



Intelligenza Artificiale

Reti neurali

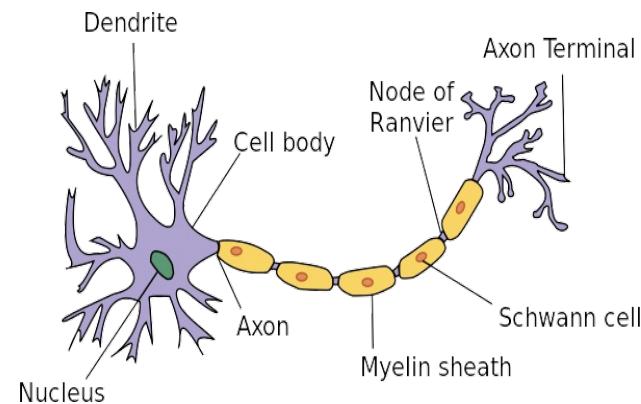


Outline

- ▶ Neuroni
- ▶ Reti neurali
- ▶ Percettrone
- ▶ Percettrone multistrato
- ▶ Backpropagation

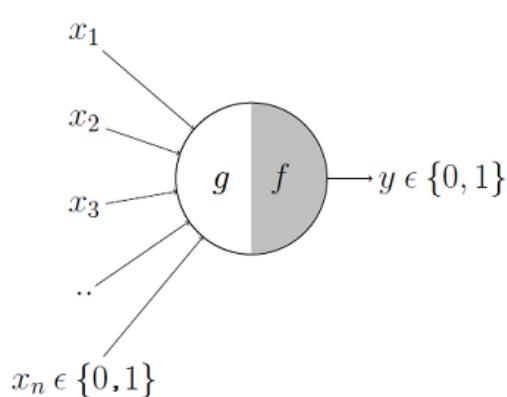
Neuroni

- ▶ L'idea delle reti neurali è nata come modello di come funzionano i neuroni nel cervello.
- ▶ I circuiti collegati sono stati utilizzati per simulare il suo comportamento intelligente.
- ▶ Il cervello è composto da neuroni.
 - ▶ un corpo cellulare
 - ▶ dendriti (ingressi)
 - ▶ un assone (uscite)
 - ▶ sinapsi
- ▶ Le **sinapsi** possono essere stimolate o inibite e possono cambiare nel tempo.
- ▶ Quando la somma degli ingressi raggiunge una certa soglia, verrà inviato un impulso elettrico sull'assone.



Reti neurali – McCulloch-Pitts Unit

- ▶ Una **rete neurale artificiale** è un grafo diretto di **unità** e **collegamenti**. Un link dall'unità i all'unità j propaga l'**attivazione a_i** , dall'unità i all'unità j , ed ha un peso $w_{i,j}$ ad essa associato.
- ▶ Nel 1943 McCulloch e Pitts proposero un modello semplice per un neurone.
- ▶ Un'**unità McCulloch Pitts** calcola prima una somma su tutti gli input booleani e poi applica una **funzione di attivazione g** .



$$g(x_1, x_2, x_3, \dots, x_n) = g(\mathbf{x}) = \sum_{i=1}^n x_i$$

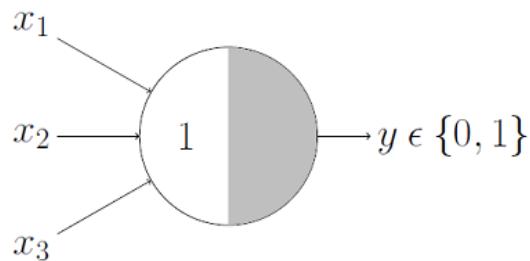
$$\begin{aligned} y = f(g(\mathbf{x})) &= 1 & \text{if } g(\mathbf{x}) \geq \theta \\ &= 0 & \text{if } g(\mathbf{x}) < \theta \end{aligned}$$

- ▶ Una **rete McCulloch Pitts** è una **rete neurale** con unità McCulloch Pitts.

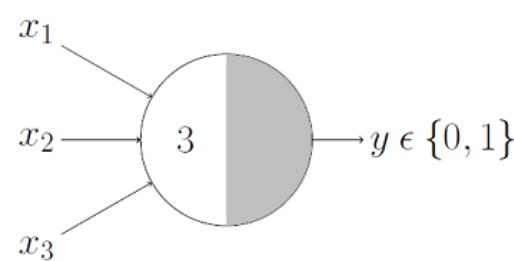
Implementazione di funzioni logiche come unità

- Le unità di McCulloch Pitts sono una grossolana semplificazione eccessiva dei neuroni reali, ma il suo scopo è sviluppare la comprensione di ciò che possono fare le reti neurali di unità semplici.
- Ogni funzione booleana può essere implementata come reti McCulloch Pitts.

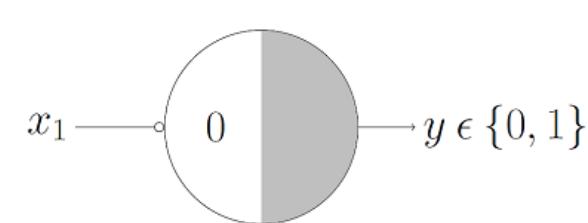
▶ OR



AND



NOT

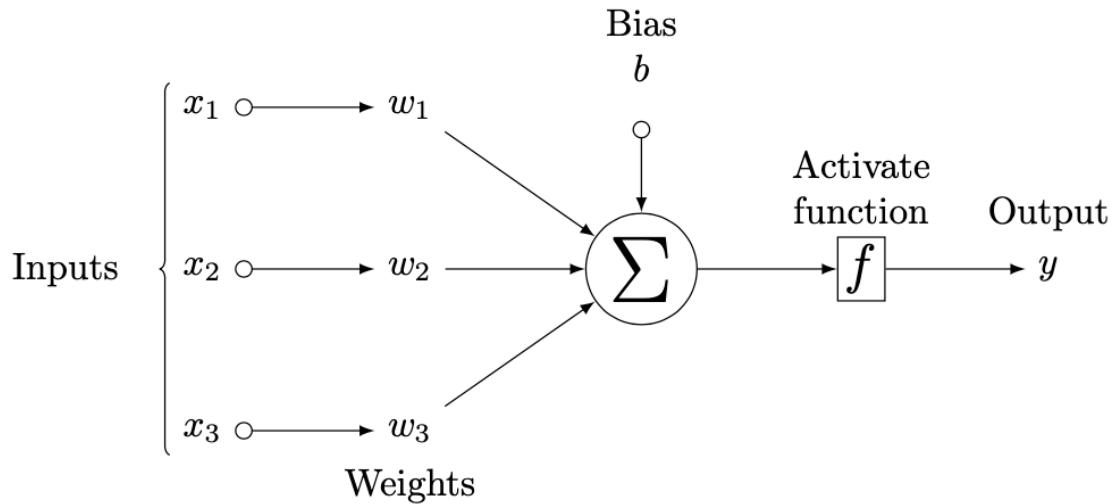


Percettrone

- ▶ Problemi con i neuroni di McCulloch e Pitts
 - ▶ I pesi e le soglie sono determinati analiticamente (non è possibile apprenderli).
 - ▶ Molto difficile ridurre al minimo le dimensioni di una rete.
 - ▶ Che dire dei task non discreti e/o non binari?
- ▶ Soluzione del **percettrone**.
 - ▶ I pesi e le soglie possono essere determinati analiticamente o mediante un algoritmo di apprendimento.
 - ▶ Versioni continua, bipolare e multivale.
 - ▶ Rosenblatt ha collegato casualmente i percetroni e ha cambiato i pesi per effettuare apprendimento.
 - ▶ Esistono euristiche di minimizzazione efficienti.

Percettrone

- Il Percettrone ha la seguente architettura.



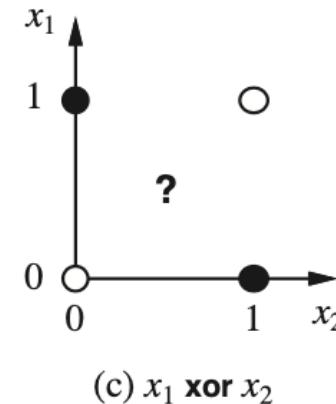
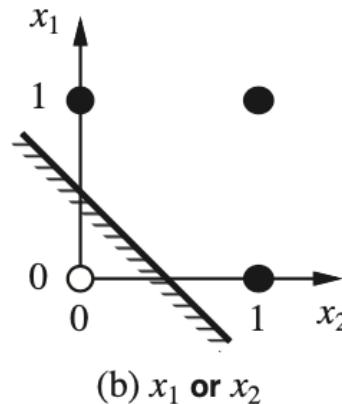
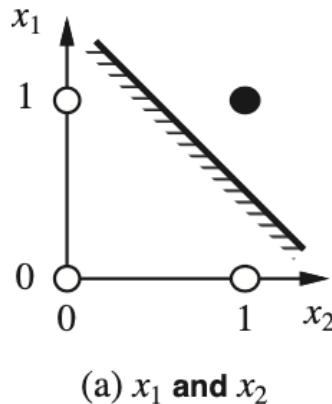
- Sia t l'output corretto e y la funzione di output della rete.
- Il percettrone aggiorna i pesi (Rosenblatt 1960)
- Il percettrone è l'elemento costitutivo delle moderni reti neurali.

$$w_j^{(t)} \leftarrow w_j^{(t)} + \alpha x_j(t - y)$$

Espressività dei percettroni

- ▶ Un **perceptron** può rappresentare AND, OR, NOT, maggioranza, ecc., ma non XOR (e quindi nessun sommatore)
- ▶ Rappresenta un **separatore lineare** nello spazio di input:

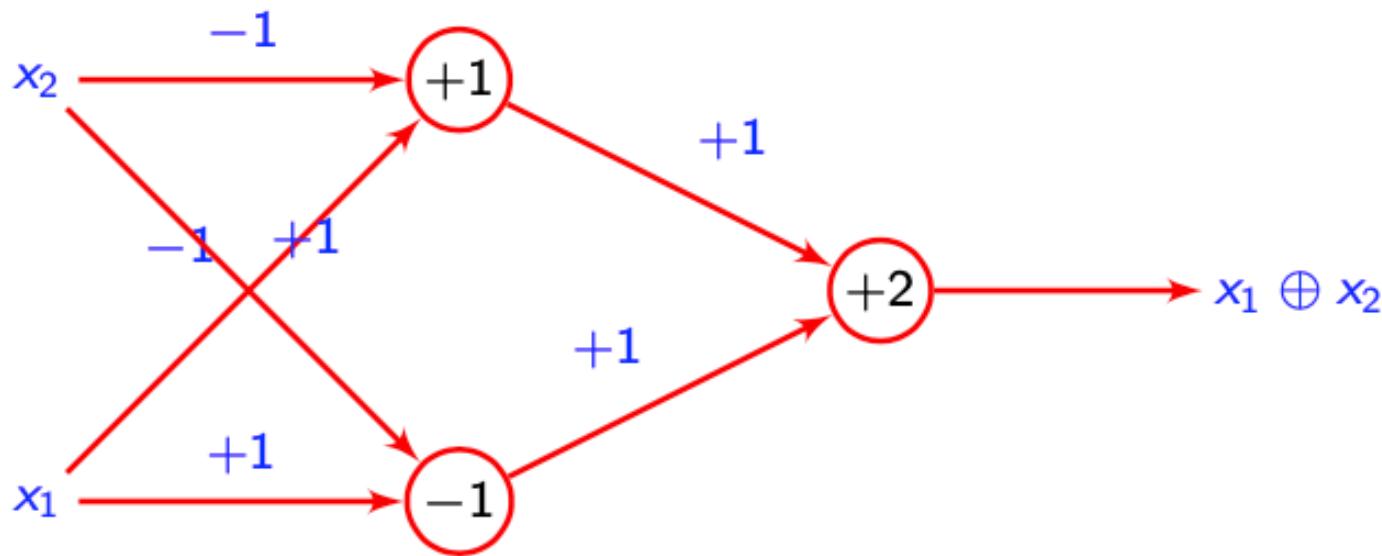
$$\sum_j w_j x_j > 0 \quad \text{or} \quad W \cdot x > 0$$



- ▶ Minsky e Papert (1969) hanno avviato l'interesse dei ricercatori sulle reti neurali!

