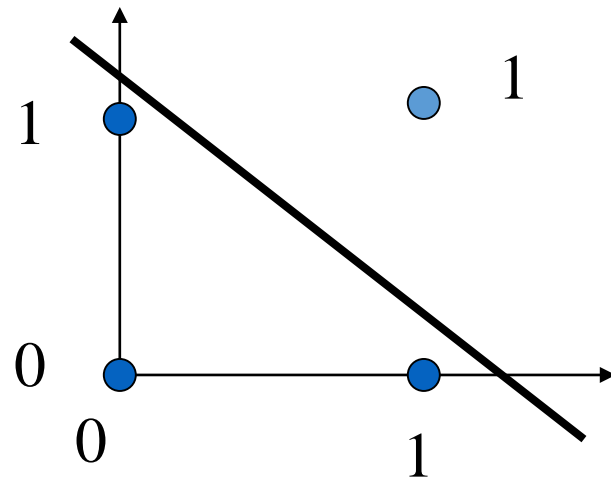
$$g(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



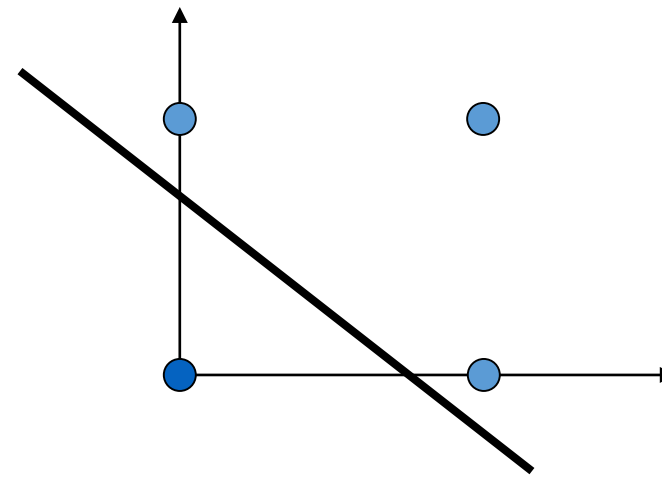
- Queste funzioni sono interessanti in quanto permettono al neurone di avere un output continuo, che ne consente una interpretazione in chiave probabilistica.

Capacità espressive del percettrone

- Ogni percettrone può rappresentare solo funzioni linearmente separabili.
- La superficie di separazione tra gli esempi positivi e negativi è un piano.



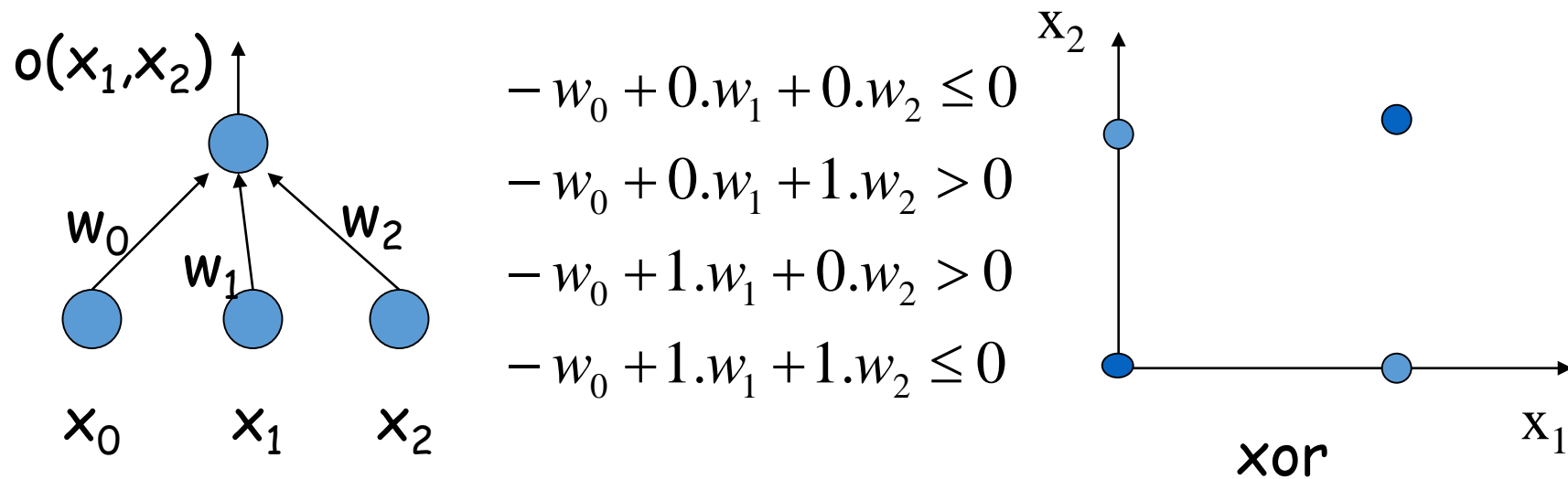
and



or

Limiti espressivi del perceptrone

- La funzione XOR non può essere rappresentata, essendo non linearmente separabile



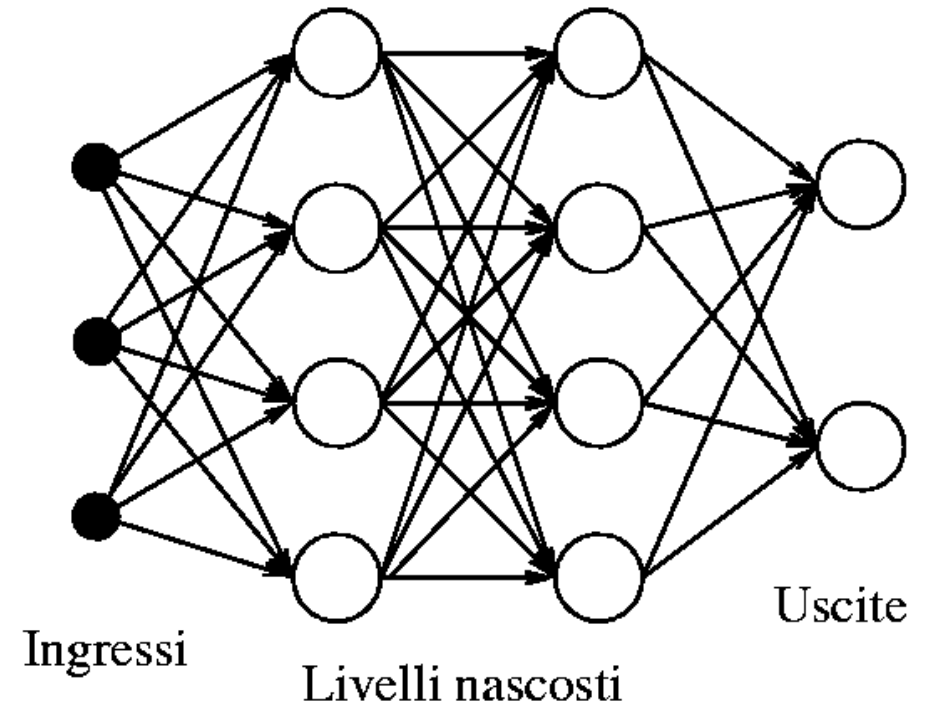
- Non è possibile trovare valori di w_0, w_1 e w_2 che soddisfino le disequazioni precedenti

La struttura delle reti

- L'aggregazione di più neuroni a formare una rete provoca un'esplosione di complessità, in quanto si possono formare topologie complesse in cui sono presenti cicli.
- Il modello più semplice, tuttavia, non prevede cicli: l'informazione passa dall'input all'output (reti *feed-forward*)
- In caso in cui siano presenti cicli, si parlerà di *reti ricorrenti*.

Le reti *feed forward*

- Topologia più semplice: non prevede cicli.
- La rete è organizzata a livelli:
 - ogni neurone di un livello riceve input solo dai neuroni del livello precedente;
 - propaga gli output solamente verso i neuroni dei livelli successivi
- I livelli compresi fra gli input e l'output vengono chiamati livelli nascosti (in questo caso 2).



- In questo tipo di reti non sono possibili autocollegamenti o connessioni con i neuroni del proprio livello.
- Ogni neurone ha quindi la funzione di propagare il segnale attraverso la rete, con un flusso di informazione dagli ingressi verso le uscite.
 - Una conseguenza immediata di ciò è rappresentata dal fatto che la rete risponde sempre nello stesso modo se sollecitata con gli stessi input.
- Nel caso in cui le unità elementari siano i percettroni, prenderà il nome di Percettrone Multilivello (o *MLP*, *Multilayer Perceptron*).

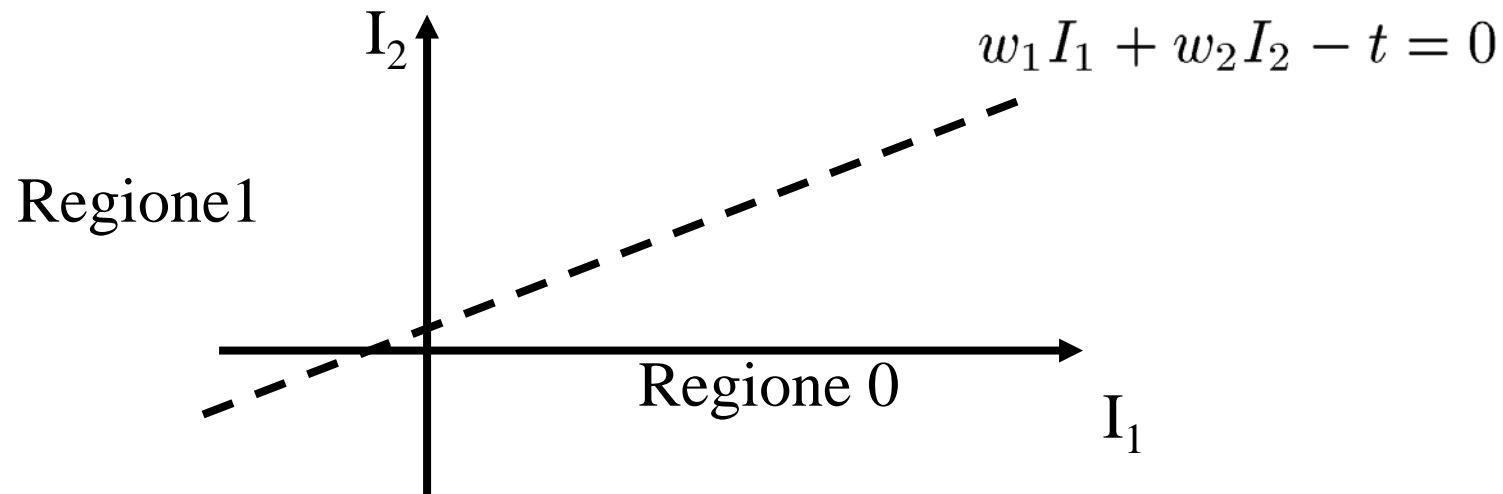
Capacità di classificazione

- Le classi sono rappresentabili come regioni nello spazio degli input.
- Nel semplice caso di due classi, vogliamo creare uno strumento che dia in uscita 0 se l'input appartiene alla prima classe e 1 se appartiene alla seconda.
- L'obiettivo diventa quindi quello di approssimare una regione nello spazio (ad esempio la regione degli input che appartengono alla classe 1, la classe 2 è il complementare):
 - input appartenenti alla regione della classe 1 faranno “accendere” l'output della rete neurale

- Obiettivo: costruire una rete neurale in grado di riconoscere una regione.
- Domanda: in che modo la topologia della rete influisce sulla complessità delle regioni riconoscibili?
- Consideriamo un percettrone con due ingressi I_1 I_2 : l'output è dato da:

$$O = \begin{cases} 1 & \text{se } w_1 I_1 + w_2 I_2 - t \geq 0 \\ 0 & \text{se } w_1 I_1 + w_2 I_2 - t < 0 \end{cases}$$

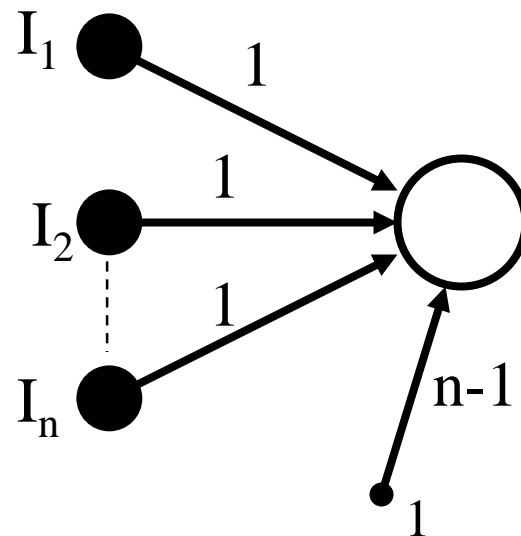
- Il neurone suddivide quindi il piano degli input in due semipiani:
 - da una parte ci sono gli input che producono come output del neurone 1, dall'altra ci sono gli input che producono output 0;



la retta che li delimita è identificata da:

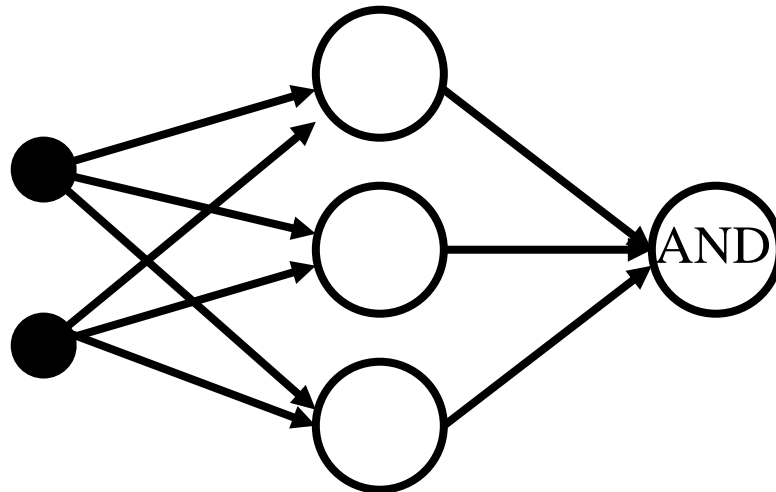
$$w_1 I_1 + w_2 I_2 - t = 0$$

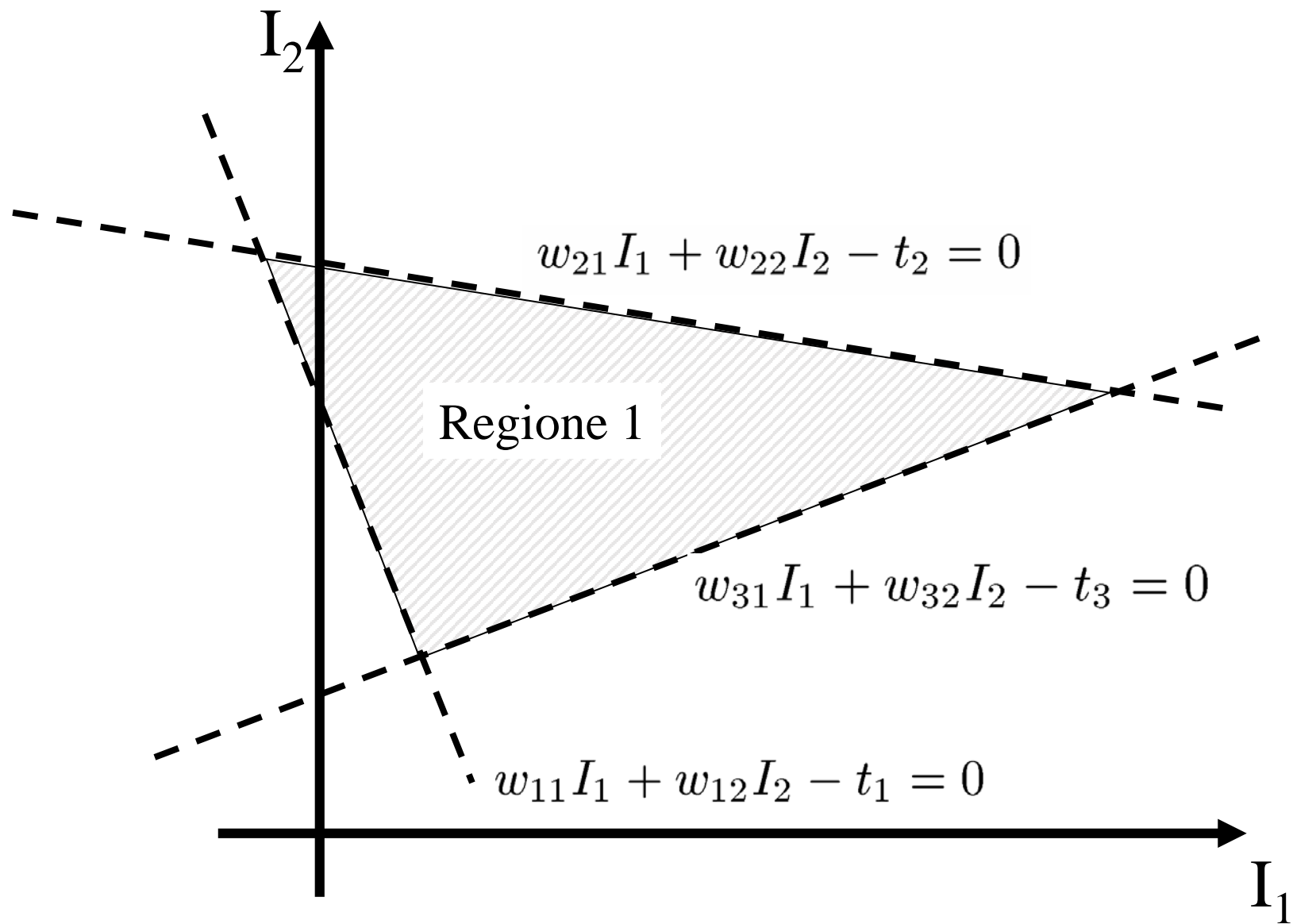
- Per n ingressi un percettrone definisce un iperpiano di dimensionalità $n-1$, che suddivide lo spazio degli input in due semispazi: da una parte ci sono i valori per i quali il neurone si accende, dall'altra i rimanenti.
- Consideriamo un neurone che funga da AND



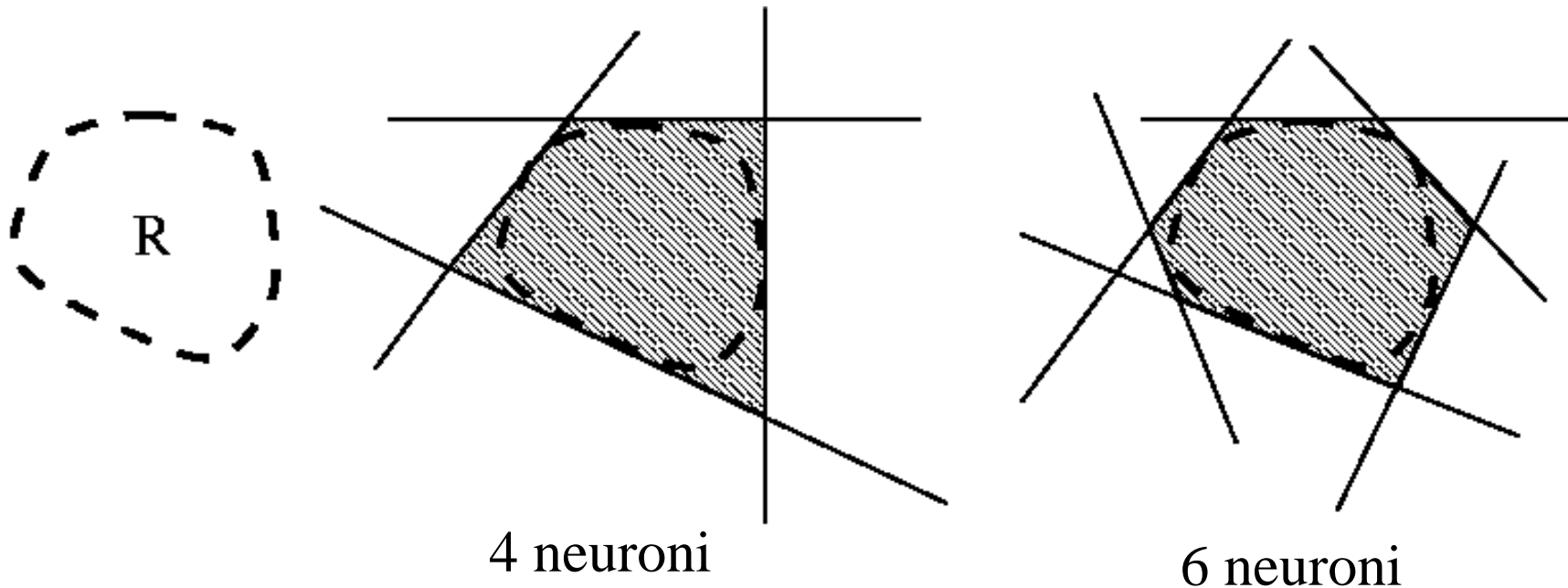
Funzione AND degli
ingressi $I_1 \dots I_n$

- Consideriamo un MLP - Percettrone Multilivello a due livelli:
 - ogni neurone identifica un semipiano
 - il neurone del secondo livello effettua un AND dell'output dei neuroni al livello precedente
 - con tre neuroni possiamo identificare una regione chiusa a forma di triangolo



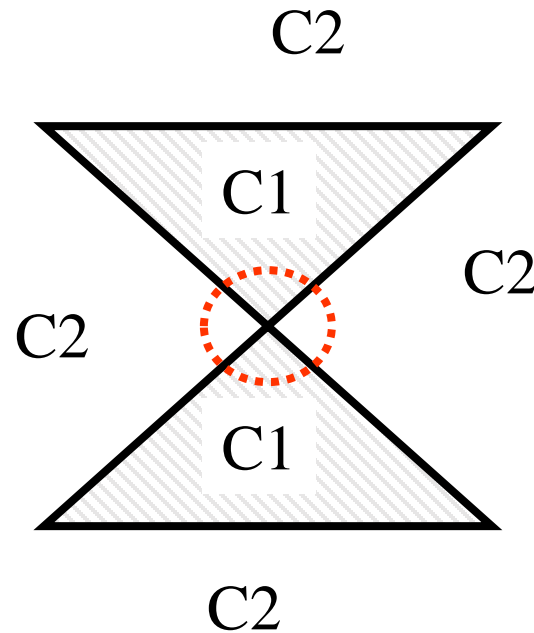


- Utilizzando 4 neuroni nel primo livello, possiamo ottenere un quadrilatero, e così via...
- Data una regione R da delimitare nello spazio degli input, più sono i percettroni, più precisa sarà l'approssimazione che la rete produrrà.



- *1 livello nascosto*: riusciamo ad ottenere approssimazioni di regioni convesse e connesse
- Domanda: ci sono delle regole per capire come varia la capacità approssimante con il variare della struttura?
- Sí! Esistono tre teoremi
- **Teorema 1** (Huang & Lipmann '88): Reti neurali *feed forward* con un livello nascosto riescono ad approssimare “quasi tutte” le superfici di separazione tra due classi.
 - “Quasi tutte”: aumentando il numero di neuroni nell'ultimo strato (da 1 a 2 o 3), si riesce a classificare anche regioni non connesse o convesse.

- **Teorema 2** (Gibson & Lowan '90): Ci sono regioni che reti neurali con un livello nascosto non riescono ad approssimare:

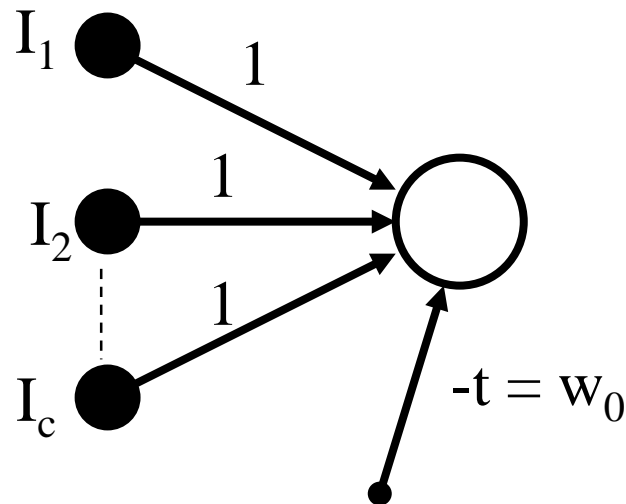


Il problema sta nel punto di contatto, cerchiato in rosso

- **Teorema 3** (Lipmann '87): Reti neurali con due livelli nascosti possono approssimare qualsiasi regione dello spazio (in un problema a 2 classi).
 - La precisione dell'approssimazione dipende poi dal numero di neuroni utilizzati negli strati intermedi
- In pratica? Non ci sono regole fisse, spesso ci si basa su regole empiriche o euristiche.

Esempio: riconoscitore di template

- Obiettivo: costruire un riconoscitore di pattern binari $\mathbf{b} = b_1 \dots b_c$.
- Utilizzo una rete semplicissima, con c input e un solo neurone: l'uscita sarà 1 se il pattern presentato corrisponde al pattern di riferimento \mathbf{b} , 0 altrimenti.