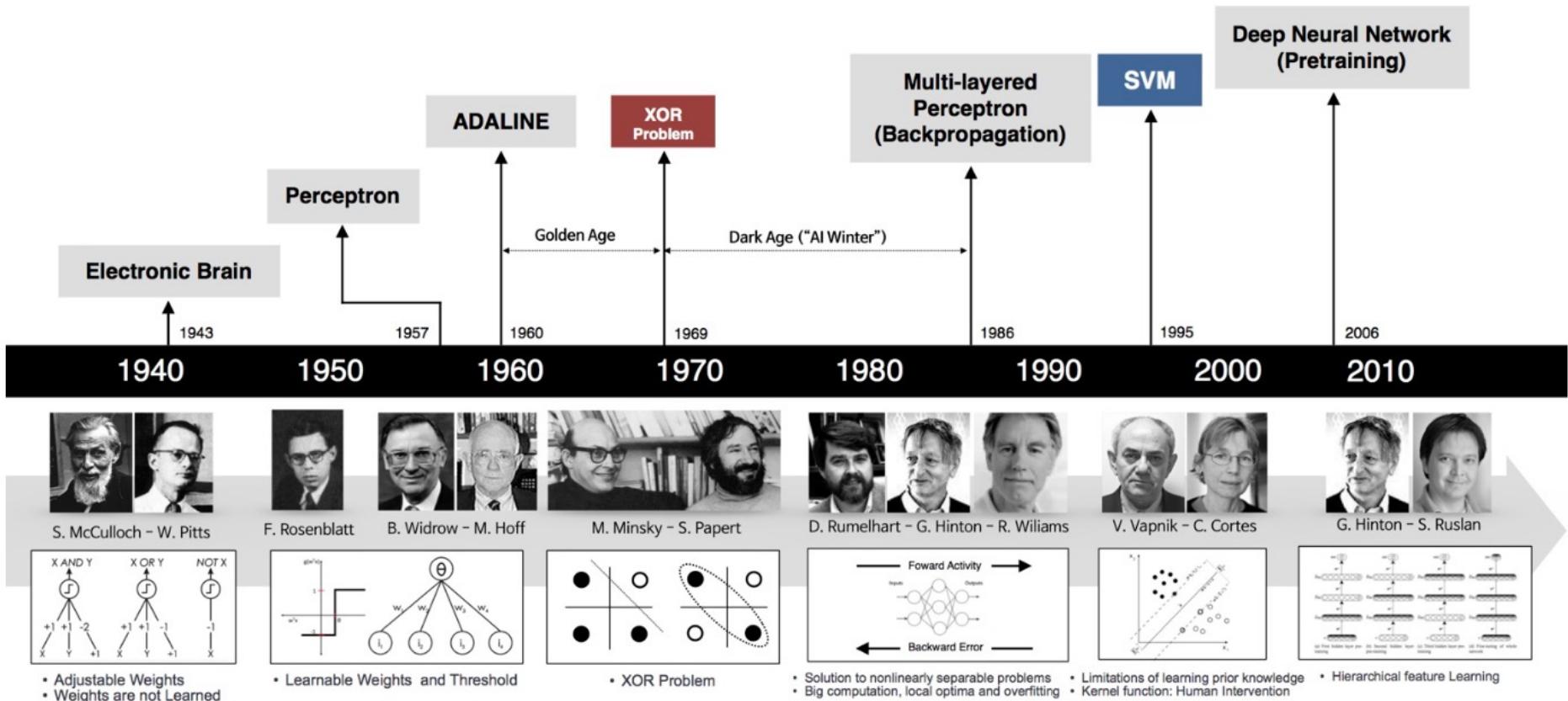
Espressività dei percettroni

- Il seguente Perceptron multistrato può risolvere il problema XOR.



- Lo strato intermedio è uno strato nascosto.

Storia delle reti neurali



Strutture di rete: reti feed-forward

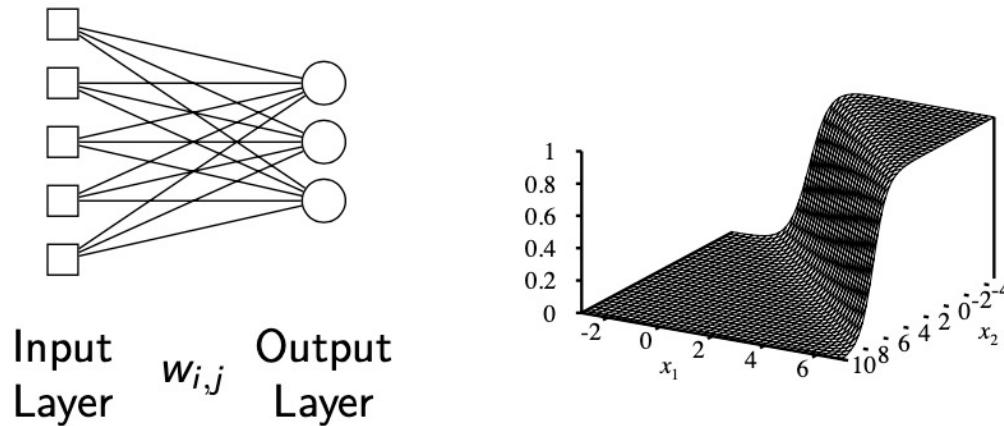
- ▶ Abbiamo modelli per i **neuroni** che li connettono alle **reti neurali**.
- ▶ Una rete neurale è chiamata **rete feed forward**, se la rete è **aciclica**.
- ▶ **Intuizione**: le reti feed forward implementano funzioni, non hanno stati interni.
- ▶ Le reti **feed forward** sono generalmente organizzate in livelli (**layers**): una **rete a n -livelli** ha una partizione $\{L_0, \dots, L_n\}$ dei nodi, in modo tale che gli archi colleghino solo i nodi dal livello successivo.
- ▶ L_0 è chiamato **livello di input** ed i suoi elementi **unità di input**, e L_n **livello di output** ed i suoi elementi **unità di output**. Qualsiasi **unità** che non si trova nel livello di input o nel livello di output viene chiamata **hidden**.

Strutture di rete: reti ricorrenti

- ▶ Una rete neurale si chiama **ricorrente**, se ha dei **cicli**.
 - ▶ Le **reti di Hopfield** hanno pesi simmetrici ($w_{i,j} = w_{j,i}$) $g(x) = \text{sign}(x)$, $a_i = \pm 1$; (memoria associativa olografica)
 - ▶ Le macchine Boltzmann utilizzano funzioni di attivazione stocastica.
- ▶ Le reti neurali ricorrenti hanno cicli diretti con ritardi → hanno uno stato interno (come i flip-flop), possono oscillare ecc.

Percetroni single-layer

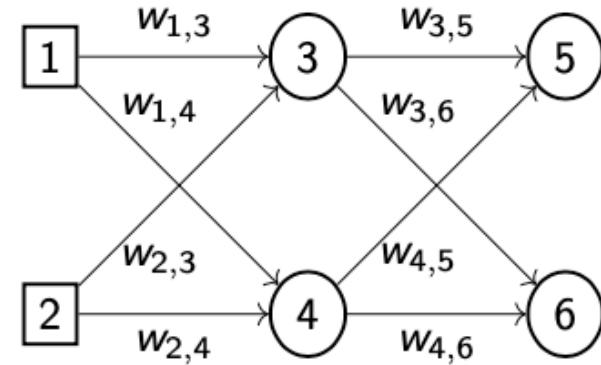
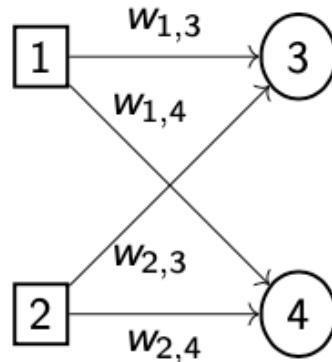
- ▶ Una **rete perceptron** è una **rete feed forward** di unità perceptron. Una rete perceptron a strato singolo è chiamata **perceptron**.



- ▶ Tutte le **unità di ingresso** sono collegate direttamente alle unità di uscita.
- ▶ Le **unità di output** funzionano tutte separatamente, nessun peso condiviso → trattate come il combinazione di n unità percetroni.
- ▶ La regolazione dei pesi sposta la posizione, l'orientamento e la pendenza della **scogliera**.

Reti Neurali Feed-Forward (Esempio)

- ▶ Feed forward network = una famiglia parametrizzata di funzioni non lineari:
- ▶ Mostriamo due reti feed forward:



a) single-layer (perceptron network) b) 2-layer feed forward network

$$\begin{aligned}a_5 &= g(w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4) \\&= g(w_{3,5} \cdot g(w_{1,3} \cdot a_1 + w_{2,3} \cdot a_2) + w_{4,5} \cdot g(w_{1,4} \cdot a_1 + w_{2,4} \cdot a_2))\end{aligned}$$

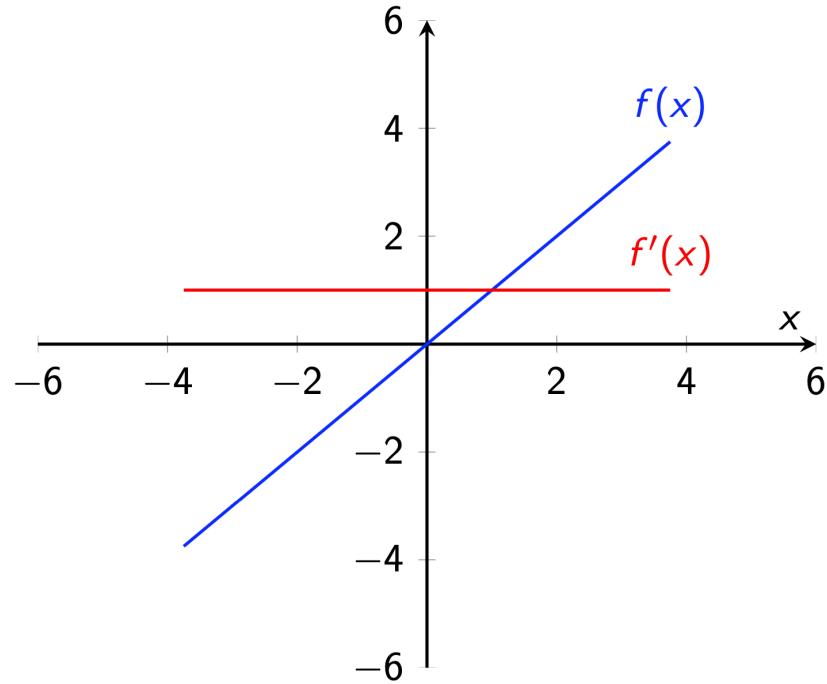
- ▶ Idea: la regolazione dei pesi cambia la funzione: apprende così!

Funzione di attivazione

- ▶ Una funzione di attivazione viene aggiunta a una rete neurale per aiutarla ad apprendere schemi complessi nei dati.
- ▶ Una funzione di attivazione converte l'output di un neurone in un'altra forma utilizzata come input per il neurone successivo.
- ▶ Perché sono necessarie funzioni di attivazione?
 - ▶ Utilizzato per mantenere l'output di un neurone a un certo intervallo secondo il nostro requisito.
 - ▶ Utilizzato per aggiungere non linearità a una rete neurale.
- ▶ Caratteristiche desiderabili di una funzione di attivazione
 - ▶ **Gradiente non nullo** Formiamo reti neurali utilizzando algoritmi basati su gradiente. Quindi, il gradiente deve essere non zero in tutti i punti di dominio. Se il gradiente tende a zero il problema è chiamato problema del vanishing gradient.
 - ▶ **Centrato sullo zero** L'uscita di una funzione di attivazione deve essere simmetrica a zero. Questo impedisce ai gradienti di spostarsi in una direzione particolare.
 - ▶ **Costi computazionali** Le funzioni di attivazione vengono applicate più volte. Dovrebbero essere computazionalmente poco costose per calcolarle insieme alla sua derivata.
 - ▶ **Differenziabile** Nell'apprendimento basato sul gradiente, dobbiamo calcolare il gradiente di attivazione funzioni. Pertanto, le funzioni di attivazione devono essere differenziabili.

Funzione di attivazione lineare

$$f(x) = x$$
$$f'(x) = 1$$



Vantaggi

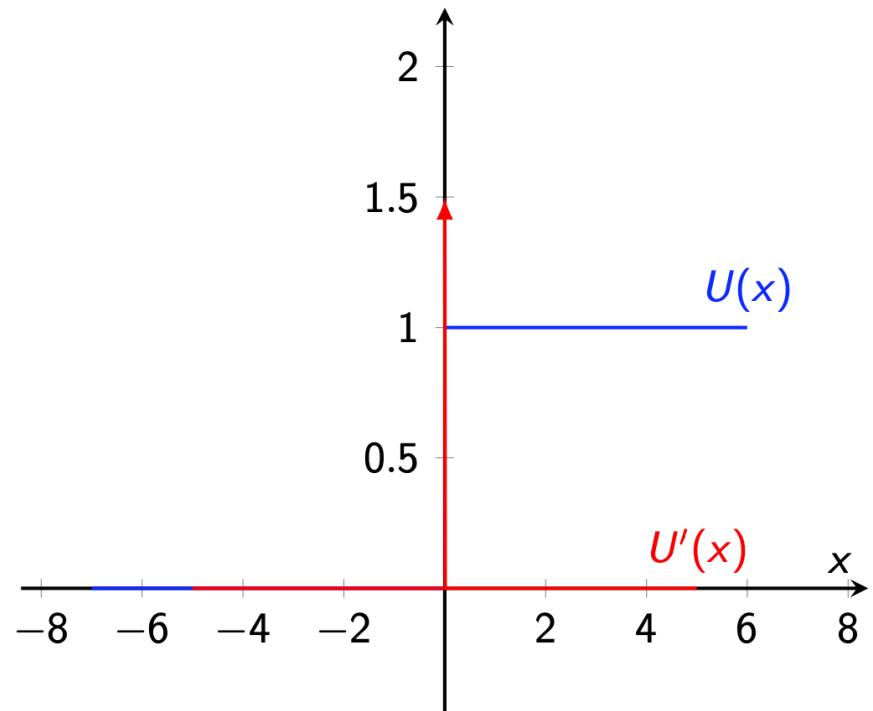
- ▶ Il suo calcolo è semplice.
- ▶ La sua derivata è diversa da zero.

Svantaggi

- ▶ L'output non è in un intervallo.
- ▶ Nessun risultato nella non linearità.

Funzione di attivazione soglia

$$U(x) = \begin{cases} 0 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}$$
$$U'(x) = \begin{cases} 0 & \text{if } x > 0 \\ \delta(0) & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases}$$



Vantaggi

- ▶ Il suo calcolo è semplice.

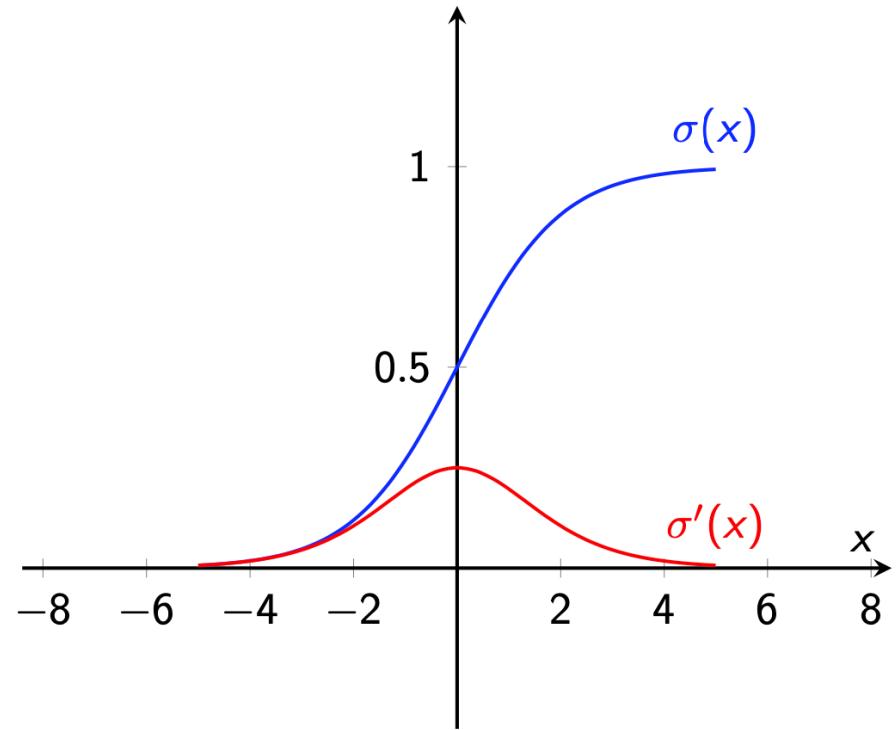
Svantaggi

- ▶ L'output non è in un intervallo.
- ▶ La sua derivata è zero tranne che all'origine.

Funzione di attivazione sigmoide

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$



Vantaggi

- ▶ L'uscita è nell'intervallo [0, 1].
- ▶ È differenziabile.

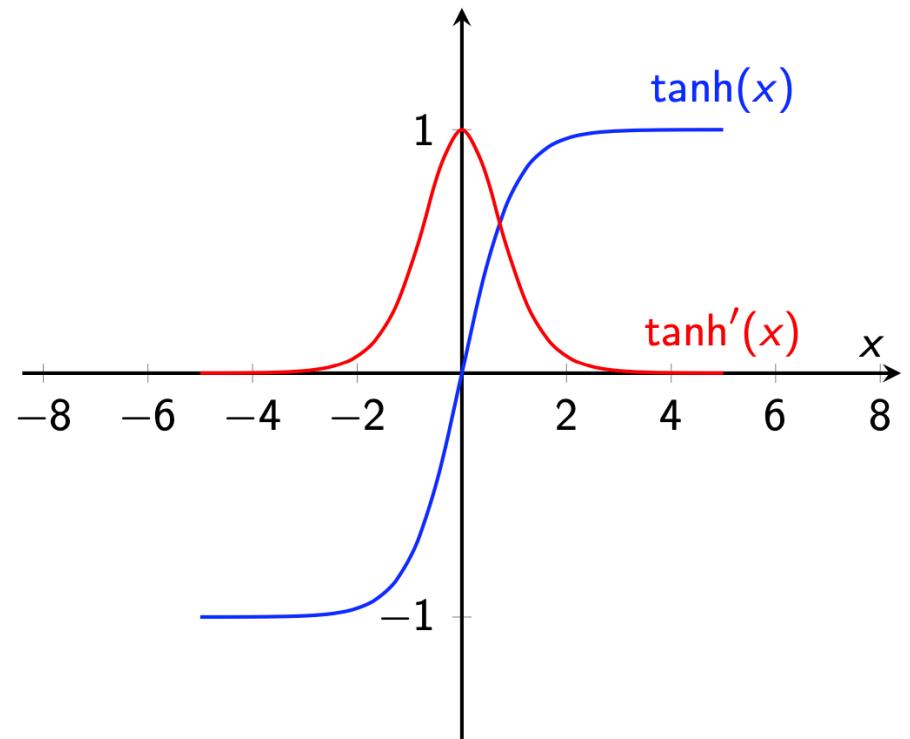
Svantaggi

- ▶ Satura e uccide i gradienti.
- ▶ Il suo output non è centrato sullo zero.

Funzione di attivazione Tanh

$$\tanh(x) = 2\sigma(2x) - 1$$

$$\tanh'(x) = 1 - (\tanh(x)^2)$$



Vantaggi

- ▶ L'output è nell'intervallo [0, 1].
- ▶ È differenziabile.
- ▶ Il suo output è centrato sullo zero.

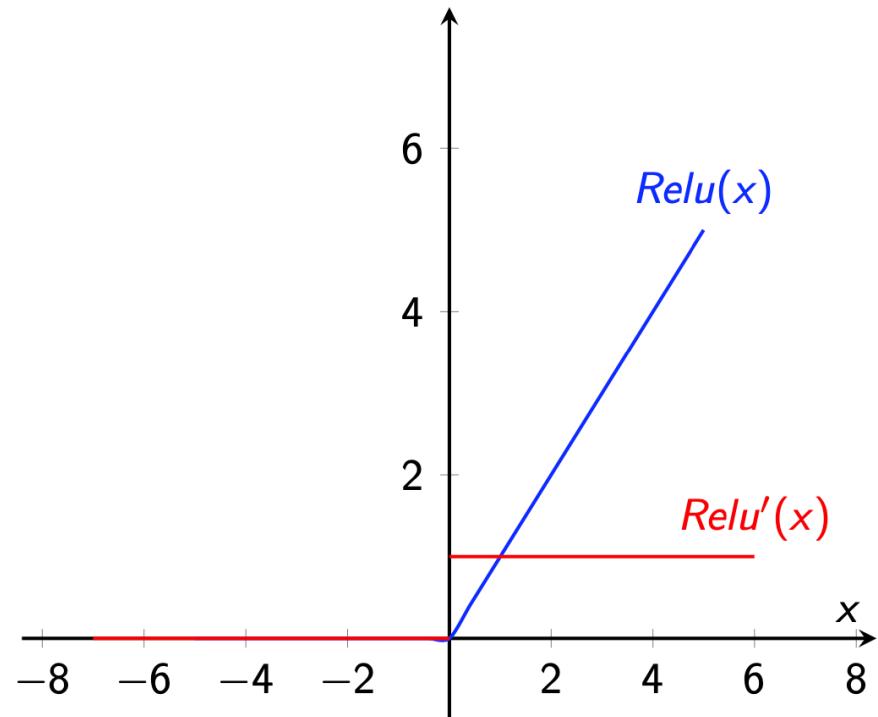
Svantaggi

- ▶ Satura e uccide i gradienti.