- Come si costruisce questa rete neurale?
- Il peso di ogni connessione w_i viene fissato a

$$w_i = \begin{cases} 1 & \text{se } b_i = 1 \\ -1 & \text{se } b_i = 0 \end{cases}$$

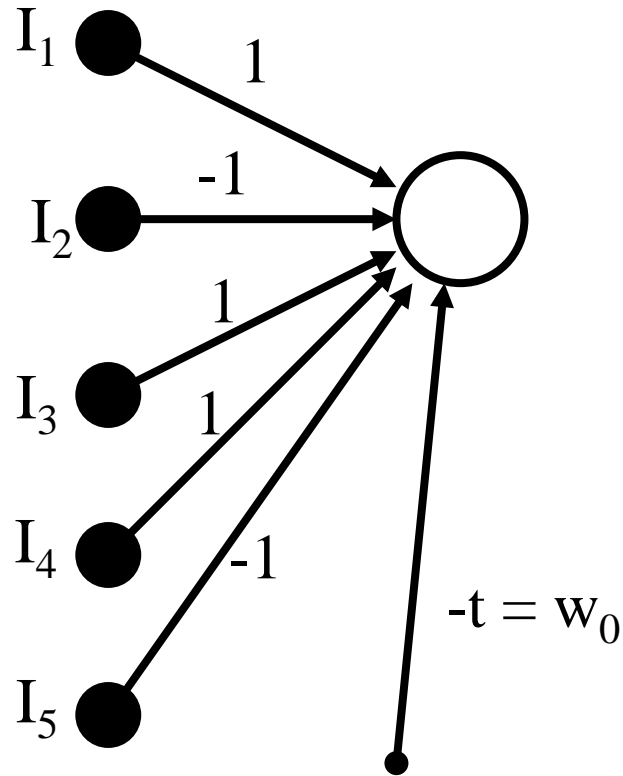
- Definiamo m come

$$m = \sum_{i=1}^c b_i$$

- Avremo che

$$\begin{cases} \sum_{i=1}^c I_i w_i = m & \text{se } I_i = b_i \quad \forall i = 1 \dots c \\ \sum_{i=1}^c I_i w_i \leq m - 1 & \text{altrimenti} \end{cases}$$

- Esempio: $\mathbf{b}=10110$, $m=3$; la rete diventa:



Proviamo a dare in input alla rete
 $\mathbf{I} = 10110 = \mathbf{b}$

$$\begin{aligned}\sum_{i=1}^c w_i I_i &= \\ &= 1 \cdot 1 + 0 \cdot -1 + 1 \cdot 1 + 1 \cdot 1 + 0 \cdot -1 = \\ &= 3 = m\end{aligned}$$

Proviamo ora a dare un input sbagliato
 $\mathbf{I} = 11110$

$$\begin{aligned}\sum_{i=1}^c w_i I_i &= \\ &= 1 \cdot 1 + 1 \cdot -1 + 1 \cdot 1 + 1 \cdot 1 + 0 \cdot -1 = \\ &= 2 \leq m - 1\end{aligned}$$

- Scegliendo come soglia $m-1$, avremo che:

$$\begin{cases} \sum_{i=0}^c I_i w_i = 1 & \text{se } I_i = b_i \quad \forall i = 1 \dots c \\ \sum_{i=0}^c I_i w_i \leq 0 & \text{altrimenti} \end{cases}$$

- Utilizzando come funzione di attivazione la funzione di *Heaviside*, otteniamo esattamente il classificatore cercato

$$O = \begin{cases} 1 & \text{se } I_i = b_i \quad \forall i = 1 \dots c \\ 0 & \text{altrimenti} \end{cases}$$

- Considerazione: posso approssimare con una rete neurale qualsiasi funzione booleana:

000	1	←
001	0	
010	1	←
011	0	
100	0	
101	1	←
110	0	
111	0	

Data una funzione booleana generica, posso costruire i neuroni che riconoscono i pattern che danno vero (quelli con la freccetta), mettendoli quindi in AND tra di loro.

L'addestramento della rete neurale

- Una volta stabilite tutte le caratteristiche della rete neurale, quali
 - topologia,
 - numero e tipo di neuroni,
 - collegamenti, etc.

occorre determinare i pesi delle connessioni in modo da costruire un classificatore, avendo a disposizione una serie di esempi tratti dal problema in questione: questa operazione prende il nome di *addestramento* della rete neurale.

- Consideriamo il caso di *addestramento supervisionato*, dove, per ogni pattern dell'insieme degli esempi, viene specificato anche il valore di uscita desiderato.
- L'addestramento supervisionato consiste quindi nel:
 - presentare alla rete esempi tratti dal problema in esame
 - aggiustare i pesi delle connessioni sulla base delle discrepanze tra l'uscita prodotta e l'uscita desiderata.

L'insieme degli esempi prende il nome di insieme di apprendimento, *training set*, ed è una tabella del tipo:

In	Out
x_1	y_1
x_2	y_2
\vdots	\vdots
x_p	y_p

Il *training set*

- Training set $T = \{(x_1, y_1) \dots (x_p, y_p)\}$:
 - x_i vettori di input
 - y_i classe a cui il vettore di input appartiene
- Rappresenta istanze del problema: descrive alcuni dei valori che le variabili assumono nelle classi.
- Deriva da misurazioni, quindi può contenere informazione incompleta, inesatta o approssimata.
- Rappresenta un **fattore critico**: deve essere rappresentativo del fenomeno in questione:
 - deve contenere esempi rappresentativi di ogni classe;
 - tutte le classi devono essere ben rappresentate.

Formalizzazione

- Dato il training set $T = \{(x_1, y_1) \dots (x_p, y_p)\}$, l'obiettivo è quello di aggiustare i pesi della rete sulla base di questi esempi
 - la rete deve funzionare bene almeno su questi esempi
- Definiamo $f_W(x_i)$ l'uscita della rete per l'input x_i , che dipende dai pesi W .
- Una volta fissata la topologia, W identifica unicamente una rete neurale.

- Si cercherà la configurazione di pesi W che minimizzi una funzione costo $E(W)$.
- Possono essere definiti diverse tipologie di funzione di costo: la più semplice è l'*errore quadratico medio*, definito come

$$E(W) = \frac{1}{2} \sum_{i=1}^p (y_i - f_W(x_i))^2$$

L'obiettivo diventa quindi quello di trovare la configurazione W^* che minimizzi l'errore $E(W)$.

$$W^* = \arg \min E(W)$$

La minimizzazione dell'errore

- Essendo tale funzione differenziabile, possiamo calcolarne il minimo utilizzando il metodo di *discesa lungo il gradiente*:
 - si parte da una configurazione arbitraria, spesso scelta in modo casuale, denotata con $W^{(0)}$.
 - I pesi vengono quindi aggiornati iterativamente secondo la formula

$$W^{(k+1)} = W^{(k)} + \Delta W^{(k)}$$

- dove

$$\Delta W^{(k)} = -\eta \left. \frac{\partial E}{\partial W} \right|_{W=W^{(k)}}$$

- dove η rappresenta un reale positivo piccolo detto parametro o tasso di apprendimento (*learning rate*).
- Ad ogni iterazione l'idea è quindi quella di spostarsi, all'interno dello spazio dei pesi, lungo la direzione di massima diminuzione dell'errore, cioè in quella opposta a quella del gradiente.
- L'aggiustamento che si apporta ai pesi ad ogni iterazione è determinato dal parametro di apprendimento η :
 - più è grande, maggiore sarà la correzione che si effettuerà ad ogni iterazione.

Aggiornamento dei pesi: due approcci

- *Approccio batch*: le modifiche ai pesi vengono apportate solo dopo che alla rete sono stati presentati tutti i patterns dell'insieme di apprendimento:
 - in altre parole si valuta l'errore della rete sull'intero (o una parte, *batch* appunto) insieme degli esempi prima di aggiornare i pesi;
 - l'idea è quella di fare poche modifiche ma sostanziali.
- *Approccio on-line*: le modifiche avvengono dopo la presentazione di ogni singolo pattern;
 - si procede quindi con tante piccole modifiche.

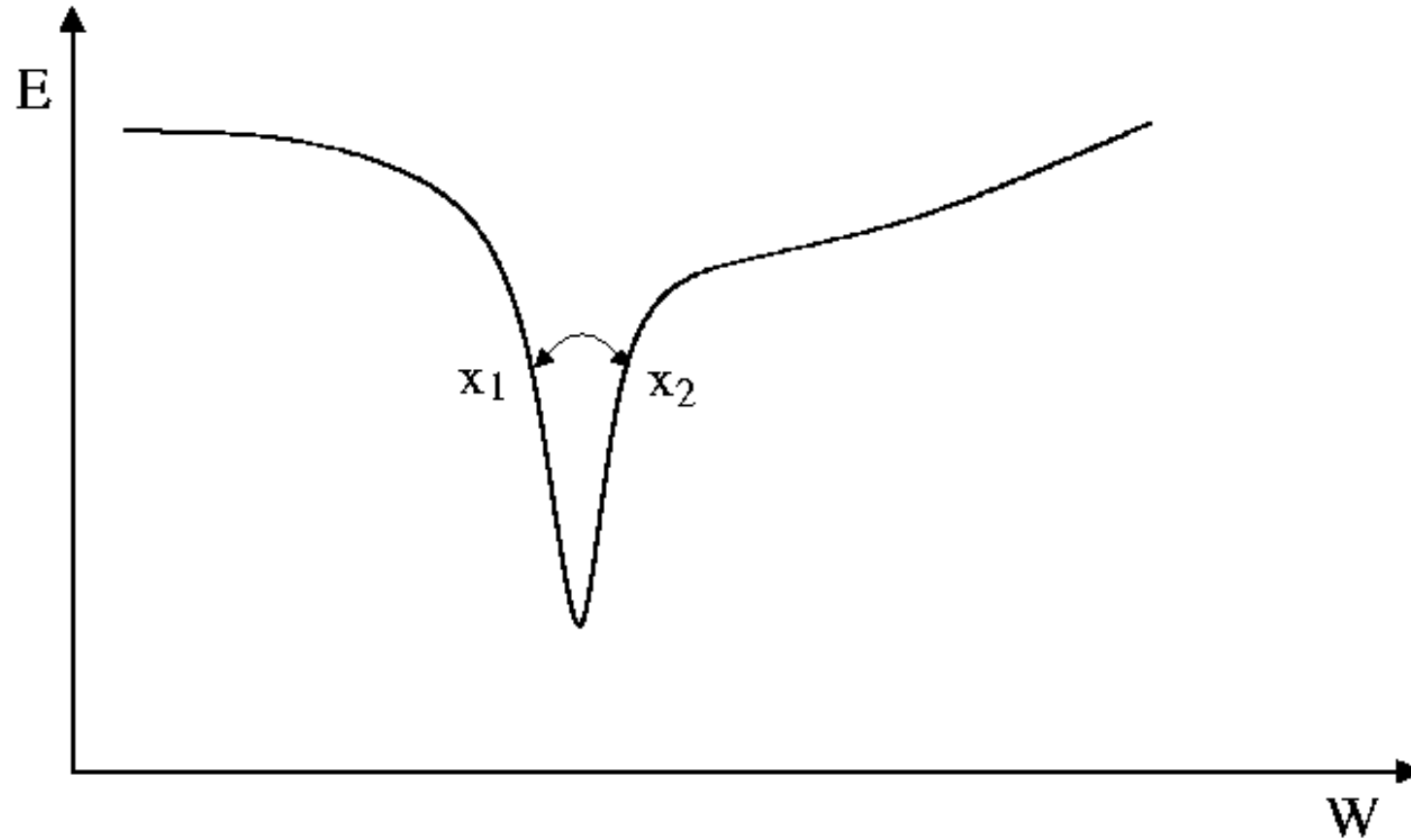
- Il secondo approccio, fissando il parametro di apprendimento η sufficientemente piccolo e scegliendo in modo casuale gli esempi da presentare alla rete, consente una vasta esplorazione dello spazio della funzione di costo.
- Il primo approccio è ora più usato non sull'intero training set ma su sottoinsiemi selezionati (*batch*).
- *Epoca*: la presentazione alla rete di tutti i *patterns* dell'insieme degli esempi:
 - essa rappresenta l'unità di misura del tempo di addestramento

Svantaggi del metodo di discesa lungo il gradiente

Svantaggio 1

Il parametro di apprendimento η è un fattore critico:

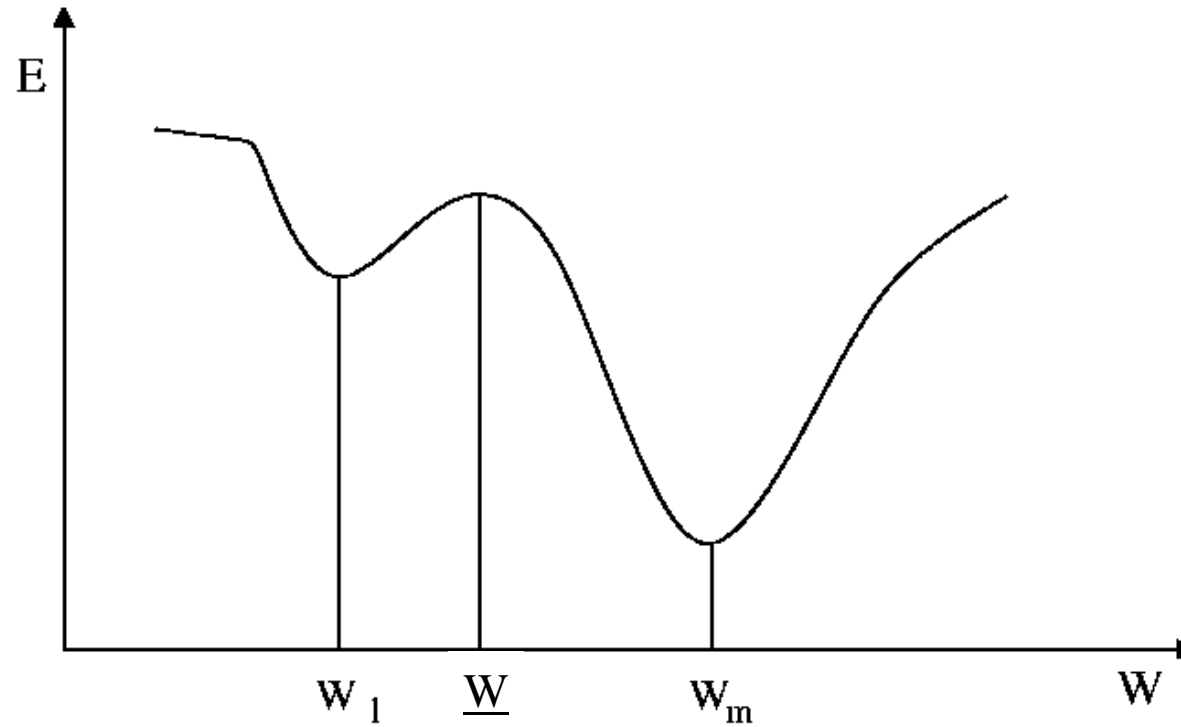
- esso rappresenta l'entità della correzione effettuata sui pesi ad ogni iterazione dell'algoritmo e ne determina quindi la velocità di convergenza;
- se η è troppo piccolo, la convergenza può essere troppo lenta;
- per contro, un η troppo grande può impedire un'accurata determinazione della configurazione minima e si potrebbero avere delle oscillazioni.