Funzione di attivazione Relu

$$Relu(x) = \max(0, x)$$

$$Relu'(x) = \begin{cases} 1 & \text{if } x > 0 \\ ? & \text{if } x = 0 \\ 0 & \text{if } x < 0 \end{cases}$$



Vantaggi

- ▶ Il suo calcolo è semplice.
- ▶ La sua derivata è diversa da zero quando $x > 0$.
- ▶ Non è computazionalmente costosa.

Svantaggi

- ▶ L'output non è in un intervallo.
- ▶ Il gradiente svanisce quando $x < 0$.

Funzione di attivazione Softmax

- ▶ Softmax è una forma più generalizzata del sigmoide.
- ▶ Softmax viene utilizzato nel livello di output delle reti neurali per problemi di classificazione multclasse.

$$a_i(x) = \frac{\exp^{z_i}}{\sum_k \exp^{z_k}}$$

- ▶ Sia $x = [1.60, 0.55, 0.98]^\top$.
- ▶ Applicando softmax si ottiene $a_i = [0.51, 0.18, 0.31]^\top$.

Funzione di loss

- ▶ L'obiettivo degli algoritmi di machine learning è costruire un modello (ipotesi) che possa essere utilizzato per stimare t in base a x .
- ▶ Sia il modello in forma di

$$h(x) = w_0 + w_1 x$$

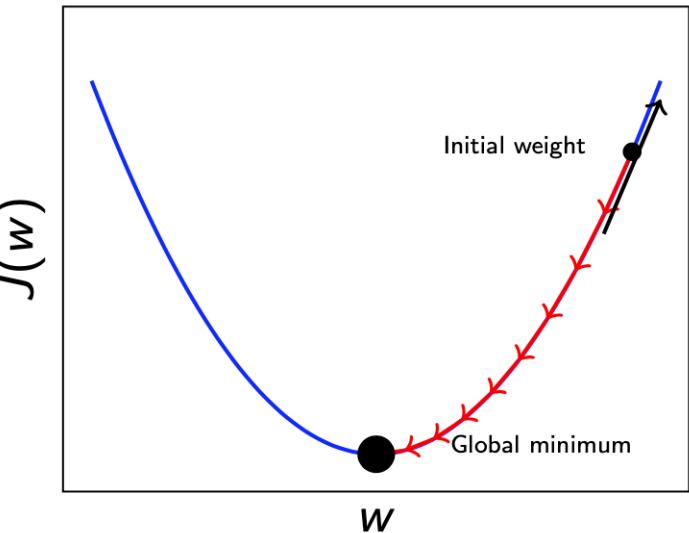
- ▶ L'obiettivo della creazione di un modello è scegliere parametri in modo che $h(x)$ sia vicino a t per i dati di training, x e t .
- ▶ Abbiamo bisogno di una funzione che minimizzi i parametri sul nostro set di dati. Una funzione che è spesso usato è errore quadratico medio,

$$J(w) = \frac{1}{2m} \sum_{i=1}^m (h(x_i) - t_i)^2$$

- ▶ Come troviamo il valore minimo della funzione di loss?

Discesa del gradiente

- ▶ La discesa del gradiente è di gran lunga la strategia di ottimizzazione più popolare, utilizzata al momento nell'apprendimento automatico e nel deep learning.
- ▶ Il costo (errore) è una funzione dei pesi (parametri).
- ▶ Vogliamo ridurre/ridurre al minimo l'errore.
- ▶ Discesa del gradiente: avvicinarsi al minimo errore.
- ▶ Calcola il gradiente, che implica ottenere la direzione verso il errore minimo.
- ▶ Regolare i pesi nella direzione dell'errore minimo.



Discesa del gradiente

- ▶ Abbiamo la seguente ipotesi e dobbiamo adattare i dati di training

$$h(x) = w_0 + w_1 x$$

- ▶ Usiamo una funzione di loss come [Errore quadratico medio](#)

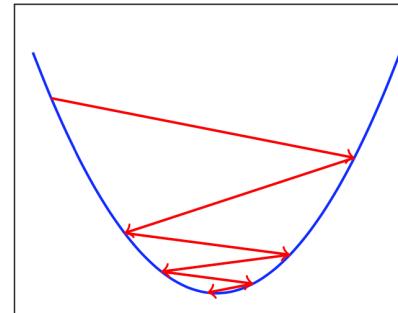
$$J(w) = \frac{1}{2m} \sum_{i=1}^m (h(x_i) - t_i)^2$$

- ▶ Questa funzione di loss può essere ridotta al minimo utilizzando la discesa del gradiente con il [tasso di apprendimento](#) α .

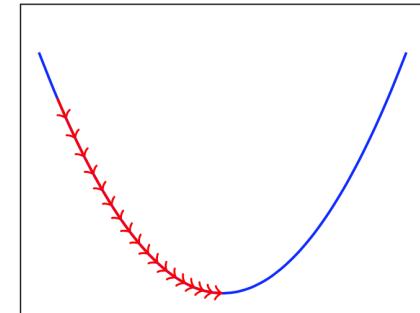
$$w_0^{(t+1)} = w_0^{(t)} - \alpha \frac{\partial J(w^{(t)})}{\partial w_0}$$

$$w_1^{(t+1)} = w_1^{(t)} - \alpha \frac{\partial J(w^{(t)})}{\partial w_1},$$

Big step size

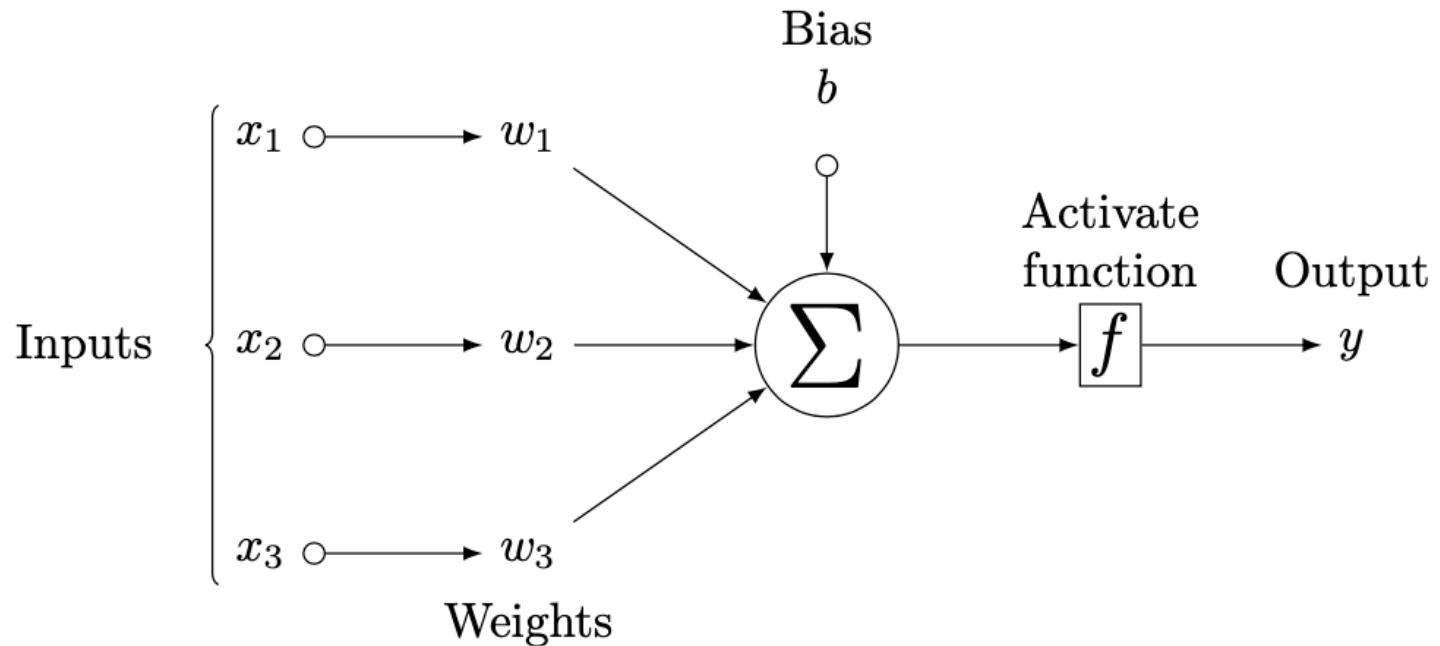


Small step size



Apprendimento basato su gradiente per singola unità

- ▶ Consideriamo il seguente singolo neurone

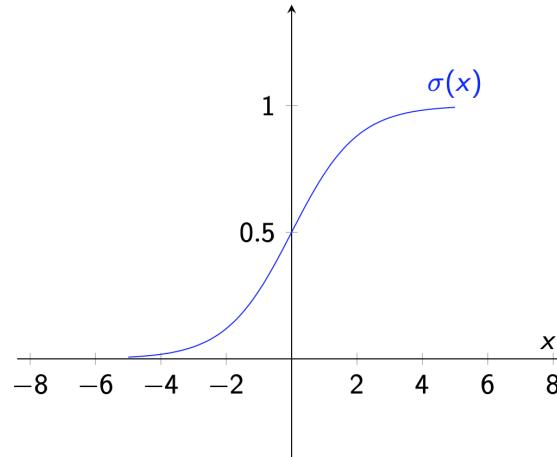


Allenare neurone con attivazione sigmoidea (regressione)

- ▶ Vogliamo addestrare questo neurone per ridurre al minimo la seguente funzione di costo

$$J(w) = \frac{1}{2m} \sum_{i=1}^m (h(x^i) - t^i)^2$$

- ▶ Considerando la funzione di attivazione sigmoidea $f(z) = \frac{1}{1+e^{-z}}$



- ▶ Vogliamo calcolare $\frac{\partial J(w)}{\partial w_i}$

Allenare neurone con attivazione sigmoidea (regressione)

- ▶ Vogliamo calcolare $\frac{\partial J(w)}{\partial w_i}$
- ▶ Usando la **chain rule**, otteniamo

$$\frac{\partial J(w)}{\partial w_j} = \frac{\partial J(w)}{\partial f(z)} \times \frac{\partial f(z)}{\partial z} \times \frac{\partial z}{\partial w_j}$$

$$\frac{\partial J(w)}{\partial f(z^i)} = \frac{1}{m} \sum_{i=1}^m (f(z^i) - t^i)$$

$$\frac{\partial f(z)}{\partial z} = \frac{e^{-z}}{(1 + e^{-z})^2} = f(z)(1 - f(z))$$

$$\frac{\partial z}{\partial w_j} = x^j$$

$$w_j^{(t+1)} = w_j^{(t)} - \alpha \frac{\partial J(w)}{\partial w_j}$$

- ▶ α è il tasso di apprendimento.