Per η troppo grande, l'algoritmo potrebbe non riuscire a scendere nella valle, oscillando tra i due punti x_1 e x_2 .

■ Svantaggio 2

Questa tecnica converge al più vicino minimo locale: la scelta iniziale dei valori dei pesi può essere critica.



- Se si sceglie come configurazione iniziale un qualsiasi punto maggiore di \underline{W} , l'algoritmo riesce a raggiungere il minimo W_m della funzione di errore, altrimenti si ferma nel minimo locale W_1 .

- Una soluzione potrebbe essere quella di introdurre nella formula di aggiornamento dei pesi dei termini chiamati *momenti*:
 - l'idea è quella di aggiungere alla variazione sui pesi da effettuare ad ogni iterazione un contributo che derivi dal passo precedente, imponendo una specie di *inerzia* al sistema.

$$\Delta W^{(k)} = -\eta \frac{\partial E}{\partial W} \bigg|_{W=W^{(k)}} + \alpha (W^{(k)} - W^{(k-1)})$$

con $0 < \alpha < 1$ detto *parametro del momento* (usualmente $\alpha = 0.9$).

- In alternativa si possono utilizzare tecniche di minimizzazione globali, quali ad esempio *simulated annealing*, o gli *algoritmi genetici*, oppure ancora *Reactive Tabu Search*.

■ Svantaggio 3

Se con la discesa lungo il gradiente si arriva in una zona in cui le funzioni di attivazione saturano (cioè $E(g)$ è quasi costante):

- la derivata di g è quasi nulla,
- ad ogni iterazione la correzione è molto piccola,
- avremo una drastica riduzione della velocità di apprendimento.

- Questo può essere parzialmente evitato se si sceglie una configurazione iniziale con pesi molto piccoli: almeno inizialmente le funzioni di attivazione non saturano.

L'algoritmo di *Backpropagation*

- Tecnica ottimizzata per l'addestramento della rete
- Motivazione: il calcolo della derivata di E rispetto a W effettuata con il rapporto incrementale, è oneroso: la sua complessità è $O(W^2)$, con W numero di pesi della rete.
 - Occorre infatti calcolare per ogni peso il rapporto incrementale

$$\frac{E(W + h) - E(W)}{h}$$

- il cui calcolo ha complessità è $O(W)$
- occorre valutare tutta la rete per avere $E(W)$, quindi occorre valutare W volte il rapporto incrementale

Backpropagation (2)

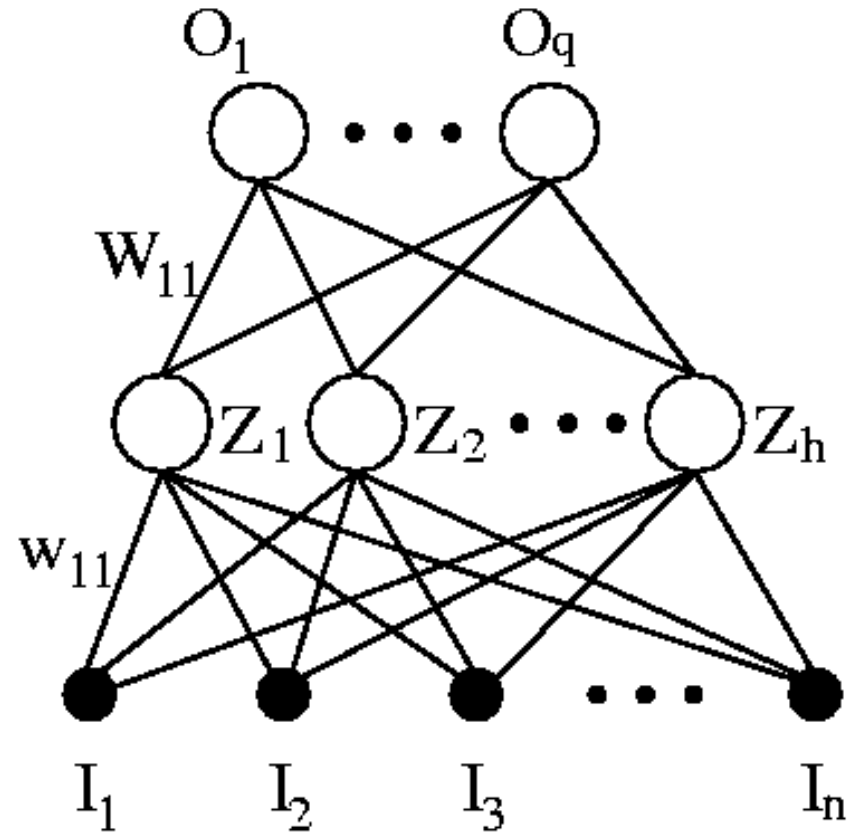
- Metodo di addestramento delle reti neurali feed forward.
- Si basa sul metodo di discesa lungo il gradiente.
- Ottimizza il calcolo della derivata, arrivando ad avere una complessità totale di $O(W)$, con W numero di pesi.

Schema generale

- La *back propagation* prevede due fasi, una fase *in avanti* e una fase *indietro*:
- *Fase in avanti (forward)*:
 - presentare un esempio alla rete;
 - determinare l'uscita e calcolare l'errore.
- *Fase indietro (backward)*:
 - l'errore viene propagato indietro nella rete, aggiustando progressivamente i pesi.

Formalizzazione

- Consideriamo una rete *feed forward* a 2 livelli a n input, h neuroni nel livello nascosto e q output
- g funzione di attivazione derivabile (es.: logistica)



$$O_k = g(a_k)$$

$$W_{kj}$$

$$Z_j = g(a_j)$$

$$w_{ji}$$

$$I_i$$

- I_i : input della rete.
- w_{ji} : peso della connessione dall' i -esimo input al j -esimo neurone del livello nascosto.
- a_j : somma pesata degli input del j -esimo neurone del livello nascosto

$$a_j = \sum_{i=1}^n w_{ji} I_i$$

- Z_j : output del j -esimo neurone del livello nascosto

$$Z_j = g(a_j)$$

- W_{kj} : peso della connessione dal j -esimo neurone del livello nascosto al k -esimo neurone del livello dell'output.
- a_k : somma pesata degli input del k -esimo neurone del livello di uscita

$$a_k = \sum_{j=1}^h W_{kj} Z_j$$

- O_k : output del k -esimo neurone del livello di uscita

$$O_k = g(a_k)$$

- Considerando un approccio *on-line* all'addestramento, la funzione di errore (errore quadratico medio) diventa:

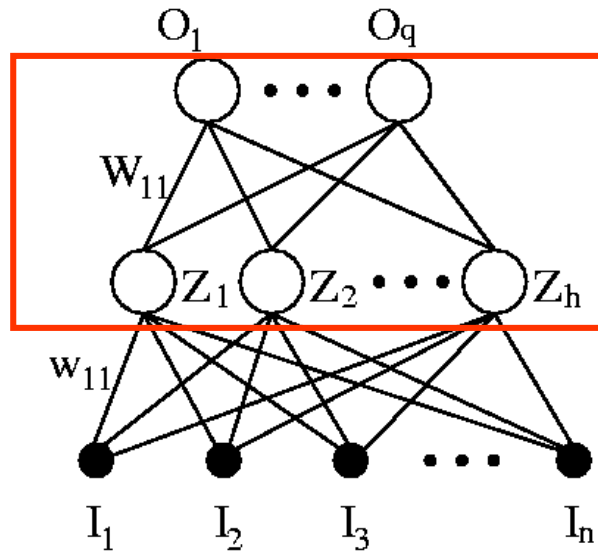
$$\begin{aligned} E(W) &= \frac{1}{2} \sum_i (y_i - O_i)^2 \\ &= \frac{1}{2} \sum_i \left[y_i - g \left(\sum_j W_{kj} g \left(\sum_i w_{ji} x_i \right) \right) \right]^2 \end{aligned}$$

dove (x_i, y_i) rappresenta un pattern dell'insieme dell'addestramento, e O_i sono le uscite della rete con ingressi x_i (ossia I_i in figura)

Calcolo derivata

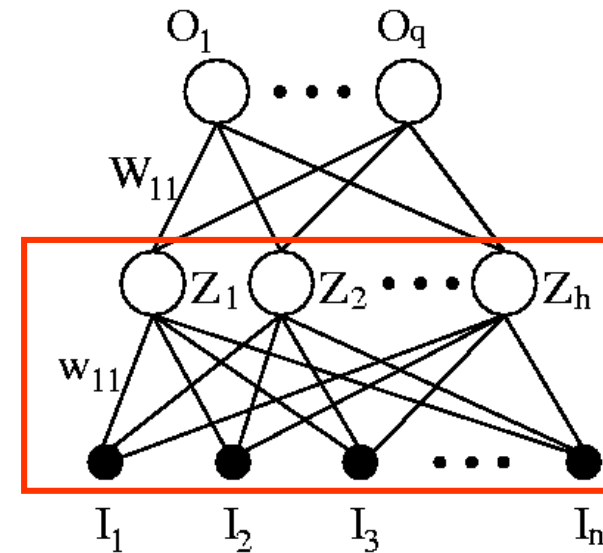
- Calcoliamo la derivata di E rispetto ai pesi, in due passi separati:

Passo 1



Pesi delle connessioni
livello nascosto - output

Passo 2



Pesi delle connessioni
input - livello nascosto

Connessioni livello nascosto - uscita

- Per ogni singolo peso w_{lm} si può applicare la regola della catena

$$\frac{\partial E}{\partial w_{lm}} = \frac{\partial E}{\partial a_l} \frac{\partial a_l}{\partial w_{lm}}$$

- Per le connessioni tra il livello nascosto e l'uscita si ha che

$$\frac{\partial E}{\partial W_{kj}} = \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial W_{kj}}$$

con

$$\begin{aligned}\frac{\partial E}{\partial a_k} &= \frac{\partial E}{\partial O_k} \frac{\partial O_k}{\partial a_k} \\ &= (y_k - O_k) g'(a_k)\end{aligned}$$

e

$$\frac{\partial a_k}{\partial W_{kj}} = Z_j$$

Definendo:

$$\delta_k = \frac{\partial E}{\partial a_k} = (y_k - O_k) g'(a_k)$$

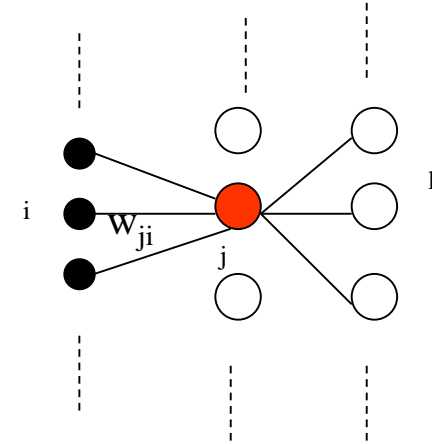
otteniamo che

$$\frac{\partial E}{\partial W_{kj}} = \delta_k Z_j$$

Connessione input-livello nascosto

Applicando la regola della catena:

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



abbiamo che

$$\begin{aligned} \delta_j &= \frac{\partial E}{\partial a_j} = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} = \\ &= \sum_k \delta_k w_{kj} \frac{\partial z_j}{\partial a_j} = \sum_k \delta_k w_{kj} g'(a_j) = g'(a_j) \sum_k \delta_k w_{kj} \end{aligned}$$

- La derivata di a_j rispetto w_{ji} è

$$\frac{\partial a_j}{\partial w_{ji}} = I_i$$

- Riassumendo, definendo

$$\delta_j = g' (a_j) \sum_k \delta_k w_{kj}$$

- otteniamo

$$\frac{\partial E}{\partial w_{ji}} = \delta_j I_i$$

Generalizzando ...

- La derivata nei due passi è calcolata secondo una formula simile. Generalizzando, abbiamo che

$$\Delta W_{pq} = -\eta \delta_{output} \times V_{input}$$

dove:

- *input* e *output* rappresentano il neurone origine (*q*) e il neurone destinazione (*p*) della connessione
- *V* rappresenta il valore di uscita del neurone
- δ rappresenta “l’errore” che viene propagato all’indietro, varia da livello a livello ma mantiene una formula simile a

$$\delta_j = g' (a_j) \sum_k \delta_k w_{kj}$$

Riassumendo:

- Fase *forward*:
 - l'ingresso x_i viene inserito nella rete;
 - vengono calcolate le Z e le O e memorizzate (in generale, le attivazioni delle unità nascoste e di uscita)
 - La complessità è $O(W)$
- Fase *backward*:
 - si calcola δ_k per le unità di uscita
 - si propaga δ_k all'indietro per calcolare δ_j (per ogni unità nascosta)
 - La complessità è $O(W)$
- Il tutto viene ripetuto per ogni pattern.

Considerazioni

- La complessità del calcolo della derivata è $O(W)$: si ha quindi una drastica riduzione della complessità rispetto all'uso del rapporto incrementale
- La regola di aggiornamento dei pesi è una regola *locale*:
 - per calcolare per la variazione di un peso l'algoritmo necessita solamente delle informazioni presenti alle due estremità della connessione;
 - questo permette un'alta parallelizzazione del processo di addestramento della rete neurale.

La generalizzazione

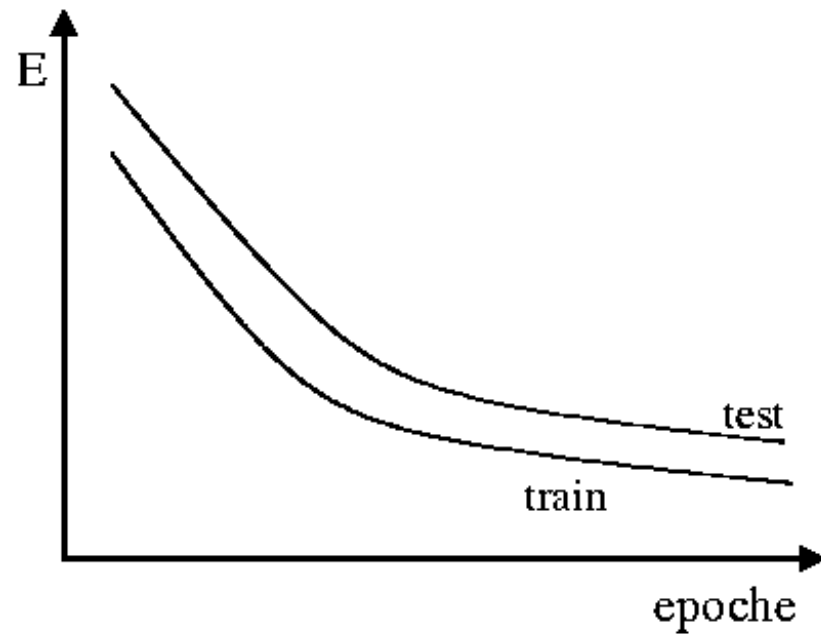
- *Capacità di generalizzazione di una rete*

Abilità nel riconoscere e valutare correttamente esempi del problema in esame non presenti nell'insieme di apprendimento.

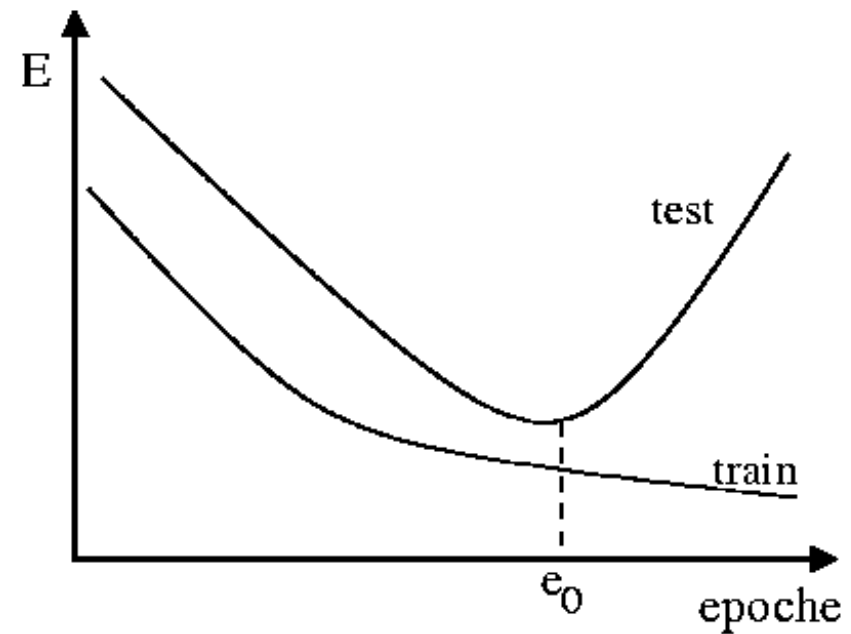
- Questa capacità è determinata dalla topologia della rete (numero di livelli, neuroni per livello etc.) e dalla scelta dell'insieme di addestramento:

- più esso è rappresentativo del fenomeno in questione, maggiore sarà la capacità di generalizzazione.

- Misura della capacità di generalizzazione:
 - occorrerebbe riestrarre altri esempi dal problema e utilizzarli per testare la rete: purtroppo questo può risultare troppo dispendioso;
 - l'alternativa che si adotta nella maggior parte dei casi è quella di suddividere l'insieme a disposizione in due parti
 - utilizzare una parte per addestrare
 - utilizzare l'altra parte per testare la rete.
 - Calcolando l'errore sull'insieme di addestramento e su quello di test in funzione del tempo si può decidere quando smettere il *training*



La rete generalizza bene e si può arrivare ad avere un basso valore di errore



Dopo un numero di epoche e_0 , l'errore sull'insieme di test comincia a crescere: si verifica una situazione di *over-training*. Questo significa che la rete comincia a perdere la capacità di generalizzare ed è quindi opportuno interrompere l'addestramento

- *Overtraining*

situazione in cui la rete ha imparato talmente bene i pattern di esempio da aver perso la flessibilità per poter riconoscere altri pattern, ovvero la rete ha “memorizzato” i pattern del training set, senza aver imparato nulla.

- Nota: l'addestramento della rete non è equivalente alla minimizzazione dell'errore sull'insieme di training (occorre tener conto dell'overtraining).
- In altre parole, non è sempre detto che il minimo dell'errore corrisponda al miglior addestramento per la rete neurale.

Suddivisione Training - Testing set

- Tipicamente l'insieme di dati a disposizione è limitato:
 - non è pensabile di costruire insiemi di test ripetendo esperimenti.
 - occorre sfruttare al meglio l'insieme dato:
- Esistono diversi metodi per suddividere il training set dal testing set:
 - metodo *Resubstitution*
 - metodi di *Cross Validation*

Il metodo *Resubstitution*

- L'idea in questo metodo è quella di costruire il classificatore e di testarlo con lo stesso insieme, l'intero insieme dei dati a disposizione.
- Questo approccio fornisce chiaramente una stima ottimistica dell'errore, nel senso che ne è un limite inferiore.

I metodi di *Cross Validation*

- I metodi della classe della *Cross-Validation* (ne esistono molte varianti) si basano sulla convinzione che sia necessario utilizzare due insiemi disgiunti per realizzare e per testare un classificatore.
- Varianti:
 - *Holdout*
 - *Averaged Holdout*
 - *Leave One-Out*
 - *Leave K-Out*

■ *Holdout*

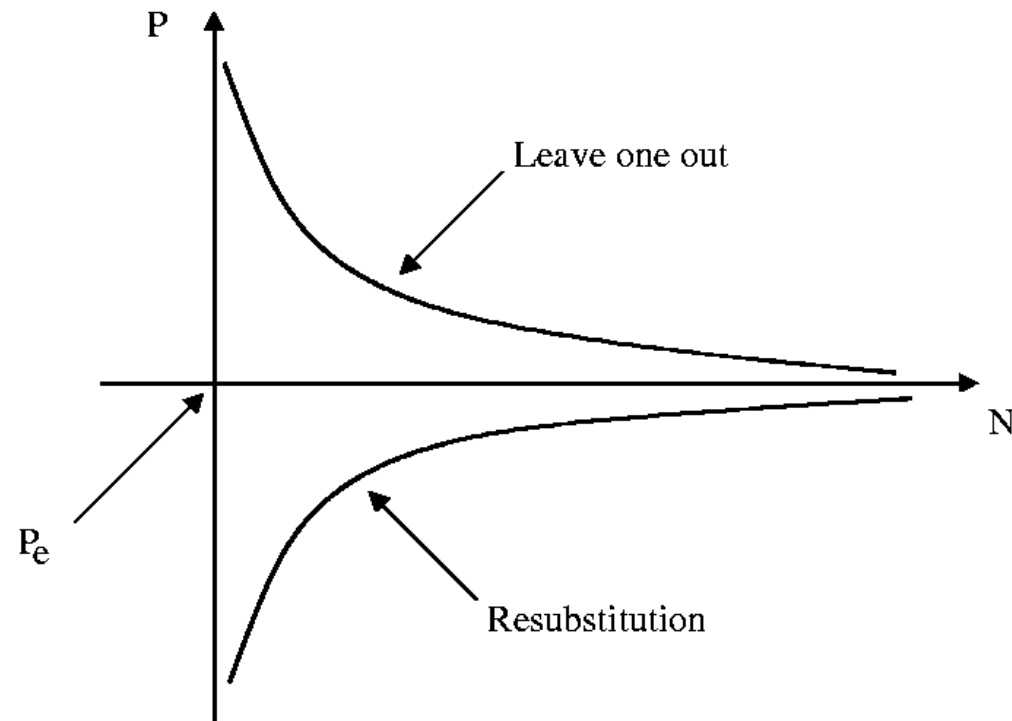
- L'insieme dei dati viene partizionato casualmente in due sottoinsiemi disgiunti di eguale dimensione;
- uno dei due sottoinsiemi viene utilizzato come *Learning Set* e l'altro come *Test Set*;
- questo metodo fornisce una stima superiore dell'errore.

■ *Averaged Holdout*

- per rendere il risultato meno dipendente dalla partizione scelta, si mediano i risultati calcolati su più partizioni holdout;
- le partizioni sono costruite casualmente, oppure in modo esaustivo;
- questo metodo fornisce una stima superiore dell'errore.

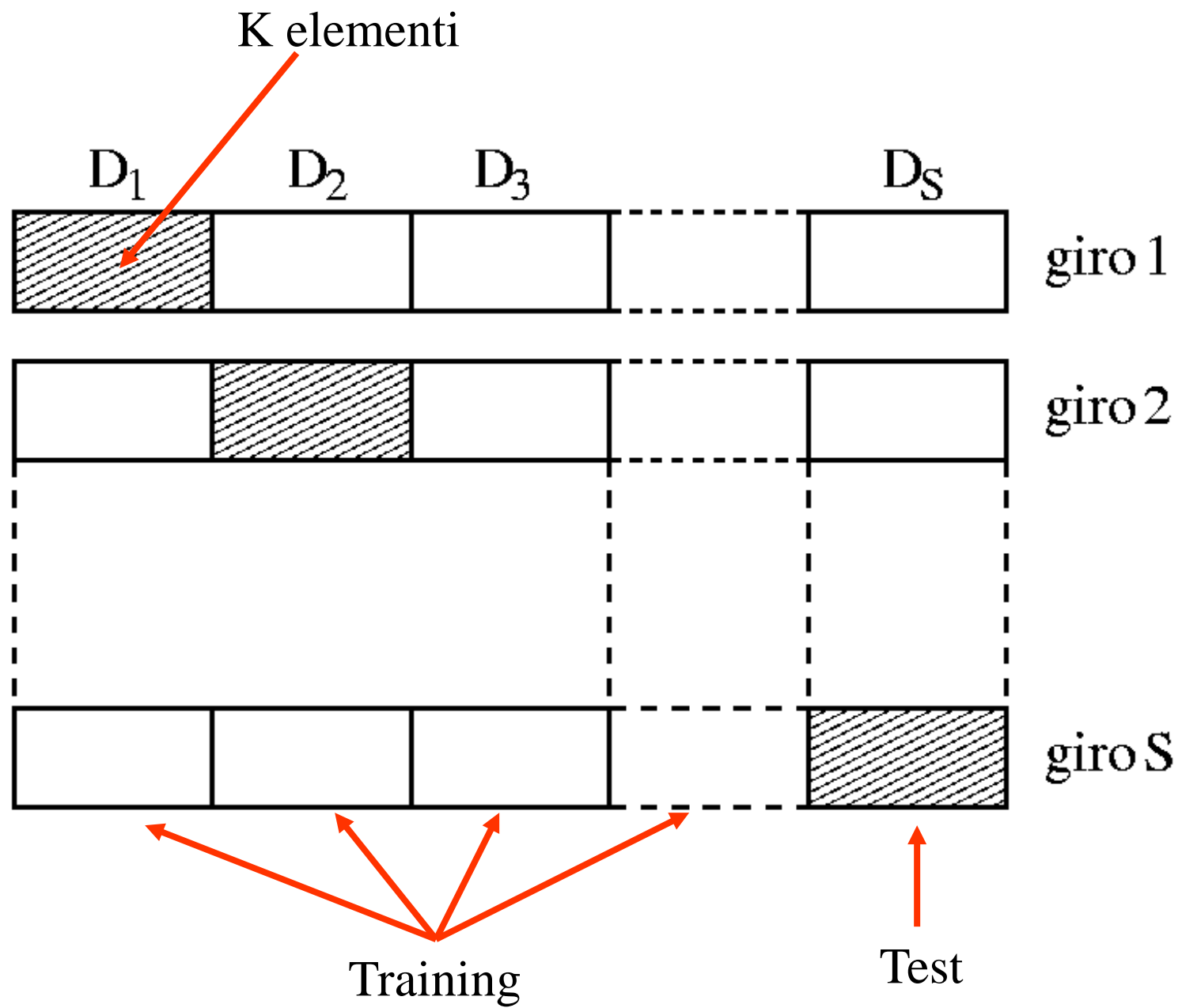
■ *Leave One-Out*

- dato un insieme di dati di cardinalità N , questo metodo ne utilizza $N-1$ per costruire il classificatore, mentre il dato escluso viene usato per testarlo;
- anche qui si effettua una media sui trial;
- questo metodo produce una stima superiore dell'errore.