Allenare un neurone con attivazione sigmoidea (regressione)

- ▶ Vogliamo addestrare questo neurone per ridurre al minimo la seguente funzione di costo

$$J(w) = \sum_{i=1}^m [-t^i \ln h(x^i) - (1 - t^i) \ln(1 - h(x^i))]$$

- ▶ Calcolando i gradienti di $J(w)$ rispetto a w , otteniamo

$$\nabla J(w) = \sum_{i=1}^m t^i x^i (h(x^i) - t^i)$$

- ▶ L'aggiornamento del vettore di peso utilizzando la regola di discesa del gradiente risulterà

$$w^{(t+1)} = w^{(t)} - \alpha \sum_{i=1}^m t^i x^i (h(x^i) - t^i)$$

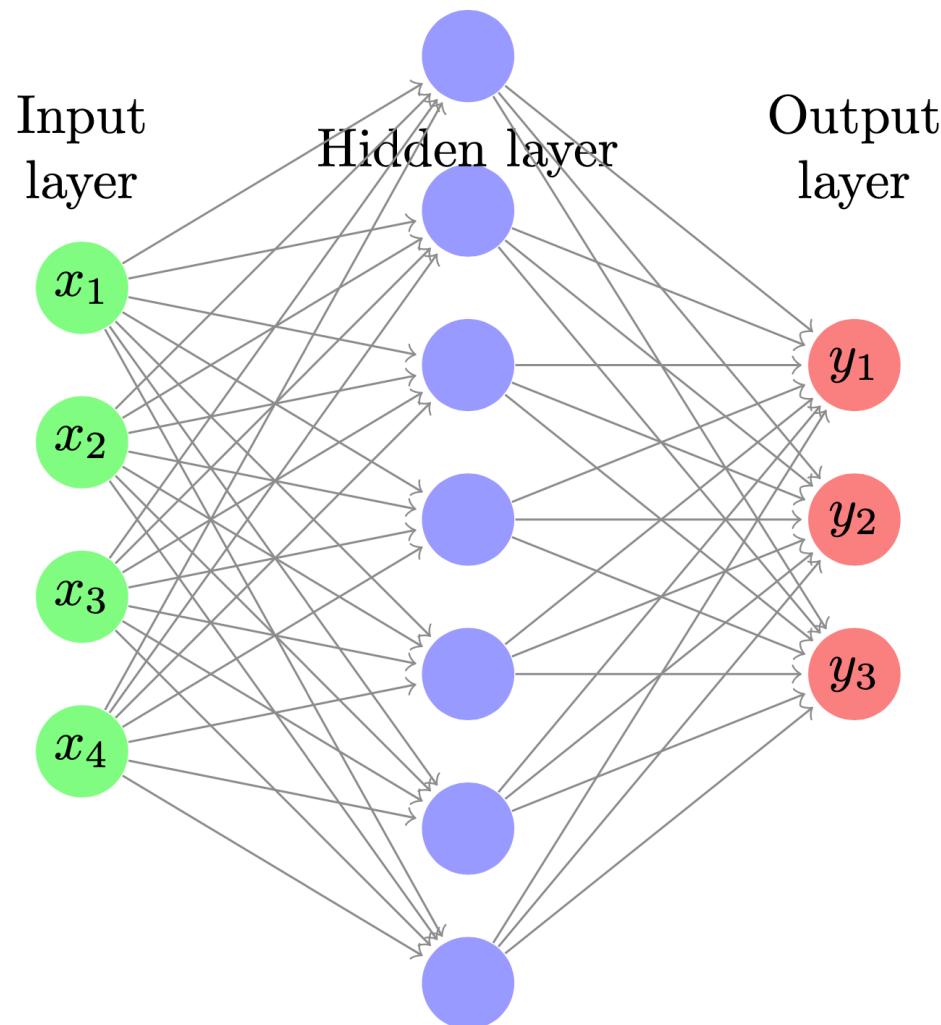
Tuning learning rate (α)

- ▶ Se α è troppo alto, l'algoritmo diverge.
- ▶ Se α è troppo basso, rallenta la convergenza dell'algoritmo.
- ▶ Una pratica comune consiste nel rendere α_k una funzione decrescente del numero di iterazione k . Per esempio.

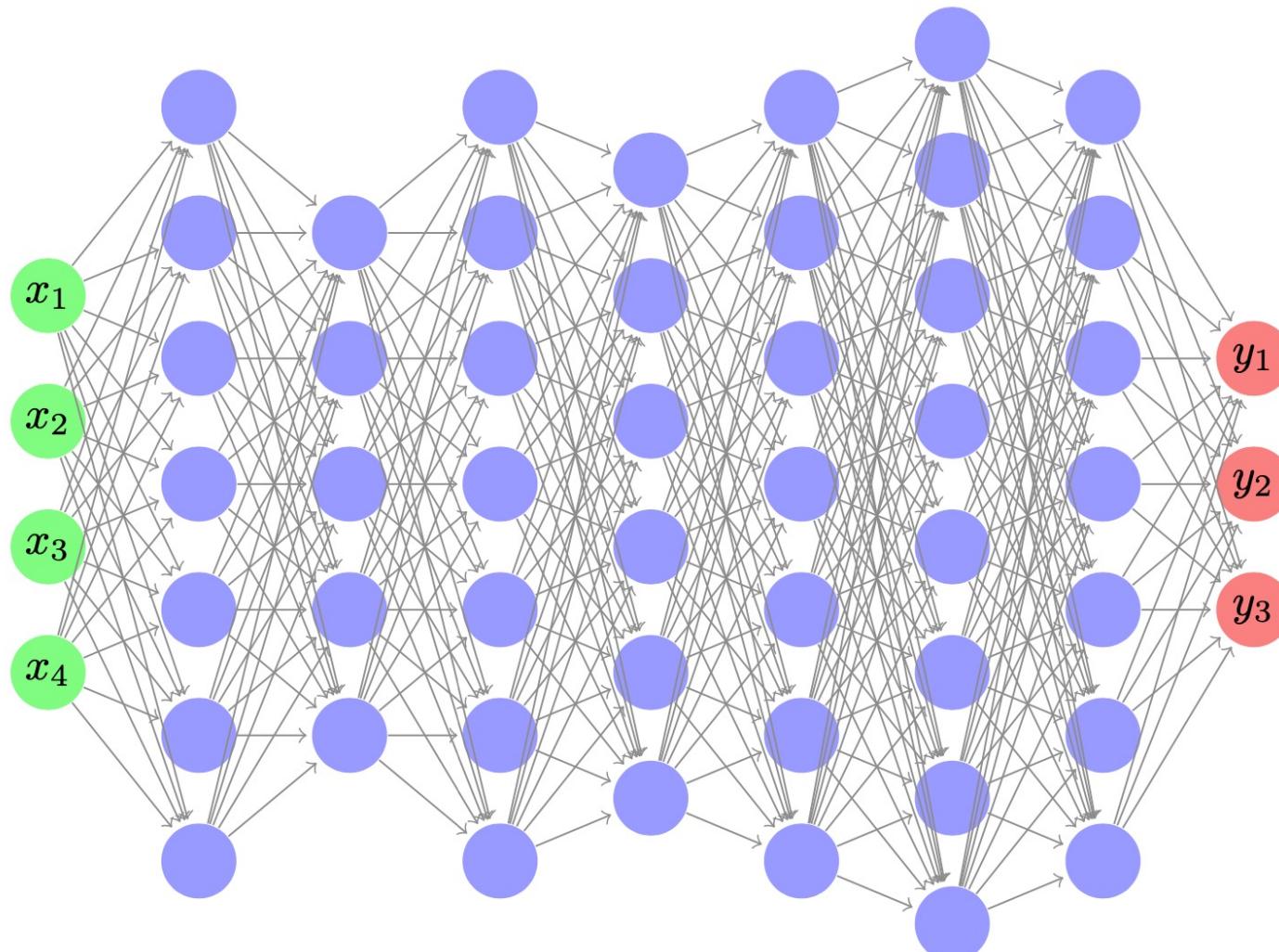
$$\alpha_k = \frac{c_1}{k + c_2}$$

- ▶ dove c_1 e c_2 sono due costanti.
- ▶ Le prime iterazioni causano grandi cambiamenti nella w , mentre le successive effettuano solo il fine-tuning.

Deep feed-forward networks

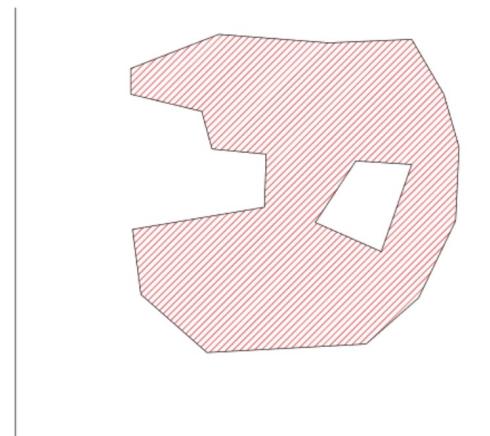
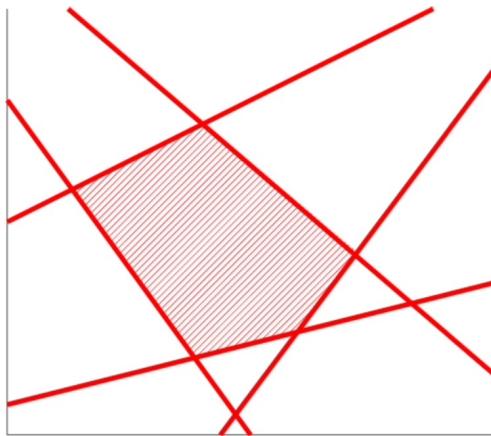


Deep feed-forward networks



La topologia ottimale delle reti

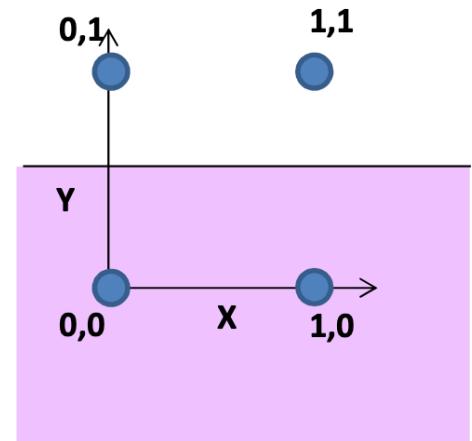
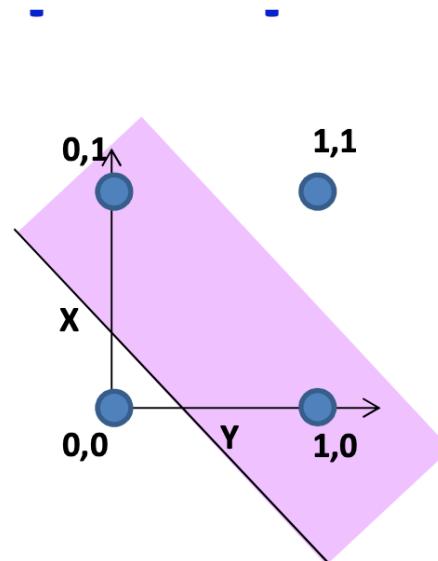
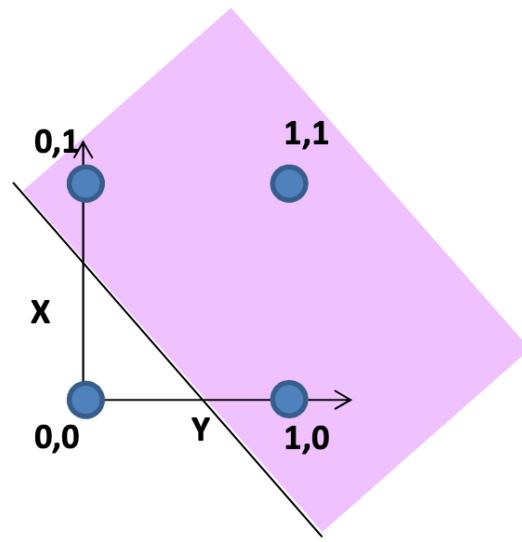
- ▶ Qual è la topologia della rete per il problema dato?
- ▶ Possiamo costruire una rete per creare ogni confine decisionale?
- ▶ Le reti neurali sono approssimatori universali.