■ *Leave K-Out:*

- questa tecnica è una generalizzazione della tecnica precedente;
- l'idea è quella di suddividere l'insieme dei dati in S segmenti distinti e casuali;
- si realizza il classificatore utilizzando $S-1$ segmenti, mentre lo si testa utilizzando il segmento rimanente;
- questa operazione viene effettuata S volte, variando a turno il segmento del Test Set;
- infine l'errore viene mediato tra gli S risultati.



Quando le reti neurali sono appropriate

- Se le istanze del problema sono date in coppie (attributi - classe)
 - è necessaria una fase di preprocessing: i valori di input devono essere scalati nel range [0-1], mentre i valori discreti devono essere convertiti in booleani.
- Se gli esempi del training set sono numerosi.
- Se sono accettabili tempi lunghi di addestramento.
- Se non è importante che la funzione determinata sia espressiva per un umano.

Vantaggi e svantaggi

■ Vantaggi:

- è un algoritmo inerentemente parallelo, ideale per hardware multiprocessori
- rappresenta un sistema di classificazione molto potente, utilizzato in innumerevoli contesti

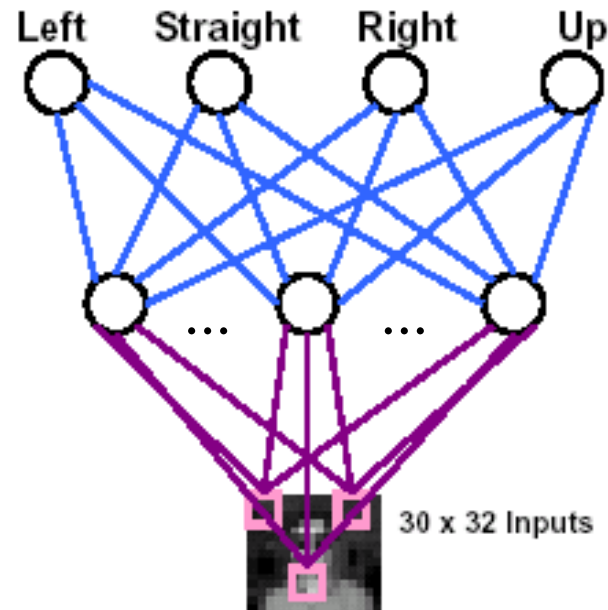
■ Svantaggi:

- determinare la topologia è un'arte
- generalizzazione vs. memorizzazione: con troppi neuroni, la rete tende a memorizzare i pattern e non riesce più a generalizzare
- le reti richiedono un addestramento lungo e oneroso

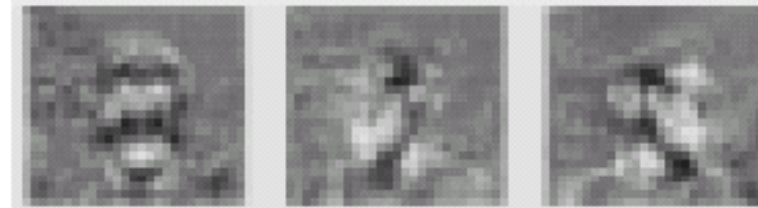
Esempi di applicazioni in Visione

- Riconoscimento di facce e caratteri
- Edge detection
- Guida di veicoli
- ... *molte molte altre*

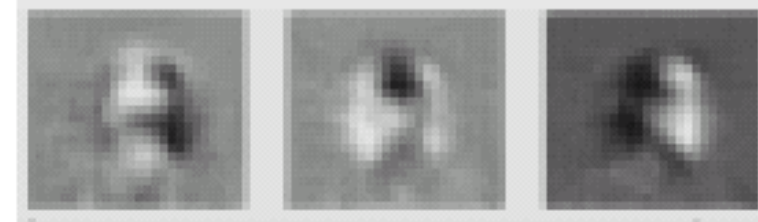
Example: Neural Nets for Face Recognition



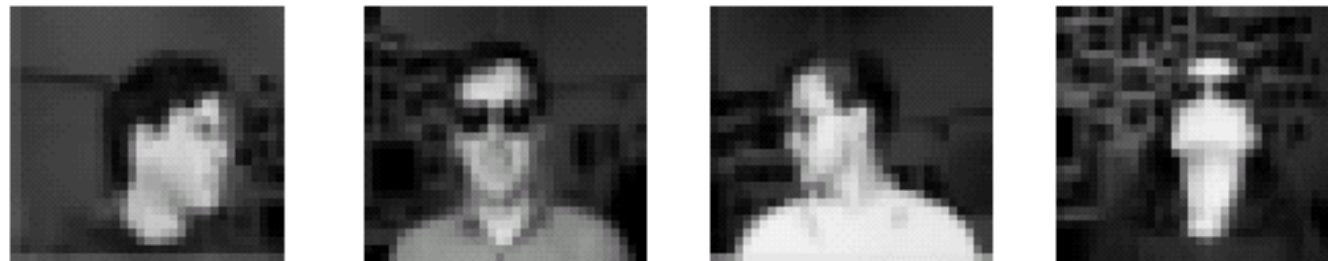
Output Layer Weights (including $w_0 = \theta$) after 1 Epoch



Hidden Layer Weights after 25 Epochs



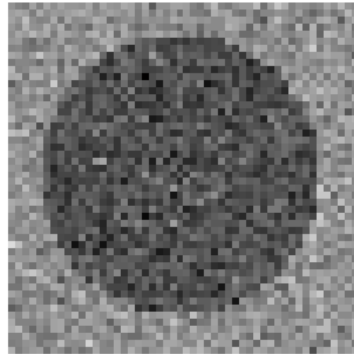
Hidden Layer Weights after 1 Epoch



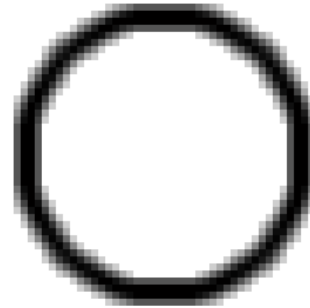
- 90% Accurate Learning Head Pose, Recognizing 1-of-20 Faces
- <http://www.cs.cmu.edu/~tom/faces.html>