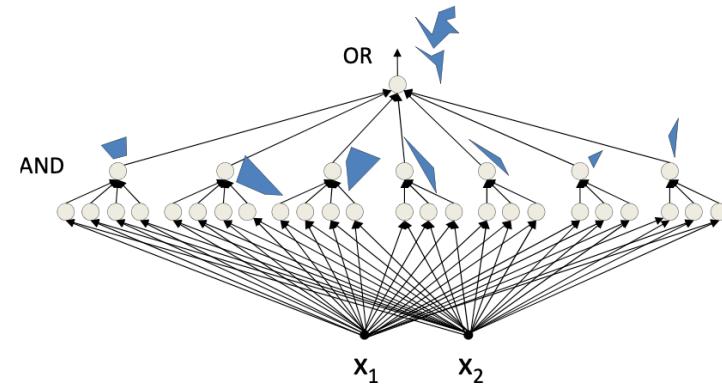
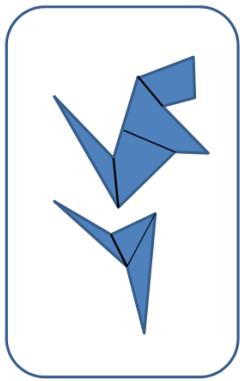
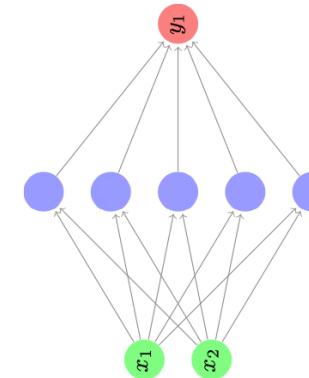
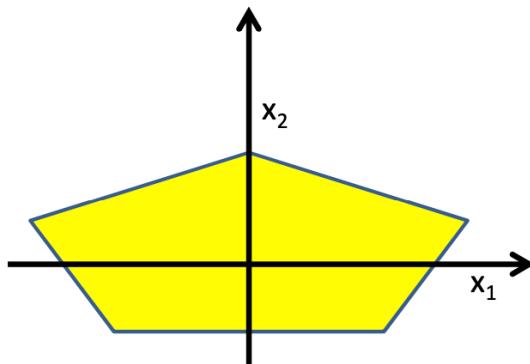
- ▶ Possiamo costruire una rete senza minimi locali in funzione dei costi?

Superficie decisionale del perceptron

- ▶ Qual è la superficie decisionale del percettrone?



Progettazione di reti per confini decisionali più complessi



- ▶ Si può costruire una tale regione con una rete di livelli nascosti?

Training feed-forward networks

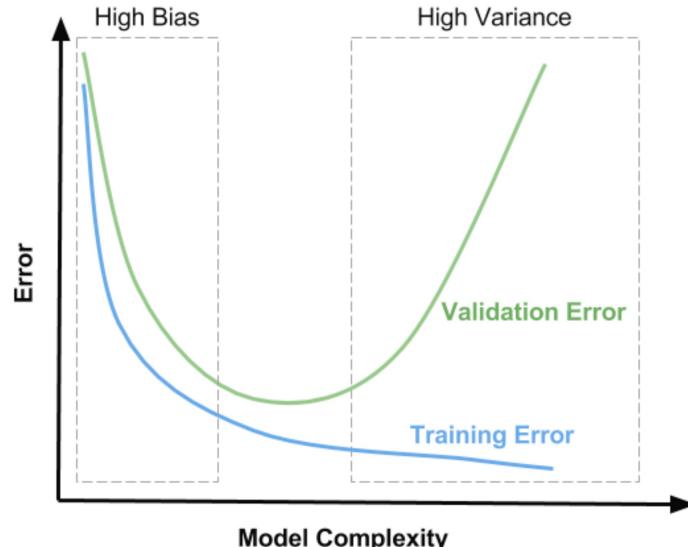
- ▶ Specificare la topologia della rete.
 - ▶ #-strati
 - ▶ #-nodi in ogni livello
 - ▶ funzione di ogni nodo
 - ▶ attivazione di ogni nodo
- ▶ Qual è la topologia della rete per il problema dato?
- ▶ Possiamo costruire una rete per creare ogni confine decisionale?
- ▶ Le reti neurali sono approssimatori universali.
- ▶ Possiamo costruire una rete senza minimi locali in funzione di loss?
- ▶ Specificare la funzione di loss.
- ▶ Usiamo l'algoritmo discesa del gradiente per addestrare la rete.
- ▶ Ma non abbiamo il vero output di ciascuna unità nascosta.

Scelta delle funzioni di attivazione

- ▶ Output layer
 - ▶ Linear activation function
 - ▶ ReLU activation function
 - ▶ Sigmoid activation function
 - ▶ Softmax activation function
- ▶ Hidden layers
 - ▶ Linear activation function
 - ▶ ReLU activation function
 - ▶ Sigmoid activation function

Scelta della topologia di rete

- ▶ Un approccio semplice per la scelta della topologia di rete è per tentativi ed errori (trial and error).
- ▶ Suddividiamo i dati disponibili in tre parti: dati di **training**, dati di **validation** e dati di **testing**.
- ▶ Scegliamo una topologia e addestriamo la rete utilizzando i dati di addestramento.
- ▶ Dopo l'addestramento, valutiamo la rete addestrata utilizzando i dati di convalida.



Backpropagation algorithm

Training a neural network

Data: A training set $S = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)\}$

Result: Weight matrices of the neural network

Initialize randomly weights in the network;

while *not at trained* **do**

 Create a batch $S_B \subseteq S$;

 Let $K \leftarrow |S_B|$;

for $i \leftarrow 1$ **to** K **do**

 Give \mathbf{x}_i to the network and $\hat{\mathbf{y}}_i$;

end

 Compute $J(w)$;

 Compute $\nabla_w J(w)$;

 Compute $w^{t+1} \leftarrow w^t - \alpha \nabla_w J(w)$;

end

Definition (Epoch)

Epoch is defined as one forward pass and one backward pass of all training examples.

Definition (Batch size)

Batch size is defined as the number of training examples in one forward/backward pass.

Definition (Number of iterations)

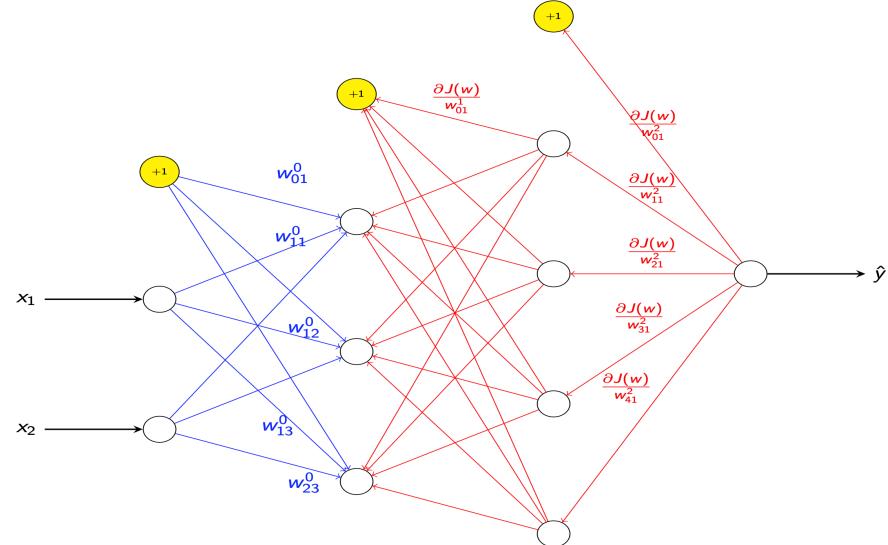
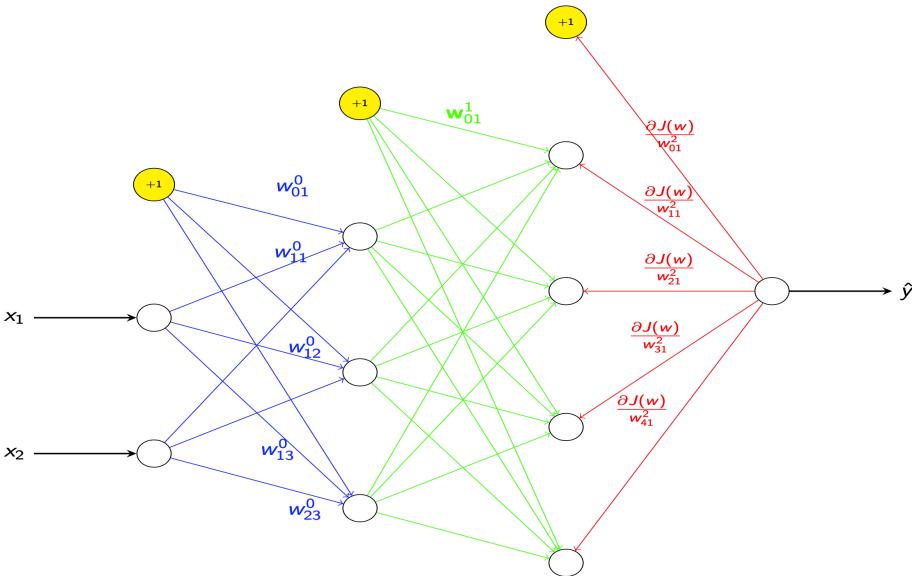
The number of iterations is defined as the number of passes, each pass using the number of examples in the batch.

Batch size

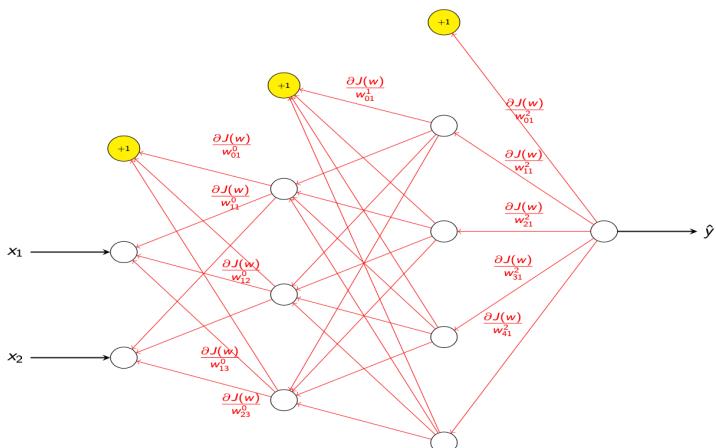
- ▶ La discesa del gradiente può essere utilizzata con batch di diverse dimensioni.
 - ▶ $K = 1$ Discesa stocastica del gradiente
 - ▶ $K \ll m$ Discesa gradiente mini-batch
 - ▶ $K = m$ Discesa gradiente batch
- ▶ Esempio
 - ▶ Sia la dimensione del training set $m = 1000$.
 - ▶ Nella discesa stocastica del gradiente, utilizziamo 1000 batch di dimensione 1.
 - ▶ Sia $K = 50$, in discesa gradiente mini-batch, utilizziamo 20 lotti di dimensione 50.
 - ▶ Nella discesa del gradiente batch, utilizziamo un batch di dimensione 1000.

Backward pass

- Dopo aver calcolato la funzione di loss, dobbiamo calcolare $\nabla_w J(w)$



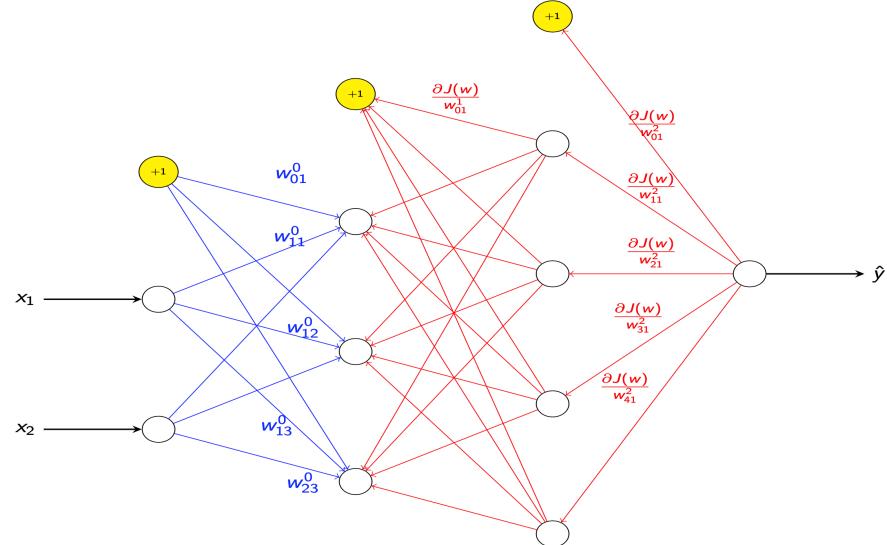
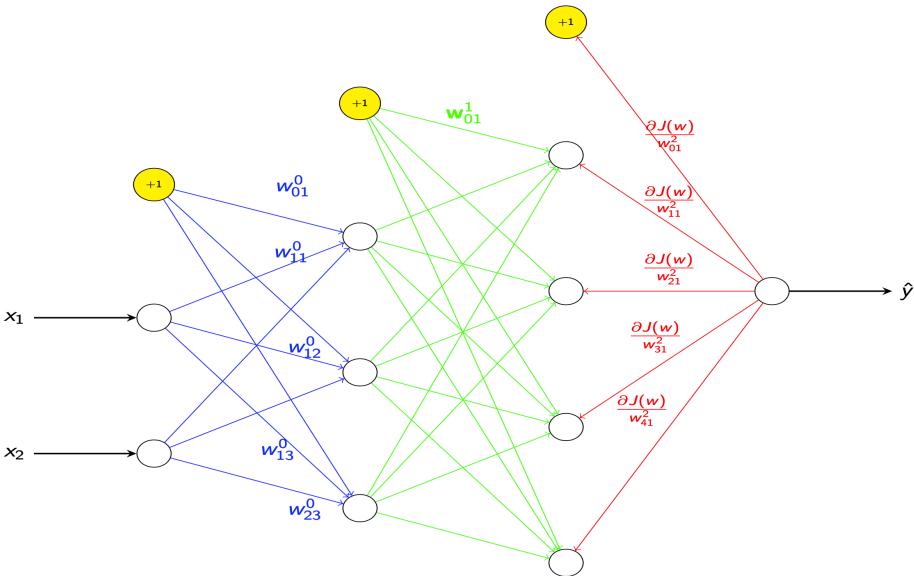
Quindi per ogni peso w , utilizziamo la seguente regola per aggiornare quel peso.



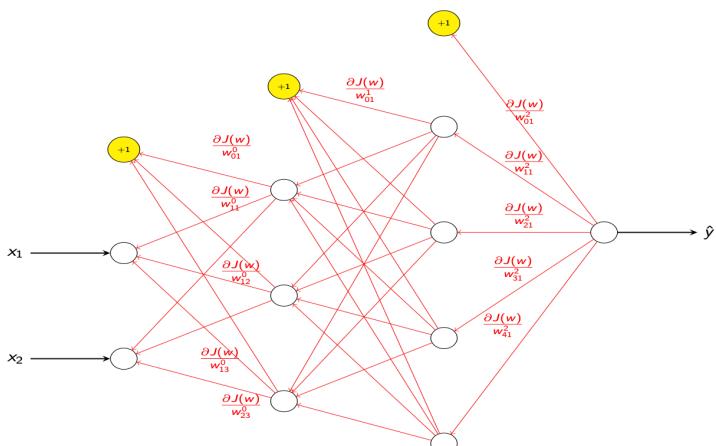
$$w^{t+1} = w^t - \alpha \frac{\partial J(w)}{\partial w}$$

Backward pass

- Dopo aver calcolato la funzione di loss, dobbiamo calcolare $\nabla_w J(w)$



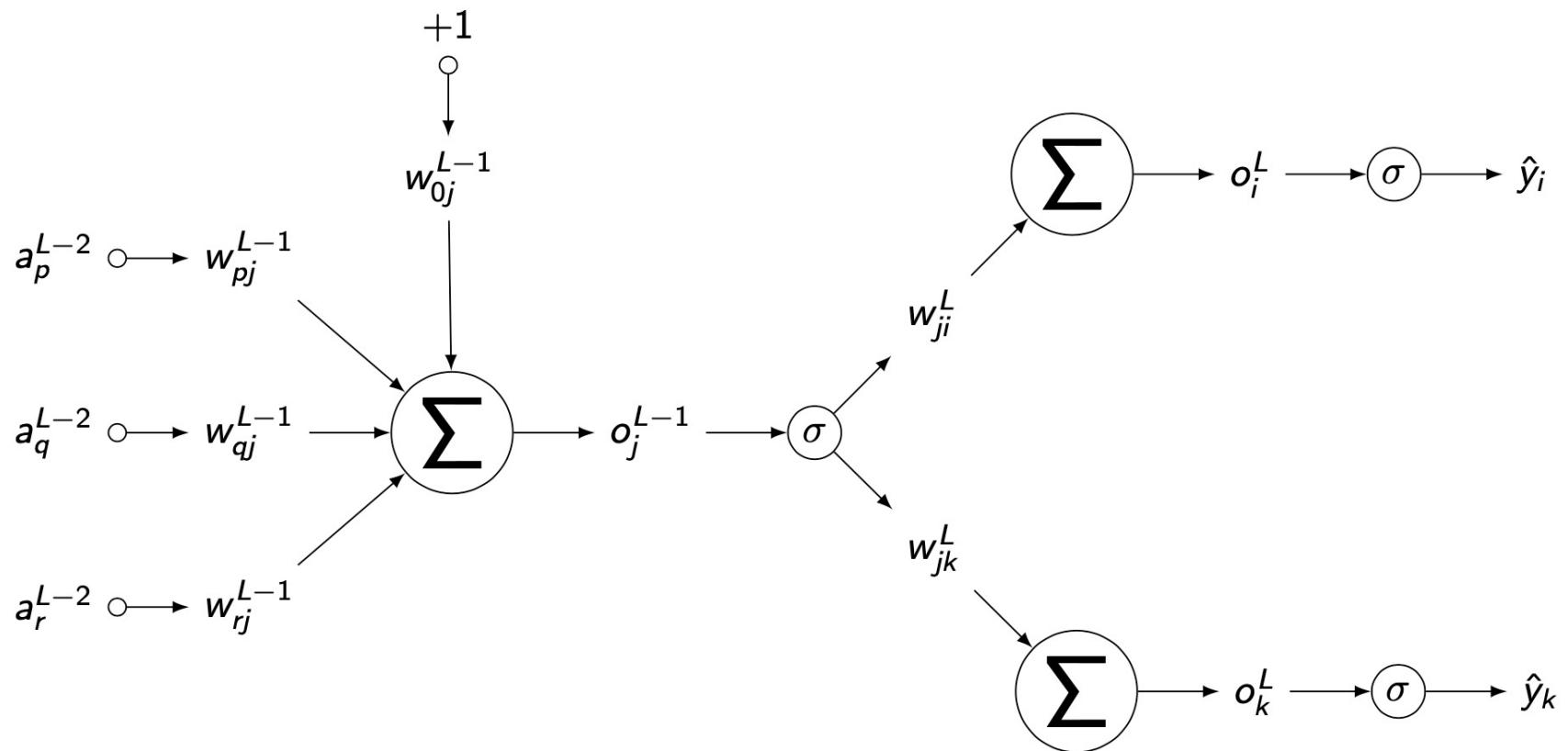
Quindi per ogni peso w , utilizziamo la seguente regola per aggiornare quel peso.