Edge detection: addestramento

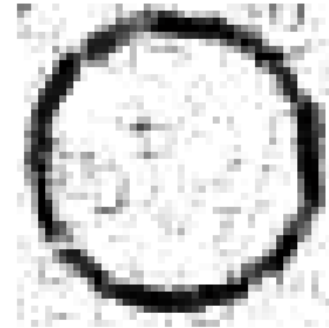
Training set 1



Input

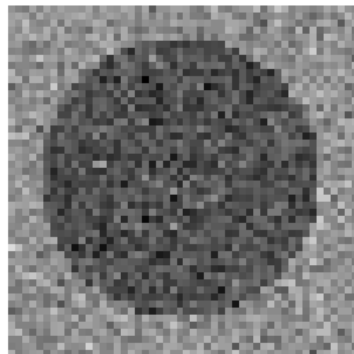


Desired output

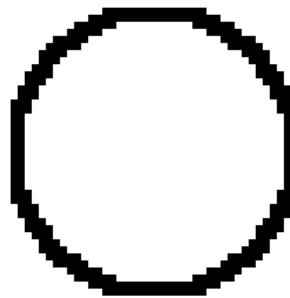


Fitted output

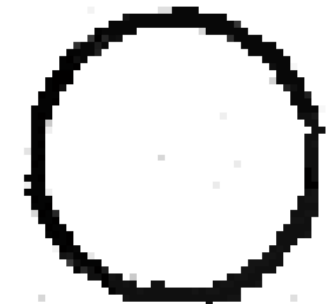
Training set 2



Input



Desired output



Fitted output

Edge detection: risultati



Original image



Dilation-erosion edges



Fitted single sigmoid to
3x3 fields edges



Fitted plane edges

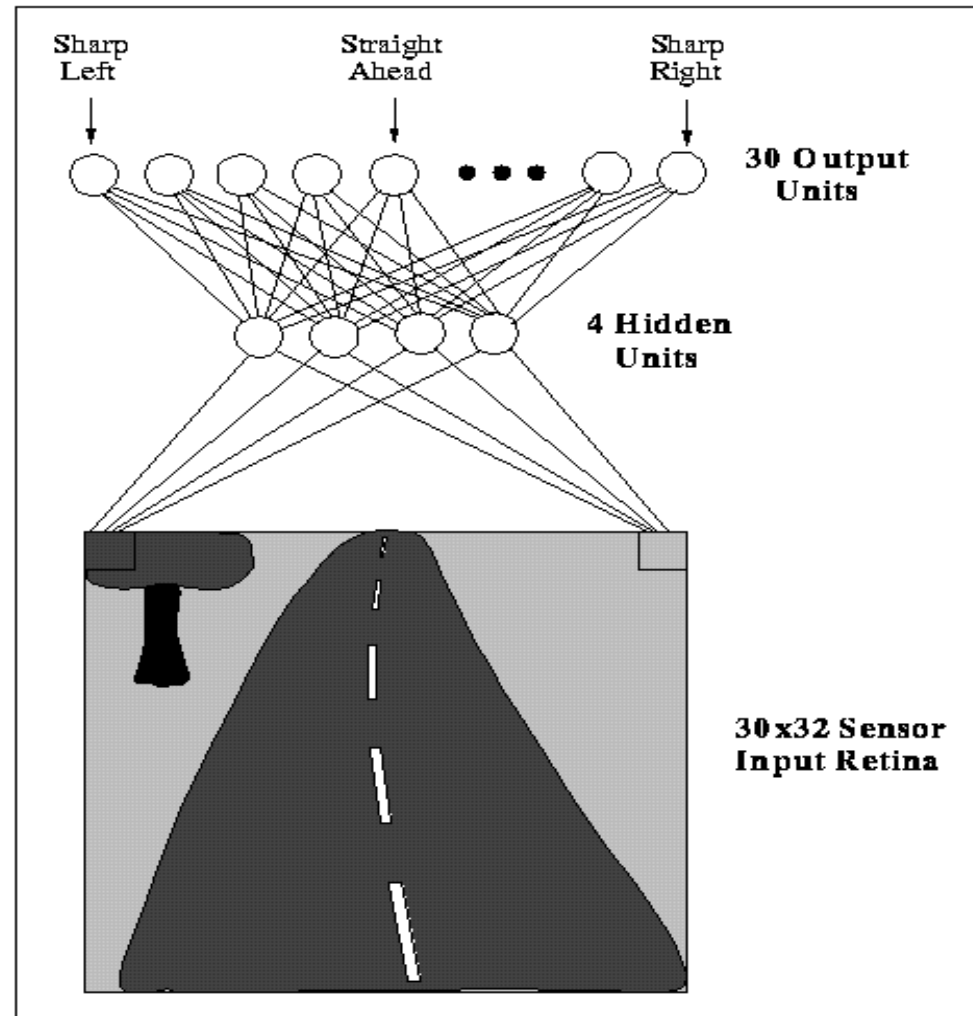


FF ANN, training set 1



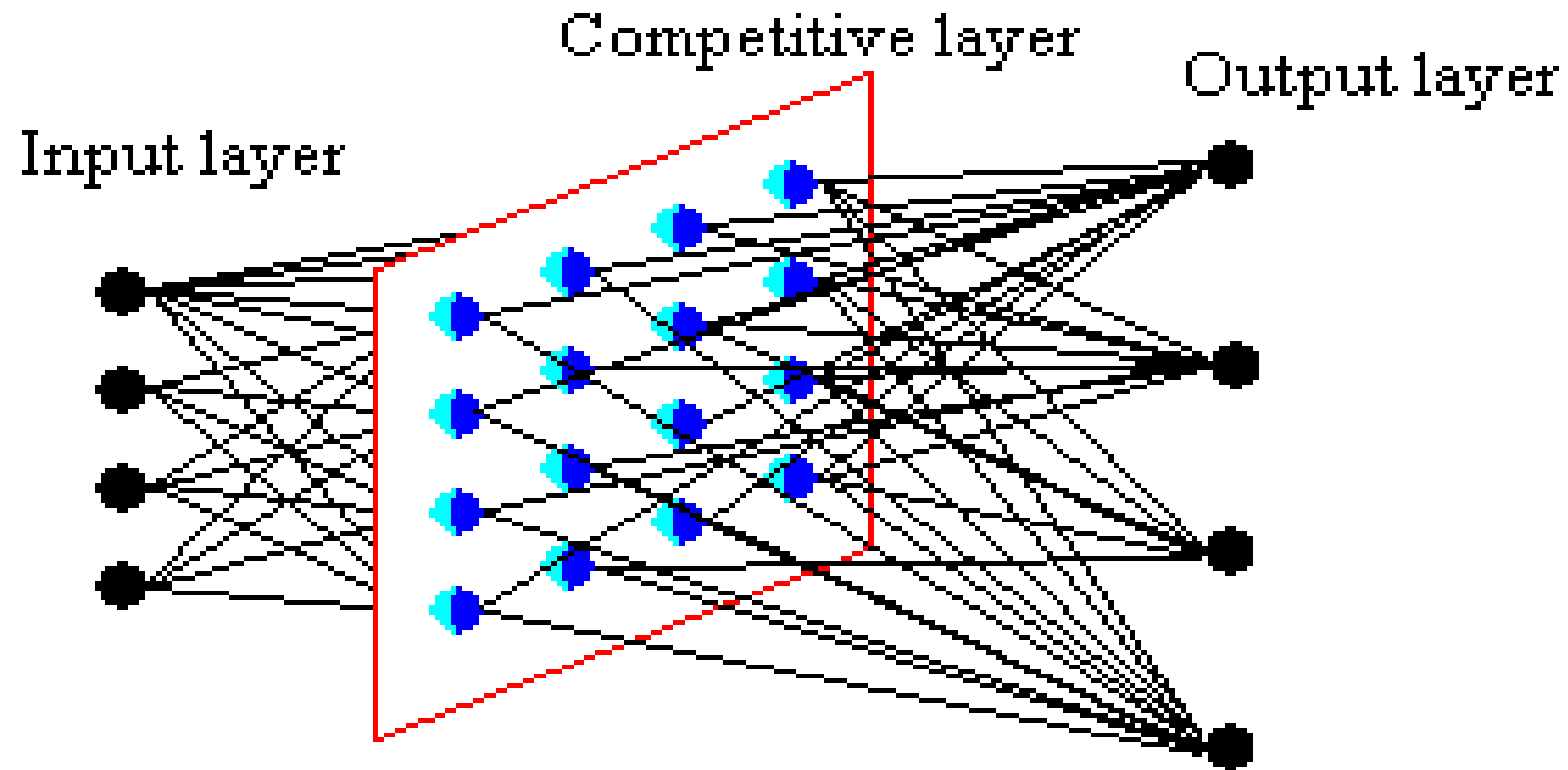
FF ANN, training set 2

Guida di camion (Pomerlau)



Reti di Kohonen

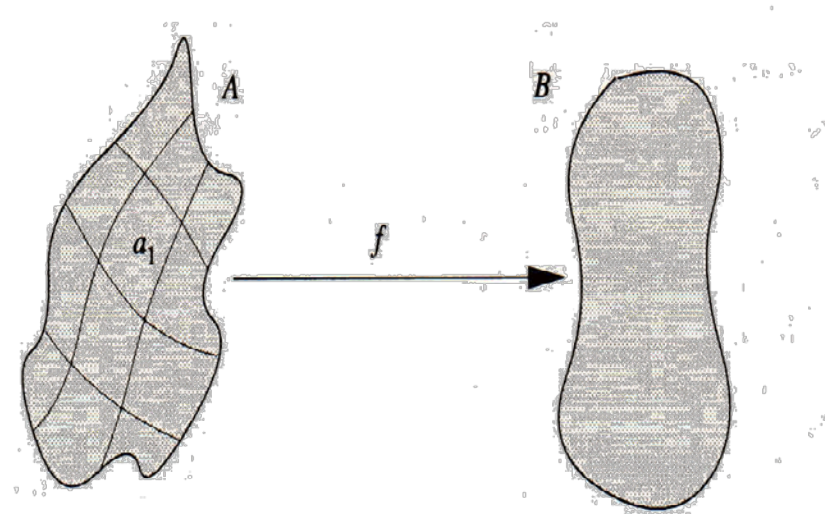
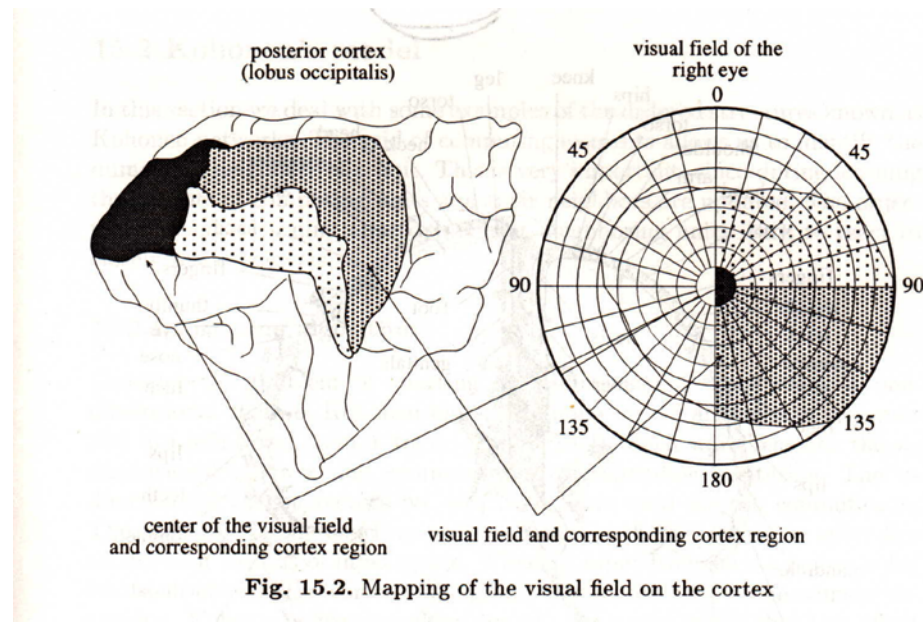
Le reti di Kohonen implementano una *self-organizing feature map*



Self-organizing feature map

- L'esperienza dei sensi è multidimensionale
- Esempio: il suono è caratterizzato dal timbro, dal tono, dall'intensità, dal rumore etc.
- Il cervello mappa la rappresentazione multidimensionale esterna del mondo (con le sue relazioni spaziali) in una rappresentazione interna 1D o 2D (mappa di feature - *feature map*). Esempi:
 - mappa tonotopica: le frequenze del suono sono mappate in regioni della corteccia spazialmente ordinate rispetto alla frequenza;
 - mappa retinotopica: il campo visivo è mappato nella corteccia visiva (lobo occipitale) con una risoluzione che cresce dal centro verso l'esterno del campo visivo, mantenendo relazioni di vicinanza

Modello matematico



- La rete di Kohonen determina una funzione f dallo spazio di input A allo spazio di output B
- Il dominio di f è coperto dalla rete di Kohonen in modo che, quando viene scelto un vettore da una zona prefissata, solo un neurone nella rete si accende.

Addestramento competitivo

- Il livello di input riceve un pattern multidimensionale in ingresso, il vettore $\mathbf{x} = (x_1, \dots, x_n)$
- Il livello competitivo (*competitive layer*) può essere monodimensionale, bidimensionale o n-dimensionale (tipicamente 2D)
- Ogni neurone nel livello competitivo riceve una somma pesata degli ingressi del livello di input

$$a_i = g\left(\sum_j w_{ij} x_j\right) = g\left(\sum_j w_{i,\cdot} x_j\right)$$

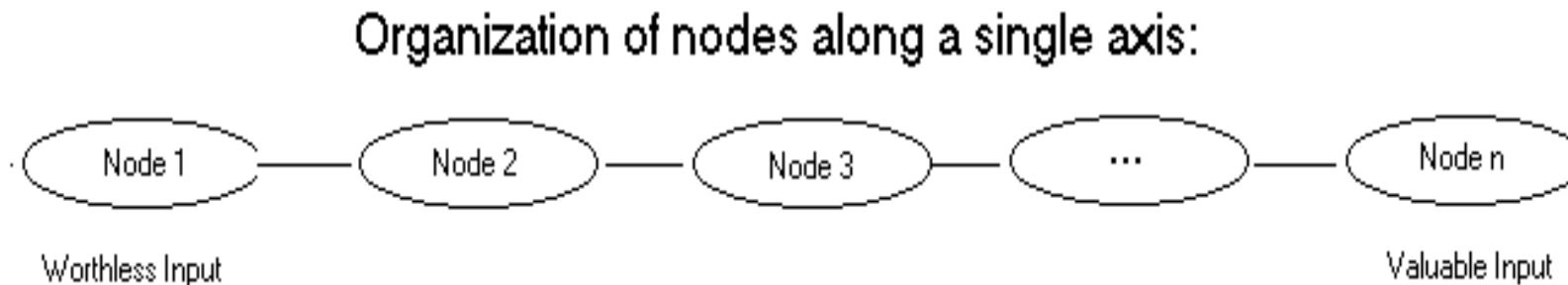
- dove $w_{i,\cdot}$ è la i -esima riga di w , contenente i pesi delle connessioni dal neurone i al livello di input. La funzione di attivazione g è l'identità.
- $w_{i,\cdot}$ e \mathbf{x} devono essere normalizzate

- Ogni neurone del livello competitivo viene associato con un insieme di altri neuroni a formare un insieme di “vicinato”.
- Dopo aver ricevuto un determinato input, alcuni neuroni saranno sufficientemente eccitati da accendersi.
- Il neurone la cui risposta è massima avrà la possibilità di aggiustare i pesi dei neuroni del suo vicinato, compreso se stesso.
- Sia k il neurone vincitore; per ogni neurone i nel suo vicinato

$$w_{i,.} \Leftarrow w_{i,.} + \eta (x - w_{i,.})$$

Livello di uscita

- L'organizzazione del livello di uscita dipende strettamente dall'applicazione.
- Per un corretto funzionamento della rete di Kohonen non è necessario che sia presente.
- Se i nodi sono dislocati lungo un'unica dimensione, l'uscita può essere interpretata come un continuo



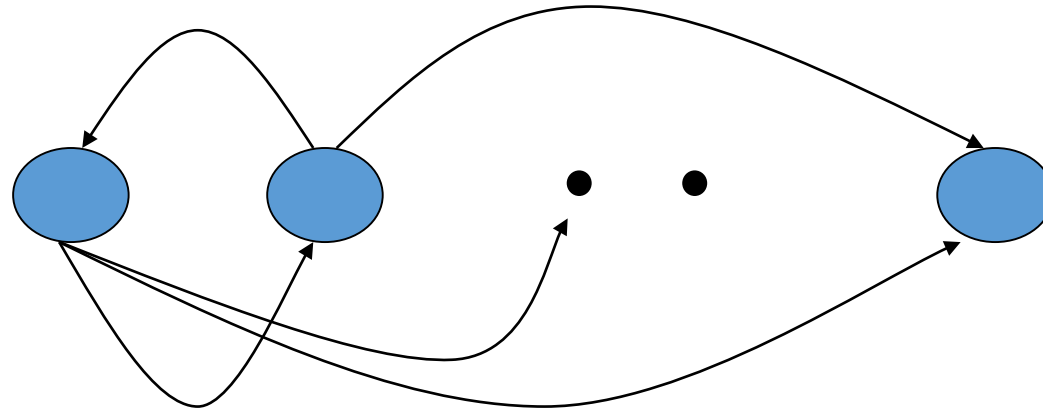
Vantaggi

- Ottimo per problemi di clustering
- Può ridurre la complessità computazionale di un problema
- Molto sensibile a input frequenti
- Può creare nuovi modi per scoprire associazioni tra dati
- Non necessita di supervisione

Svantaggi

- Il sistema è una scatola nera
- Non viene garantita la convergenza della rete per reti di dimensione maggiore a due
- Molti problemi sono difficilmente rappresentabili con una SOFM
- Viene richiesto un insieme di addestramento grande
- Le associazioni sviluppate dalla SOFM non sempre sono facilmente interpretabili

Reti di Hopfield



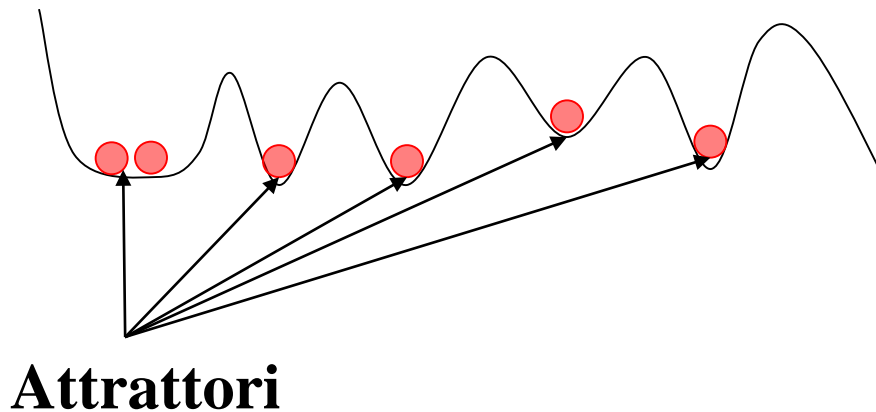
- Rete ricorrente (tutti connessi con tutti), un solo strato.
- Connessioni simmetriche: $w_{ij} = w_{ji}$, $w_{ii} = 0$
- Funzione di attivazione a soglia:
$$x_i(t+1) = a_i = \text{step}(\sum_{j \neq i} w_{ij} x_j(t))$$
- In ogni istante di tempo, i valori di attivazione di tutti i neuroni definiscono lo stato della rete $x(t) = (x_1(t) \dots x_N(t))$
- Man mano che i neuroni applicano la regola di aggiornamento, lo stato della rete evolve.