$$w^{t+1} = w^t - \alpha \frac{\partial J(w)}{\partial w}$$

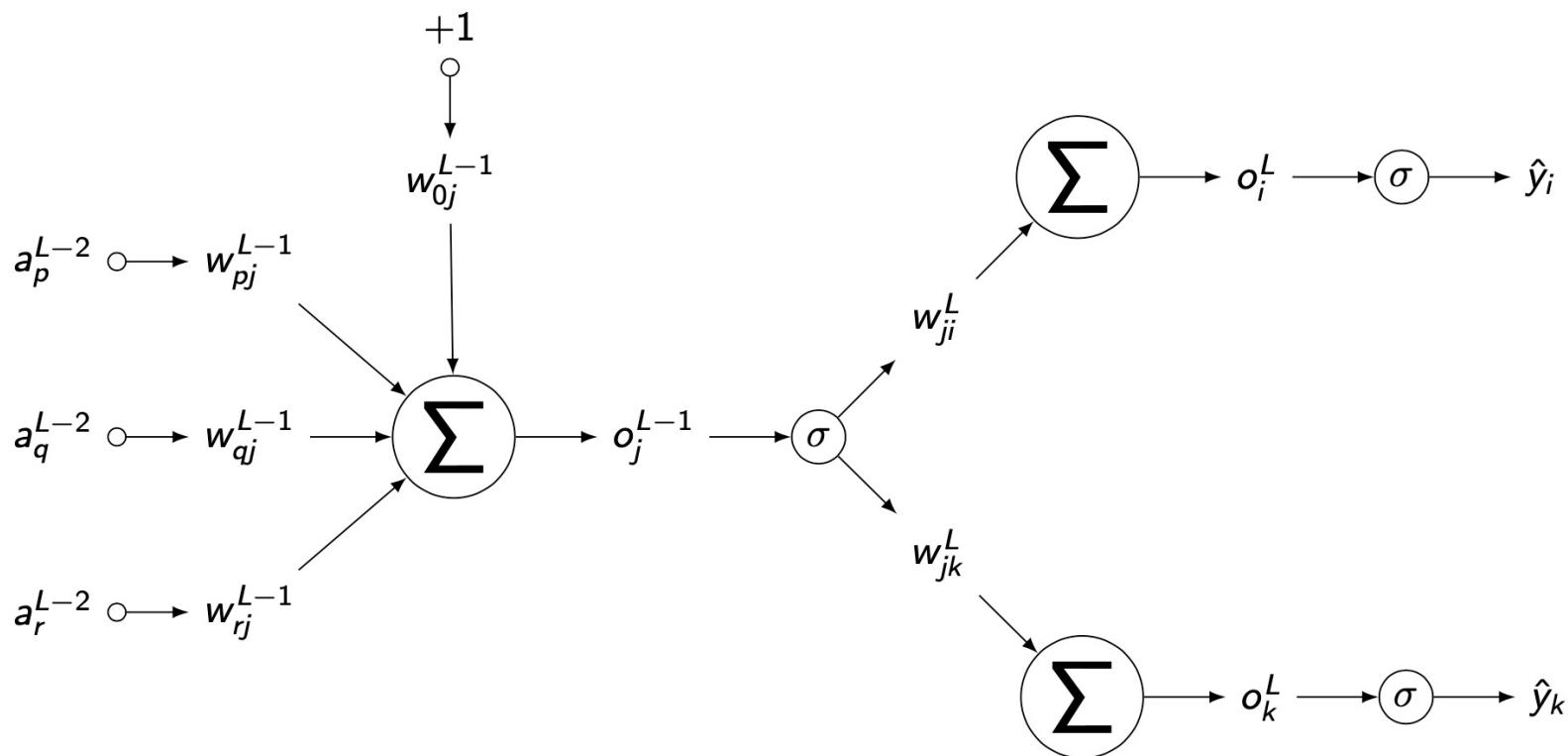
Backward pass

- Usiamo la seguente notazione per il calcolo $\nabla_w J(w)$

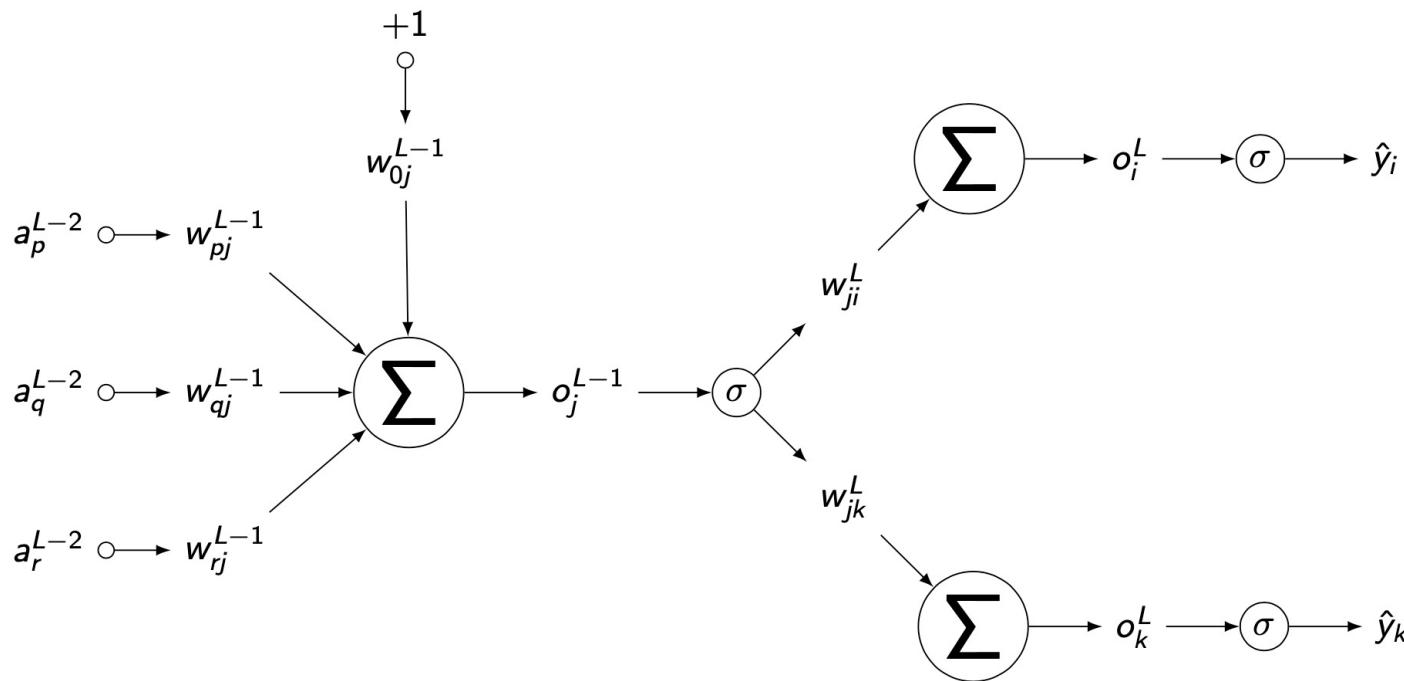


Backward pass

- ▶ Supponiamo di avere una rete con L livelli, in cui l'ultimo livello ha nodi di output C.
- ▶ Supponiamo che tutti i nodi di utilizzano funzioni di attivazione del sigmoide.
- ▶ Indichiamo l'output del j -esimo nodo nel layer L-esimo strato con a_j^L .



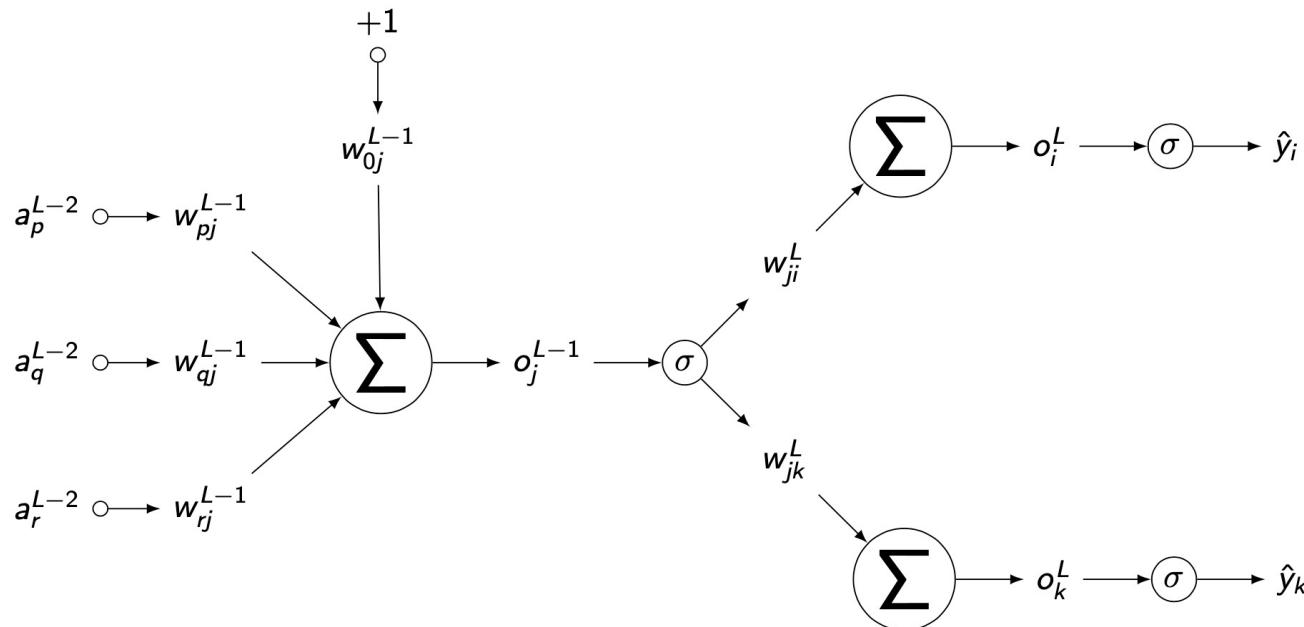
Backward pass



► Dobbiamo calcolare $\frac{\partial J(w)}{\partial w}$

$$\frac{\partial J(w)}{\partial w_{pj}^{L-1}} = \frac{\partial \sum_{s=1}^K \sum_{c=1}^C \ell(\hat{y}_{cs}, y_{cs})}{\partial w_{pj}^{L-1}} = \frac{\partial \sum_{c=1}^C \sum_{s=1}^K \ell(\hat{y}_{cs}, y_{cs})}{\partial w_{pj}^{L-1}} = \sum_{c=1}^C \frac{\partial \sum_{s=1}^K \ell(\hat{y}_{cs}, y_{cs})}{\partial w_{pj}^{L-1}}$$

Backward pass



- Come calcolare $\sum_{c=1}^C \frac{\partial \sum_{s=1}^K \ell(\hat{y}_{cs}, y_{cs})}{\partial w_{pj}^{L-1}}$?

$$\sum_{c=1}^C \frac{\partial \sum_{s=1}^K \ell(\hat{y}_{cs}, y_{cs})}{\partial w_{pj}^{L-1}} = \sum_{c=1}^C \sum_{s=1}^K \frac{\partial \ell(\hat{y}_{cs}, y_{cs})}{\partial \hat{y}_{cs}} \frac{\partial \hat{y}_{cs}}{\partial o_c^L} \frac{\partial o_c^L}{\partial a_j^{L-1}} \frac{\partial a_j^{L-1}}{\partial o_j^{L-1}} \frac{\partial o_j^{L-1}}{\partial w_{pj}^{L-1}}$$

Backward pass

- ▶ Abbiamo

$$\sum_{c=1}^C \frac{\partial \sum_{s=1}^K \ell(\hat{y}_{cs}, y_{cs})}{\partial w_{pj}^{L-1}} = \sum_{c=1}^C \sum_{s=1}^K \frac{\partial \ell(\hat{y}_{cs}, y_{cs})}{\partial \hat{y}_{cs}} \frac{\partial \hat{y}_{cs}}{\partial o_c^L} \frac{\partial o_c^L}{\partial a_j^{L-1}} \frac{\partial a_j^{L-1}}{\partial o_j^{L-1}} \frac{\partial o_j^{L-1}}{\partial w_{pj}^{L-1}}$$

- ▶ Otteniamo

$$\frac{\partial \ell(\hat{y}_{cs}, y_{cs})}{\partial \hat{y}_{cs}} = ?$$

$$\frac{\partial \hat{y}_{cs}}{\partial o_c^L} = \sigma(o_c^L) (1 - \sigma(o_c^L))$$

$$\frac{\partial o_c^L}{\partial a_j^{L-1}} = \frac{\partial \sum_l w_{lc}^L a_l^{L-1}}{\partial a_j^{L-1}} = w_{jk}^L$$

$$\frac{\partial a_j^{L-1}}{\partial o_j^{L-1}} = \sigma(o_j^{L-1}) (1 - \sigma(o_j^{L-1}))$$

$$\frac{\partial o_j^{L-1}}{\partial w_{pj}^{L-1}} = \frac{\partial \sum_l w_{lj}^{L-1} a_l^{L-2}}{\partial w_{pj}^{L-1}} = a_p^{L-2}$$

- ▶ Come generalizzi le equazioni di cui sopra ad altri livelli?

Riassunto

- ▶ Multi-layer Perceptron (MLP) utilizza l'algoritmo di backpropagation dell'errore per propagare l'errore a tutti i livelli.
- ▶ MLP utilizza la discesa del gradiente (stocastico/mini-batch) per aggiornare i pesi.
- ▶ La funzione di loss contiene molti minimi locali e non vi è alcuna garanzia di convergenza.
- ▶ Quanto bene apprende la MLP e come possiamo migliorarla?
- ▶ Quanto bene si generalizzerà MLP (dati di test esterni)?