Attrattori

- In analogia ai sistemi fisici si definisce una funzione **energia** per la rete:

$$E(x) = - \frac{1}{2} \sum_{i \neq j} w_{ij} x_i x_j$$

- Si dimostra che l'aggiornamento dello stato di un neurone non può che decrementare l'energia (funzione di Lyapunov).
- Quindi la rete evolve verso uno stato di minima energia, un **attrattore**.
- Gli attrattori sono **pattern memorizzati** che le rete recupera quando viene inizializzata "vicino" all'attrattore



Nota: con aggiornamento asincrono casuale si hanno solo punti fissi, altrimenti si potevano avere cicli

Apprendimento Hebbiano

- I pesi delle connessioni determinano gli attrattori.
- Come fissare i pesi in modo che la rete memorizzi un insieme dato di pattern $A^1 \dots A^M$?

- Regola Hebbiana:

$$w_{ij} = \sum_m (2A_i^m - 1) (2A_j^m - 1)$$

- Vengono rinforzate le connessioni tra neuroni che sono contemporaneamente attivi in un dato pattern, indebolite le altre.
- Nel caso più favorevole la rete di Hopfield è capace di apprendere un numero di pattern pari al 10% del numero di neuroni.

Esempio

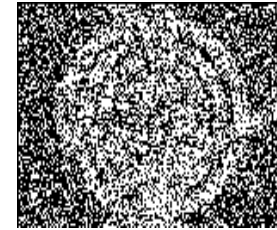
Addestramento

2 immagini memorizzate
nella rete



Funzionamento

La rete inizia con questa
immagine



Stato stabile (attrattore)

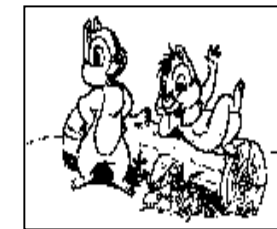
Immagine ricostruita



Altri esempi:



input



output

Source:

<http://www2.psy.uq.edu.au/~brainwav/Manual/Hopfield.html>

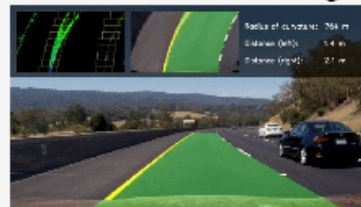
Deep learning applications



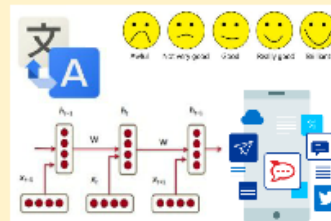
Social Network Analysis



Autonomous Driving



Natural Language Processing



Sentiment Classification
Entity Extraction
Translation

Visual Data Processing

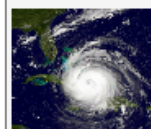


Computer Vision
Multimedia Data Analysis

Biomedicine



Disaster



Speech and Audio Processing



Speech Enhancement
Speech Recognition

Information
Retrieval



Deep learning applications

Aerospace, Defense and Communications



Communications devices,
security



Multi-standard communications
receivers, drone recognition

Consumer Electronics and Digital Health



Voice assistants



Digital health

Automotive



Voice control enabled
Infotainment



Sensor processing,
automated driving

Industrial Automation

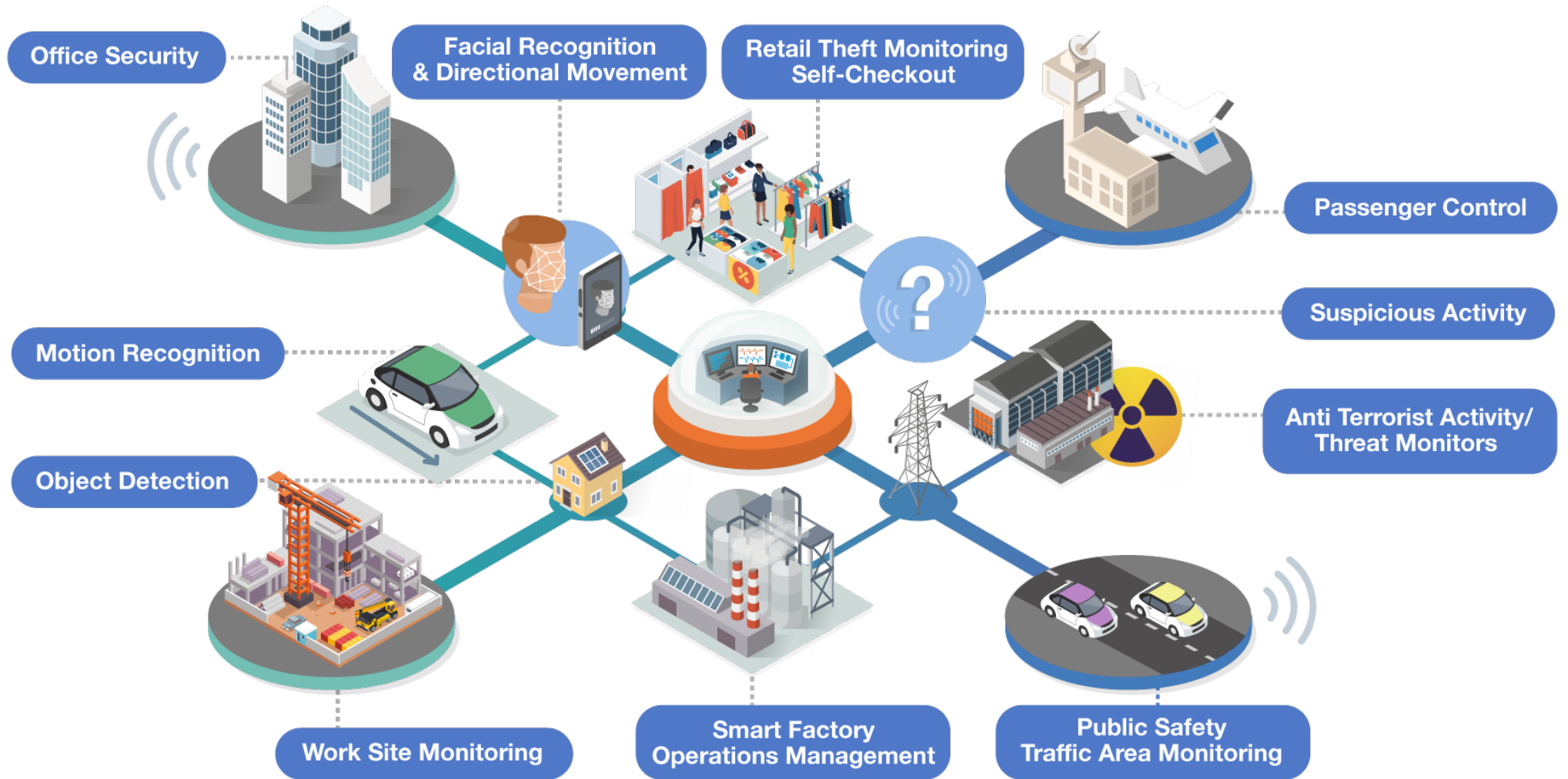


Condition monitoring



Predictive maintenance

Deep learning applications



Deep learning applications



ADAS



Drones



Mobile



Surveillance



Augmented Reality



Gaming



Cameras



Infotainment



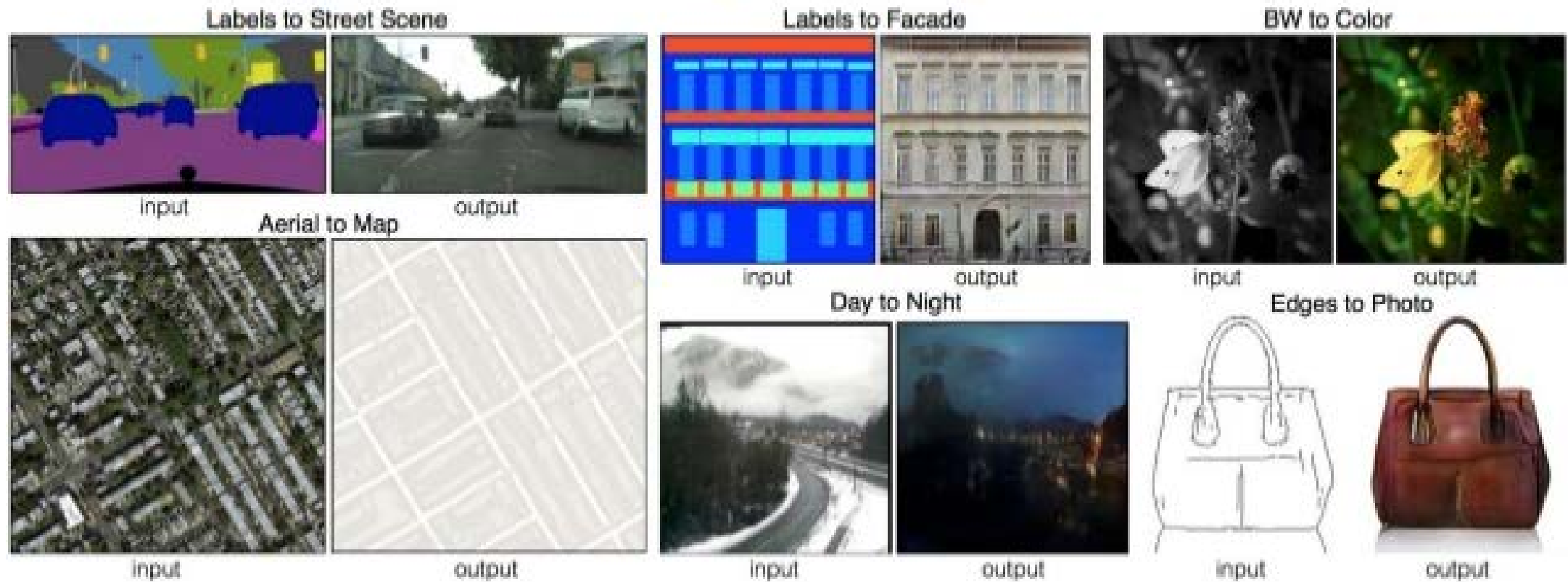
Retail



Home Automation

Deep learning applications

GAN's application: Image to Image Translation



(Isola et al, 2016)

37

Deep learning applications

Monet ↔ Photos



Monet → photo

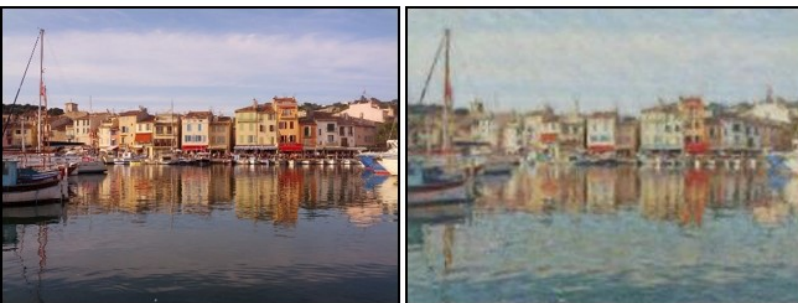
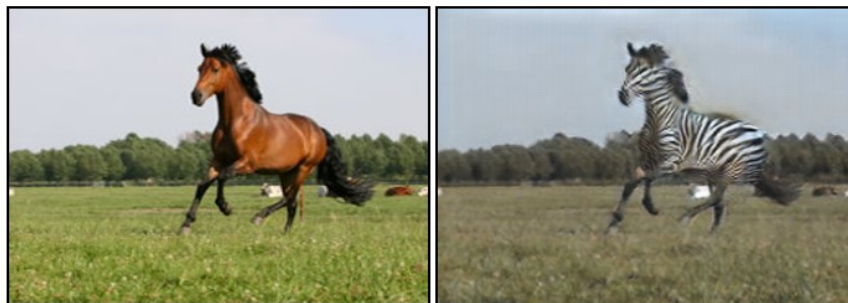


photo → Monet

Zebras ↔ Horses



zebra → horse



horse → zebra

Summer ↔ Winter



summer → winter



winter → summer



Photograph



Monet



Van Gogh



Cezanne



Ukiyo-e

Deep learning applications



